This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TII.2021.3121783, IEEE Transactions on Industrial Informatics

Juggler-ResNet: A Flexible and High-Speed ResNet Optimization Method for Intrusion Detection System in Software-Defined Industrial Networks

Zongwei Zhu, Wenjie Zhai, Huanghe Liu, Jiawei Geng, Mingliang Zhou, Cheng Ji, Gangyong Jia,

***Abstract*—*ResNets* are widely used in the Intrusion Detection System (IDS) of Software-Defined Industrial Network (SDIN) to construct accurate intelligence detection of network attacks. However, the *resnets*-based IDS has a long detecting interval since the multi-branch architecture exists the fine-grained operator and intermediate results. To address this problem, we propose Juggler-ResNet with a fusible residual structure that preserves the feature extraction ability of the residual structure and enables equivalent transformation to linear topology to support low latency inference service in the industrial application (e.g. malicious network behavior detection, fault diagnosis). First, we propose a fusible multi-branch residual structure to avoid gradient vanishing problems in the training phase. Second, we convert it to liner-topology by using a set of equivalent fusion operators. Finally, the liner-topology model is deployed to accelerate inference speed. Our experimental results on cifar-10 and cifar-100 show that fusible residual structure can achieve 2.08-4.3x acceleration with state-of-the-art level accuracy performance.**

***Index Terms*—Software-Defined Industrial Networks, resnets, intrusion detection system, image classification, optimization method, inference acceleration.**

I. Introduction

In recent years, residual networks (e.g., *resnets*, *resnexts*) have been widely applied in Software-Defined Industrial Network (SDIN) and industrial automation control. Fig. 1 shows the wide usage range of *resnets* in SDIN architecture. On the control plane, to guarantee the security and privacy of SDIN, the current-generation intrusion detection system (IDS) employs *resnets* [1], [2], [3] to detect various attacks includes denial of Service (DOS), user-to-root (U2R), remote-to-login (R2L), and probing. On the infrastructure plane, *resnets* are applied to address typical industrial problems [4] including fault diagnosis [5], quality of service (QoS) optimization, automatic control [6], industrial flotation process [6], [7], etc. Compared to traditional machine learning solutions (e.g., decision trees, support vector machines), *resnets* have shown excellent accuracy in various AI applications in SDIN. However, the inference speed is also of significant importance in various AI-empowered SDIN and industrial applications. Taking the IDS as an example, the long inference delay of *resnets* increases the interval between each network monitoring, causing failure in detecting some critical network attacks. Therefore, it is critical for both SDIN and industrial automatic production and management to accelerate inference speed while ensuring accuracy to guarantee low latency service [5], [8], [9], [10].

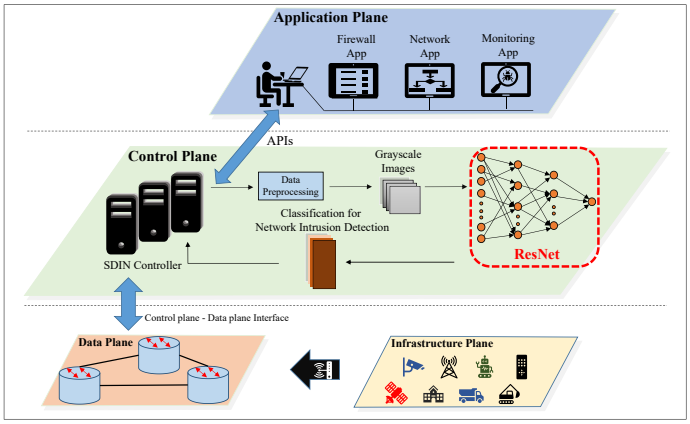


Fig. 1: An illustration of network intrusion detection for the SDIN.

The inference efficiency plays an important role in enhancing the IDS performance of SDIN architecture. Parallelization [11], [12], [13] is a common method to improve inference efficiency. Previous studies mainly focus on two approaches to maximum the parallelization of neural networks. 1) Improving parallelism by scheduling CNN operators. Deep learning frameworks such as TensorFlow [14], PyTorch [15], and TVM [16] schedule operators and optimize input computation graph by performing greedy rule-based on the input graph, which parallelizes arithmetic operations within a single CNN operator. Ding *et al.* proposed IOS [11] to schedule multiple operators through a dynamic programming algorithm to speed up network inference. 2) Transforming the computation graph to increase CNN operator granularity. Jia *et al.* [13] used a backtracking search algorithm to fuse multiple operators matching a specific pattern into a larger operator to improve parallelism. TensorRT[17] not only fused operators but also optimized inference speed on operation systems such as kernel auto-tuning and dynamic tensor allocation. With these methods, it is possible to reduce the overhead of memory access

Zongwei Zhu and Wenjie Zhai contributed equally to this work. (Corresponding authors:Zongwei Zhu;Mingliang Zhou.)

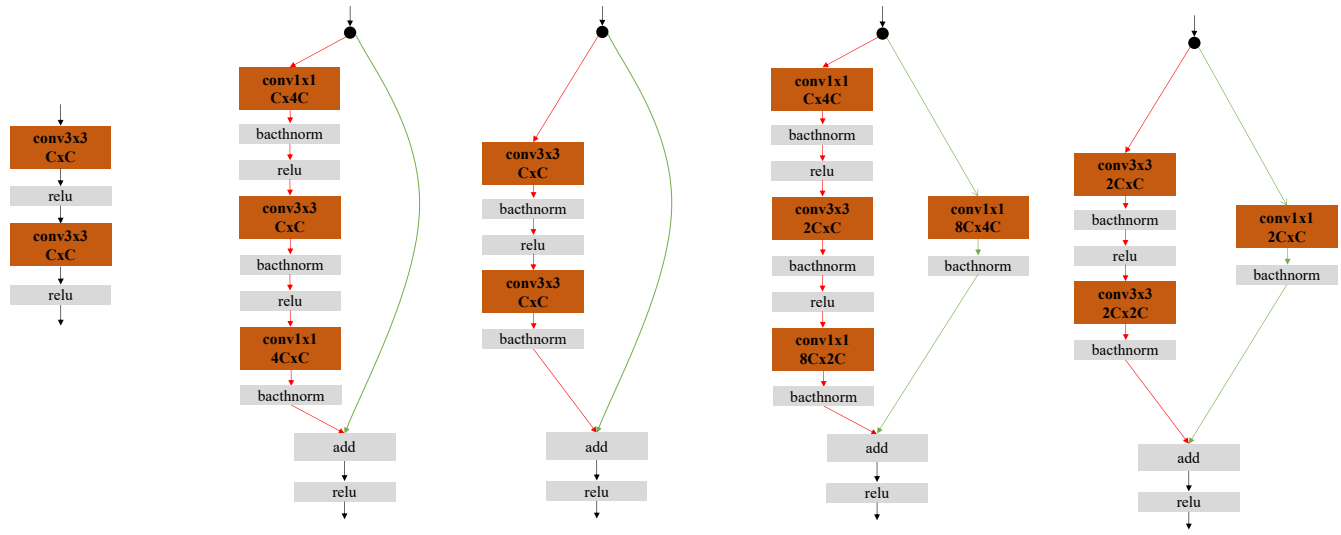
Zongwei Zhu, Wenjie Zhai, Huanghe Liu, Jiawei Geng are with Suzhou Institute for Advanced Research, University of Science and Technology of China, Suzhou, China; e-mail: zzw1988@ustc.edu.cn, {zwjsec, liuhh, gjw1998}@mail.ustc.edu.cn

Mingliang Zhou is with The School of Computer Science, Chongqing University, Chongqing, China; e-mail: mingliangzhou@cqu.edu.cn

Cheng Ji is with Nanjing University of Science and Technology, Nanjing, China; e-mail: cheng.ji@njust.edu.cn

Gangyong Jia is with The Department of Computer Science, Hangzhou Dianzi University, Hangzhou, China; e-mail: gangyong@hdu.edu.cn

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TII.2021.3121783, IEEE Transactions on Industrial Informatics



(a) Vgg connetion (b) Bottleneck without downsample (c) BasicBlock without downsample (d) Bottleneck with downsample (e) BasicBlock with downsample

Fig. 2: Different network connections of *vgg* and *resnet*. Fig. 2. (a) shows the basic modules used in vgg. Fig. 2. (b), (c), (d), and (e) are the basic modules used in *resnets18-152*, where green and red highlight the difference in the data flow from *vgg*. The character ***C*** on convolution layer denotes the number of channel of a convolution layer. For example, ***Cx2C*** means that the number of input channels is ***2C*** and the number of output channels is ***C***.

and kernel context switch. However, these approaches perform their optimizations only on the computation graph (i.e., use an interpreter to interpret the computational graph at runtime), they do not change the overall architecture of networks. Accordingly, the optimizations of computational graphs are still limited by the bottleneck of network architecture.

Two drawbacks in multi-branch architecture make it less efficient in inference. 1) The complicated data flow of multi-branch models (e.g., two-brach in *resnet* and branch-concatenation in *inception*) requires extra memory space for storing intermediate results, which easily led to higher peak memory usage, slower inference speed, and lower memory utilization. 2) Multiple fine-grained CNN operators have a low degree of parallelism. As shown in Fig. 2, owing to multi-branch architecture has multiple fine-grained CNN operators, its data flow is more complex than liner-topology architecture. In contrast, the simpler data flow and the coarse-grained operators make liner-topology architecture more efficient in inference. Although the liner-topology architecture has benefits in the inference phase, its biggest drawback is in the training phase. The liner-topology architecture can not avoid the gradient vanishing problem in training [18], which poses challenges for models with liner-topology architecture to acquire the same accuracy as the multi-branch architecture ones.

Owing to liner-topology has the benefits in inference, and multi-branch architecture is beneficial for training, the proposed Juggler-ResNet takes both advantages of multi-branch and liner-topology architecture. First, it preserves the feature extraction ability of multi-branch architecture with fusible residual structure in the training phase. Second, it can be equivalent transform to liner-topology architecture by a set of fusion operators, thus taking the advantage of liner-topology architecture in the inference phase. Experimental results with a wide range of workloads show that the Juggler-ResNet accelerates inference speed in earns of 2.08-4.4x with SOTA level accuracy performance.

The main contributions of this work are as follows:

- We propose Juggler-ResNet, a flexible and efficient residual network architecture with promising acceleration and SOTA level accuracy performance.
- We establish the fusion operators to decouple the network architecture in the inference phase from the training phase, which takes both advantages of multi-branch architecture in training and liner-topology architecture in inference.
- Experiment results illustrate that the proposed schemes on a real implementation show the effectiveness of Juggler-ResNet in inference.

The remainder of the paper is organized as follows. Section II presents the background and motivation. Section III describes the proposed methodology. Section IV presents the experiments with some discussions. Section V introduces the related work. Section VI concludes this paper.

II. Background and Motivation

This section presents the background of intrusion detection in soft-defined networks and the challenge for CNN adoption in this field.

A. Intrusion Detection System in SDIN

Traditional networks cannot match up to the demands for the qualities of present industrial applications. Therefore, a dynamic and scalable SDIN architecture has emerged. Decoupling the network control plane from the data plane by the controller is the main feature of SDIN, which allows better promoting industrial intelligence applications. Nevertheless,

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TII.2021.3121783, IEEE Transactions on Industrial Informatics

there are various intrusion detection problems pertaining to the control plane toward the SDIN. The intrusion detection systems (IDS) find out whether there are any attacks by collecting information from industrial networks. Many approaches in various IDS have been implemented for defending the network attacks like random forest (RF), logistic regression (LR), Bayesian networks, and support vector machine (SVM). However, such traditional machine learning-based approaches exist with high latency and false alarm rates. Thus, deep learning (DL) is utilized to detect network intrusion attacks. Recently, *resnets*-based IDS has emerged as a new method that ensures low latency and SOTA level accuracy performance.

B. Inference Challenge for SDIN applications

1) Accuracy performance: ImageNet Large-Scale Visual Recognition Challenge (ILSVRC) is one of the most popular and authoritative academic contests in the field of machine vision [19]. We select five most representative models from the annual competition namely *alexnet* [20], *vgg* [21], *google net* [22], *resnet* [23] and *senet* [24]. As shown in Table I, there are clear trends suggesting that more complicated architectures network models were designed to achieve high accuracy performance in competition. Compared to linear topology architectures-based models like *vgg* and *alexnet*, *google net* adopted elaborately designed multi-branch architectures, and *resnet* proposed a two-branch architecture, and *senet* introduced channel attention module into *resnet*. *Google net*, *resnet*, and *senet* outperform liner-topology architecture *alexnet* and *vgg*. The reason is that as the number of layers of the network increases, the linear-topology architecture cannot avoid the gradient vanishing problems. Therefore, it is difficult for a liner-topology model to reach a comparable accuracy as the multi-branch architectures.

TABLE I: The top-5 error rate of *alexnet*, *vgg*, *google net*, and *resnet*. The top-5 error rate is the method of benchmarking CNN models in ILSVRC.

	AlexNet	**VGG**	**Google Net**	**ResNet**	**SENet**
Year	2012	2014	2014	2015	2017
Layers	8	19	22	152	152
Top-5 error	15.4%	7.3%	6.7%	3.57%	2.25%

2) Inference performance: Fig. 2 shows the difference in connection and operators of basic modules between *vgg* and *resnets*. Fig. 2(a) shows the liner topology architecture of *vgg*. Fig. 2(b),(c),(d),(e) show residual structures used in *resnet*. The connection on the right (i.e. shortcut connection) reduces the loss of gradient return and avoids the disappearance of the gradient; Likewise, the add operation can amplify the slight disturbance in the input, making the network training converges faster.

To analyze the differences in their inference efficiency in detail, we carry out our experiment on NVIDIA V100 to count their running time. Table II shows the theoretical floating point operations (FLOPs) and actual running time tested with compute unified device architecture (CUDA) 10.2 on an NVIDIA V100 GPU. As shown in Table II the theoretical FLOPs of *vgg*, *basicblock*, *basicblock+*, *bottleneck*, and *bottleneck+* are almost at the same level. However, the linear-topology architecture *vgg* has the highest theoretical tera float point operations per second (TFLOPS), which is attributed to the fact that liner-topology architecture networks have no extra memory overhead. Moreover, it outperforms other multi-branch architecture models under the same amount of theoretical FLOPs, which is about 17.6%-165.3% faster than the other four models. Therefore, this discrepancy between FLOPs and TFLOPS can be attributed to two important factors: the memory access overhead and the degree of parallelism. Memory access overhead takes a large portion of time usage in two-branches models because two-branches *resnets* requires storing two copies of independent feature maps to perform their calculation (i.e. marked as red and green in Fig. 2). In contrast, liner-topology architecture(Fig. 2 (a)) takes the output from its unique preceding layer as input and feeds the output into its unique following layer, which doesn't need extra memory expense. Notably, Table II shows that *vgg* and *basicblock* have the same amount of FLOPs. But the TFLOPS of *vgg* is higher than *basicblock*, which demonstrates that the performance improvement comes from its liner-topology architecture. On the other hand, models with coarse-grained operators could be faster than other ones with fine-grained operators. Previous studies have shown that coarse-grained operators are beneficial for parallelism. Table II shows that *vgg* and *basicblock* have the highest actual TFLOPS. Both of them only have 3x3 convolutional kernel, commonly abbreviated as conv3x3, whereas other models have more than one conv1x1 operator.

TABLE II: Infenrece time test of basic moudels of *vgg* and *resnets* shown in Fig. 2 with batch size = 32, number of channels $\boldsymbol{C}$ = 256, resolution = 56 on NVIDIA V100. The results of time usage are average of 20 runs after warming up the hardware, where '+' denotes modules with downsample.

Moudels	**Theoretical FLOPs (B)**	**Time usage (ms)**	**Theoretical TFLOPS (B)**
vgg	118.3	7.147	**16.55**
basicblock	118.4	8.312	14.24
basicblock+	92.16	9.793	9.41
bottleneck	112.4	16.845	6.67
bottleneck+	135.04	25.714	5.25

The experiment reveals that liner topology is more efficient than multi-branch architecture in the inference phase. It indicates that simpler data flow and more coarse-grained operators are beneficial for inference. Moreover, The linear topology architecture does not need to store intermediate results, thus reducing memory access and kernel context switch overhead.

3) Convolution kernel performance: The fifth column of Table III illustrates the effect of convolutional kernel size and the number of channels on the number of operations per second (TFLOPS). Among all the convolutional kernels in different numbers of channels and sizes, the conv3x3 has the highest TFLOPS, which is 6.60-6.87. The execution efficiency of conv1x1 is the lowest, which is 4.79-6.05. The TFLOPS of conv5x5 and conv7x7 are almost at the same level, which is 6.06-6.41 and 6.30-6.41, respectively. Notably, conv1x1 is too small to take full advantage of the hardware parallel capability,

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TII.2021.3121783, IEEE Transactions on Industrial Informatics

which causes a significant gap in TFLOPS between it and other large-size convolutional kernels(3x3, 5x5, 7x7). It is 43% slower than the fastest conv3x3. However, there is only a tiny difference between conv3x3, conv5x5, and conv7x7. Therefore, we can conclude that large size convolutional kernels can take full advantage of the hardware parallel capability.

We argue that the advantage of conv3x3 is related to the optimization and padding of conv3x3. For optimization, Winograd [25] is a classical algorithm for accelerating conv3x3, which has been well supported by cuDNN (the NVIDIA CUDA deep neural network library). Such as the standard F(2x2,3x3) Winograd, the floating-point multiplications of conv3x3 are 4/9 of the original ones. On the other hand, in order to ensure that the size of the feature map does not change after the convolution kernel computation, it is common to pad 0 around the feature map. While larger feature maps require more 0 for padding. Therefore, conv5x5 and conv7x7 perform more useless calculations than conv3x3.

The third column of Table III also illustrates the effect of the number of channels on FLOPs. The FLOPs increase with the number of channels. The actual growth pattern of FLOPs is in accordance with Eq. 1. Notably, the FLOPs of the conv7x7 with channel number 2048 exceeds the V100 theoretical computational capability, and thus the execution efficiency decreases by 1.69% compared to the conv7x7 with channel number 1024. In conclusion, when the FLOPs are under the peak of hardware computational capability, the execution efficiency of the convolutional kernel always increases with the increase of FLOPs.

$$Parameters = \sum_{i=0}^{n} Inc_i \times Outc_i \times KernelSize_i^2 \qquad (1)$$

where i represents each convolution kernel and n is the number of convolution kernels in a structure. Inc_i and $Outc_i$ are the input and output channel of convolution kernel i, $KernelSize$ is the size of convolution kernel i.

TABLE III: Inference time of convolution kernel in terms of different kernel size and channel number with resolution = 56x56 and disabling cuDNN optimization on NVIDIA V100. The results of time are the average value of 20 runs after warming up the hardware.

Kernel Size	Channel	FLOPs(B)	Time(ms)	TFLOPS(B)
1x1	512	26.35	5.50	4.79
	1024	105.28	18.47	5.70
	2048	421.12	69.55	6.05
3x3	512	236.82	36.46	6.49
	1024	947.24	141.66	6.69
	2048	3788.48	550.25	6.88
5x5	512	657.62	108.51	6.06
	1024	2630.72	412.73	6.37
	2048	10522.88	1639.18	6.42
7x7	512	1288.96	204.51	6.30
	1024	5156.16	804.13	6.41
	2048	20624.64	3271.01	6.30

C. Parameter Expansion Risk in Models

Reparameterization [26], [27] allows parameters acquired in training to be injected into new network architecture, thus taking advantage of network architecture. Recently, Ding *et al.* proposed ACNet [26], which uses asymmetric conv1x3 and cov3x1 to improve the performance of conv3x3 in training, and merges them into a conv3x3 in deployment to achieve performance improvement. Diverse Branch Block [27] proposed a generic Inception block-like structure to replace original convolutional kernels. However, this technique heavily depends on the original architecture of network models. Directly apply this technique to network architecture without appropriate design may cause parameter expansion, whose cost may override the benefit of transforming network architecture.

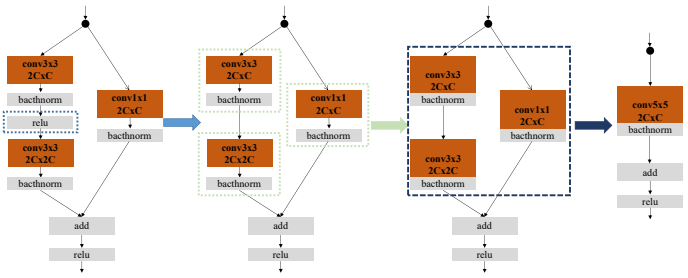


Fig. 3: An illustration of converting *Basicblock* into liner-topology architecture.

For example, converting *basicblock* and *bottleneck* to liner-topology architecture equivalently causes parameter expansion. Specifically, Fig. 3 shows the process of converting *basicblock* into liner-topology architecture. First, it needs to remove the relu layer between two convolutional layer. Next, fusing batchnorm layer into convolutional layer for merging two convolutional layers. Finally, two conv3x3 can be converted to a conv5x5 equivalently [28]. However, the amount of FLOPs of conv5x5 is significantly larger than two conv3x3. Eq.1 shows that the parameters of conv5x5 are 1.4 times larger than that of two conv3x3. Likewise, *bottleneck* has the same problem. Eq.1 shows that the parameter is expanded by 2.76 times after converting *bottleneck* into liner-topology architecture. In this case, the dominant influencing factor of inference efficiency is FLOPs. Our results on NVIDIA V100 show that expanding parameters increases the inference latency by 12.5-20.7%. In summary, directly merging *bottleneck* and *basicblock* significantly increases the parameters, overriding the benefits of converting to liner topology architecture.

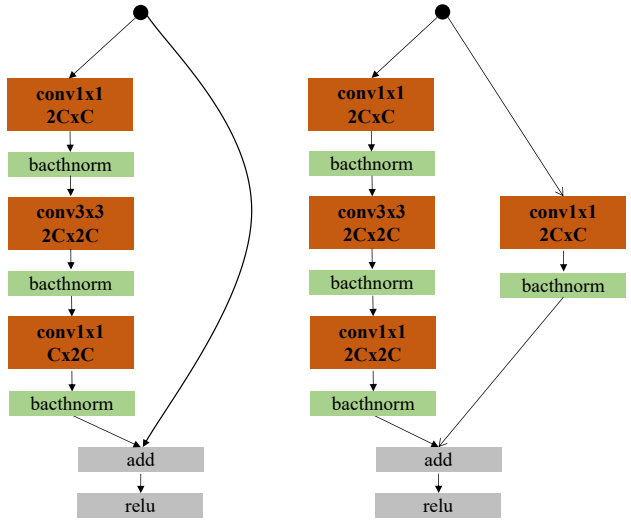
III. METHODOLOGY

A. Overview

Based on the above results, our analysis revealed that 1) Liner topology architecture is more efficient than multi-branch architecture. With the same amount of FLOPs, the large and integral operator is much more efficient than the small and fragmented operator; 2) Liner topology architecture is hard to achieve the same accurate performance as multi-branch architecture. The shortcut connection in *resnet* reduces the loss of gradient return and avoid the disappearance of the gradient; 3) Converting the original residual structure directly to a linear topology architecture leads to parameters expansion problem, which overrides the benefits of converting to liner topology architecture.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TII.2021.3121783, IEEE Transactions on Industrial Informatics

B. Juggler ResNet



(a) Fusible residual structure (b) Fusible residual structure with downsample

Fig. 4: The design of fusible residual structure in Juggler-Resnet.

To overcome the design challenges discussed above, we propose a new residual structure named Juggler-ResNet as shown in Fig. 4. It consists 8-layers network, and 4 different operators, including *convolution*, *batchnorm*, *relu* and *add*. Notably, our structure adopted a two-branch shortcut connection to reduce the loss of gradient return. The two-branch shortcut connection adds cross links with weights between two layers that are not adjacent to each other. The residual structure with downsampling has one more *convolution* and *batchnorm* layer on the shortcut connection. The *add* operation can amplify the slight disturbance in the input, making the network training converges faster. The *relu* layer between *convolution* and *convolution* layer are removed to guarantee equivalent fusion of two *convolution* layers But the last *relu* layer is retained to ensure the non-linear relationship at the structure level. The main difference between Juggler-ResNet and original *resnet* is the design of the *convolution* layer.

As discussed in section.II-C, parameters expansion in the original *resnet* model prevent equivalent merging: 1) The expansion of conv3x3 in *basicblock* increases parameters by 1.39 times 2) The channel expansion in *bottleneck* (Fig. 2 (b),(d)) increases parameters by 2.76 times. Based on this observation, we replace the structures with two conv1x1 and one conv3x3. Because two conv1x1 and one conv3x3 can be equivalent fused into one conv3x3, while the amount of parameters remains unchanged. As for the channel expansion problem, we modify the number of input channels of conv1x1 to C and output channels to 2C. Compared with the number of channels in *bottleneck*, the reason why we choose to widen the channel number at the conv3x3 instead of reducing it is listed as follows: 1) It is only used in training phase, and equivalent liner structure is deployed for accelerating inference. Therefore, the increase in computation doesn't affect the inference phase. 2) Widening the channel number at the conv3x3 reduces the precision degradation caused by removing the relu layer. In summary, widening the channel number of conv3x3 does increase the amount of computation during training, but it also can extract more feature information in the training phase. The parameters size of equivalent liner structure of fusible residual structure is about 1.12 times of *bottleneck* and *basicblock*.

C. Fuse strategies

Based on Juggler-ResNet, we propose three operators to convert it equivalently to liner architecture: 1) convolution layer and batch normalization fusion operator 2) convolution layer vertical fusion operator 3) convolution layer horizontal fusion operator.

1) convolution layer and batch normalization fusion operator: To connect two convolution kernel layers directly for further merging, we first need to fuse the batchnorm layer into convolution kernel layers. Fortunately, it is possible to fuse the batchnorm layer into convolution kernel layers without any extra expense. Feeding the output from convolution kernel layers into the batchnorm layer as input, as shown in the following formula:

$$\begin{aligned} BN(Conv(X)) &= \gamma * \frac{WX + b - mean}{\sqrt{var}} + \beta \\ &= \frac{\gamma * WX}{\sqrt{var}} + \frac{\gamma * (b - mean)}{\sqrt{var}} + \beta \end{aligned} \tag{2}$$

where X is the input parameter matrix. W and b are the weight matrix and bias matrix, respectively. $mean$ and var are the mean and variance of the input matrix X, respectively, and γ and β represent the scaling factor and bias in the normalization layer, respectively. Note that Eq.(1) is the expression of the convolution kernel. We let:

$$W_{fused} = \frac{\gamma * W}{\sqrt{var}} \quad B_{fused} = \frac{\gamma * (b - mean)}{\sqrt{var}} + \beta \tag{3}$$

where W_{fused} and B_{fused} are the parameter matrix and bias matrix after fusion respectively, and $Conv_{fused}$ is the expression of the convolution kernel after batch normalization and convolution kernel fusion. Then, we have the following equivalent representation of convolution kernel layers followed by a batchnorm layer:

$$\begin{aligned} Conv_{fused}(X) &= BN(Conv(X)) \\ &= W_{fused}X + B_{fused} \end{aligned} \tag{4}$$

Therefore, it is feasible to construct a convolution kernel layers equivalent to a convolution kernel followed by a batchnorm layer.

2) convolution layer vertical fusion operator: After fusing batchnorm layer into convolution kernel layer, each convolution kernel layer in Fig. 4(a,b) are directly connected, which means that each convolution kernel layer takes the output from its preceding convolution kernel layer as input and feeds the output into its following convolution kernel layer. Therefore, the following formula holds:

$$\begin{aligned} Conv_2(Conv_1(X)) &= W_2(W_1X + b_1) + b_2 \\ &= W_2W_1X + W_2b_1 + b_2 \\ &= (W_2W_1)X + (W_2b_1 + b_2) \end{aligned} \tag{5}$$

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TII.2021.3121783, IEEE Transactions on Industrial Informatics

Note that Eq.(4) is actually the expression of convolution kernel. We let:

$$W_{fused} = (W_2 W_1) \quad b_{fused} = (W_2 b_1 + b_2) \tag{6}$$

Then, we have the following equivalent representation of convolution kernel layers followed by a convolution kernel layers:

$$Conv_{fused} = W_{fused} X + b_{fused} \tag{7}$$

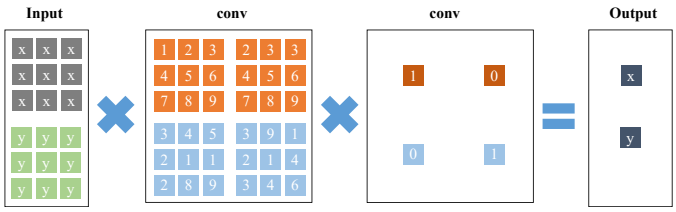


Fig. 5: Illustrations of fusing conv3x3 and conv1x1.

For further explanation, take feature map [1,2,3,3] as input and feed it into a conv3x3 layer which is directly connected to a conv1x1, and the output is [1,2,1,1]. The actual computing process is shown in Fig. 5. It can be seen as feeding the parameters of conv3x3 as input into conv1x1 and use the output as parameters to convolute the input.

3) convolution layer horizontal fusion operator: The convolution layer horizontal fusion operator can fuse the conv1x1 on the shortcut connection shown in Fig. 4. (b). To merge horizontally, we first need to expand conv1x1 to conv3x3 to match the size. Conv1x1 can be seen as a special case of conv3x3, which means it can be represented by conv3x3. As shown in Fig. 6, conv1x1 expands to conv3x3 by padding zero around the conv1x1. Then, it is possible to merge horizontally two conv3x3 into one conv3x3 by adding the conv3x3 onto the central point of expanded conv3x3.

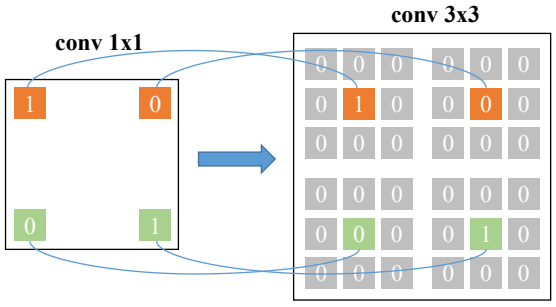


Fig. 6: Equivalent conv3x3 representation of conv1x1 convolution kernel.

With the three fusion operators described above, we can convert Juggler-ResNet to linear architecture for inferencing. As shown in Algorithm 1, the main process is as follows:

- Traverse all fusible residual structures and use convolution layer and batch normalization fusion operator to merge the batch normalization layer into convolution kernels.
- We use convolution layer vertical fusion operator to merge convolution kernel one after another.

Algorithm 1 Convert Juggler-ResNet into liner-topology architecture

Require: A multi-branch architecture Juggler-ResNet G_0, a list of fusible residual structure $\{g_1, g_2, g_3 ..... g_n\}$ of Juggler-ResNet and each g_i has convolution kernels layers $L = \{conv_1, conv_2, conv_3, sconv_4\}$, three fusion operator $F_{batch}(.)$ in III-C1, $F_{conv}(.)$ in III-C2 and $F_{expand}(.)$ in III-C3.

Ensure: Liner-topology architecture Juggler-ResNet G_1.

// $sconv_4$ represent the convolution kernel on the shortcut
// connection in a fusible residual structure g_i with downsampling.

for $i = 1, \cdots, n$ **do**
  $g_i = F_{batch}(g_i)$
  $Conv_m = F_{conv}(g_i.conv_1, g_i.conv_2)$
  $Conv_m = F_{conv}(Conv_m, g_i.conv_3)$
  if $g_i.sconv4 \neq \emptyset$ **then**
    $Conv_e = F_{expand}(g_i.sconv4)$
    $Conv_m = Conv_m + Conv_e$
  end if
  $g_i.L = Conv_m$
end for
return G_1

- For a fusible residual structure with downsampling, use convolution layer horizontal fusion operator to expand the conv1x1 on shortcut connection into conv3x3. Next, add the central point of expanded conv3x3 onto conv3x3.

IV. Experiments

A. Experiments setup

Platform setting: We deploy Juggler-ResNet on embedded platform NVIDIA Tx2 and server equipped with NVIDIA V100 GPU. The detailed configuration of the experiment machine is shown in Table IV.

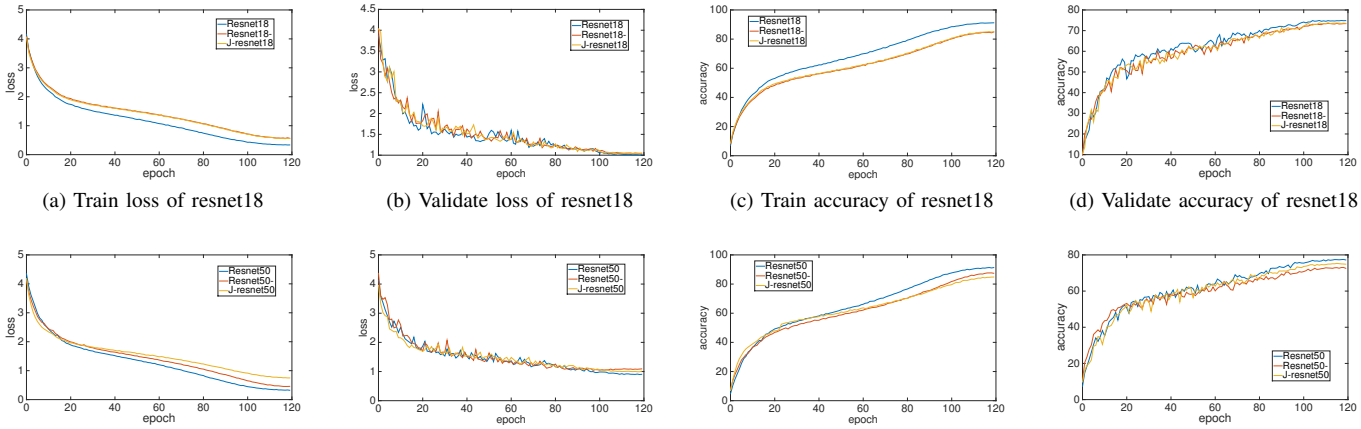
Models and datasets: Five original *resnet* models (i.e. *resnet-18*, *resnet-34*, *resnet-50*, *resnet-101* and *resnet-18*) are used to compare their accuracy and inference performance with Juggler-ResNet. The train and validate datasets are CIFAR-100 and CIFAR-10 which are widely used in CNN models [29].

Deep learning framework: To evaluate the differences in convergence speed and accuracy between Juggler-ResNets and *resnets*, ***Pytorch*** is used as the training framework. To compare the inference performance of Juggler-ResNets and *resnets* on different frameworks, ***TensorRT*** and ***Pytorch*** are used as inference computing framework.

TABLE IV: Configuration of the experiment

	Server	NVIDIA Tx2
OS	Ubuntu 16.04 Xenial	Ubuntur 18.04 bionic
CPU	2*Intel Xeon E5-2620 v4	Quad ARM A57
GPU	2*NVIDIA Tesla V100	NVIDIA Pascal 256 CUDA cores
RAM	256GB DDR4	8GB 128-bit LPDDR4

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TII.2021.3121783, IEEE Transactions on Industrial Informatics



(a) Train loss of resnet18 (b) Validate loss of resnet18 (c) Train accuracy of resnet18 (d) Validate accuracy of resnet18

(e) Train loss of resnet50 (f) Validate loss of resnet50 (g) Train accuracy of resnet50 (h) Validate accuracy of resnet50

Fig. 7: The experimental results of Resnet and Juggler-Resnet on *CIFAR-100* dataset with learning rate = 0.1, batch size = 256, standard SGD with momentum coefficient = 0.9, and cosine annealing for 120 epochs with 5-epochs warmup. The *J-resnet*$_i$ denotes the Juggler-Resnet in different depth, and the *Resnet*$_i$- denotes *resnets* without relu layer inserted between two adjacent convolution kernels. Train loss and validate loss are the losses on the training and validation datasets, respectively.

B. Experiments results

1) Training performance: Fig. 7(a),(b),(e),(f) shows that Juggler-ResNet can be applied to *resnet* in different depths, and the convergence speed is almost the same. Fig. 7(c),(g),(d),(h) shows that the Juggler-ResNet suffers a slight decrease of accuracy, which is about 1.4%-1.79% lower than original *resnet*. *This precision degradation can be solved by enhancing the training data, which is out of the scope of this paper*. Moreover, Juggler-ResNet is about 0.7%-0.9% higher than *resnet* without relu layer (i.e. marked '-' in the figure). It indicated that widening the channel number at the conv3x3 can reduce the precision degradation caused by removing the relu layer. Notably, with the same number of training epochs, *resnet* achieves higher accuracy on train set than Juggler-ResNet, which is 6.11% higher on *resnet18* and 12.03% higher on *resnet50*. This huge accuracy gap between the train set and validate set shows that Juggler-ResNet has better generalization capabilities than *resnet* under the same training epochs.

To further verify the accuracy performance of Juggler-ResNet on different datasets and *resnet* in different depth, we compare Juggler-ResNet with *resnet* in different depth ranges, from *resnet18* to *resnet152*. Table V shows that the precision degradation grows as the depth increases. The precision gap between Juggler-ResNet and *resnet* reaches the maximum at *resnet152*, which is about 1.41% on the CIFAR-10 dataset and 1.91% on the CIFAR-100 dataset. The superior performance of proposed schemes stems from the following contributions. 1) As the depth grows, relu layers can reduce the dependence of parameters by setting part of parameters to zero. 2) CIFAR-100 dataset is more complicated than the CIFAR-10 dataset. However, Juggler-ResNet still outperforms other liner topology architecture-based models like vgg and alexnet by an average of 10.9% on the CIFAR-10 and 39.78% on the CIFAR-100 dataset. It indicates that Juggler-ResNet preserves the feature extraction ability of the multi-branch architecture, reaching SOTA level accuracy performance compared to liner topology architecture-based models.

TABLE V: The comparison results on *Cifar10* and *Cifar100* trained in 120 epochs. The *Acc1* denotes the top-1 accuracy on validate dataset.

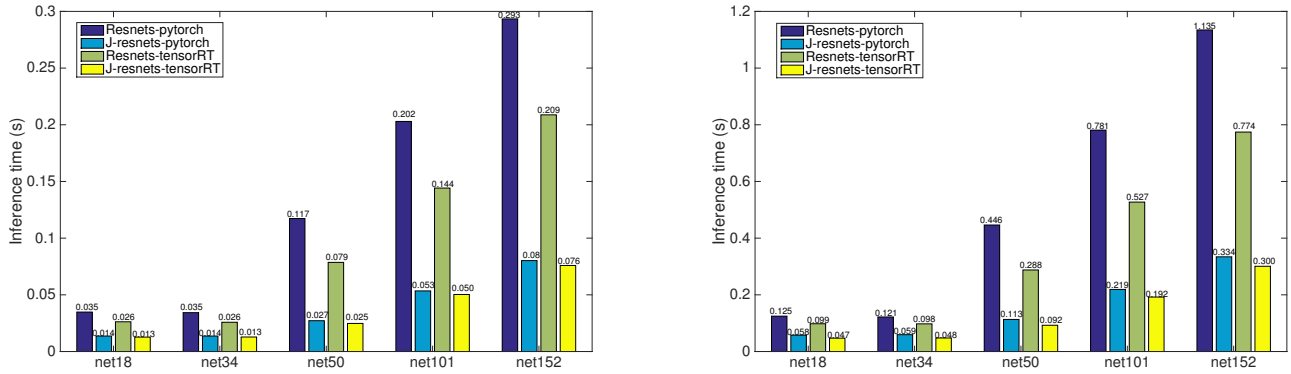
Model	CIFAR-10 Acc1	Gap	CIFAR-100 Acc1	Gap
Resnet18	94.49%	0.97%	73.92%	1.40%
J-resnet18	93.52%		72.52%	
Resnet34	94.63%	1.51%	75.86%	1.01%
J-resnet34	93.12%		74.85%	
Resnet50	94.90%	1.48%	76.52%	1.79%
J-resnet50	93.42%		74.73%	
Resnet101	94.73%	1.11%	76.85%	1.38%
J-resnet101	93.62%		75.47 %	
Resnet152	94.36%	1.41%	77.52%	1.91%
J-resnet152	92.95%		75.61%	

2) Inference acceleration: To fully evaluate Juggler-ResNet, we measure the acceleration effect by deploying Juggler-ResNet on different inference computing frameworks and hardware platforms.

Fig. 8(a),(b) shows that Juggler-ResNet can achieve up to 3.95x acceleration on NVIDIA Tx2 and 4.31x on NVIDIA V100 compared with original *resnet*. These results show that the acceleration of Juggler-ResNet is suitable for not only platforms with limited hardware resources but also platforms with abundant hardware resources. Notably, the acceleration effect increase as the network depth grows. Because the deeper the network, the higher the memory access and kernel system call and context switch overhead. As a result, the benefits of simple data flow of liner topology architecture become more apparent in deep depth networks compared with original *resnets*.

In addition, Juggler-ResNet achieves both accelerations on different inference computing frameworks as shown in Fig. 8(a),(b). Under different hardware conditions, it accelerates most significantly on the *Pytorh* framework, which is about 2.05-4.31x faster than the original *resnet*. Even

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TII.2021.3121783, IEEE Transactions on Industrial Informatics



(a) Time usage on NVIDIA V100

(b) Time usage on NVIDIA Tx2

Fig. 8: The inference performance comparison of *Juggler-ResNet* and *ResNet* between different inference computing frameworks and different hardware platforms. The results of time usage are the average of 100 runs after warming up the hardware with input images of 3 channels and 256 resolutions. The parameter setting are batch size = 1 on Tx2 and batch size = 8 on NVIDIA V100 GPU.

on the current-generation inference acceleration framework TensorRT, Juggler-ResNet outperforms original *resnets* by up to 3.17x. Notably, comparing the acceleration effect of Juggler-ResNet on PyTorch and Tensorrt frameworks, TensorRT is only 1.06-1.22x faster than PyTorch, which is almost the same level of inference speed. Because all inference computing frameworks perform rule-based graph transformations optimization before executing it. This result indicates that there are few optimization rooms for Juggler-ResNet since the performance improvement actually comes from its liner topology architecture.

3) Throughput: Given that resnet152 has the largest amount of parameters, which can show both the performance of our optimization and the inflection point of the hardware computational capability. Therefore, we choose it as a comparison model. As indicated in Fig. 9, it is obvious that the curves of both networks have almost the same upward trend as the batchsize grows. However, the throughput of Juggler-ResNet152 significantly outperforms that of resnet152 on a wide range of batchsize settings. On Tx2, the peak throughput of Juggler-ResNet152 is 1.38 times higher than that of resnet152. This gap is even more pronounced on V100, where the Juggler-ResNet152 on V100 has 3.22 times the peak throughput of the resnet152.

In Fig. 9, it can be observed that Juggler-ResNet152 is able to reach the peak throughput quickly on different hardware platforms. For example, on V100, the Juggler-ResNet152 with linear topology reaches the peak throughput for V100 and Tx2 hardware platforms at $batchsize = 10$ and $batchsize = 36$, respectively. The multi-branched resnet152, however, reaches peak throughput at $batchsize = 64$ and $batchsize = 48$. Moreover, we observe that resnet152 shows a significant throughput plunge at $bathsize = 2^n + 2$. For example, the throughput decreases by 9.93% for $bathsize = 18$ vs. $bathsize = 16$ on Tx2 and by 6.51% for $bathsize = 18$ vs. $bathsize = 16$ on V100. This makes the throughput curve of resnet152, in the graph, show a clear jagged upward trend. The Juggler-ResNet152 overall curve, however, is smoother and does not show a significant throughput plunge at the $bathsize = 2^n + 2$ setting. Therefore, Juggler-ResNet152 reaches the optimal throughput faster and grows more steadily than resnet152 when the batchsize increase.

V. RELATED WORK

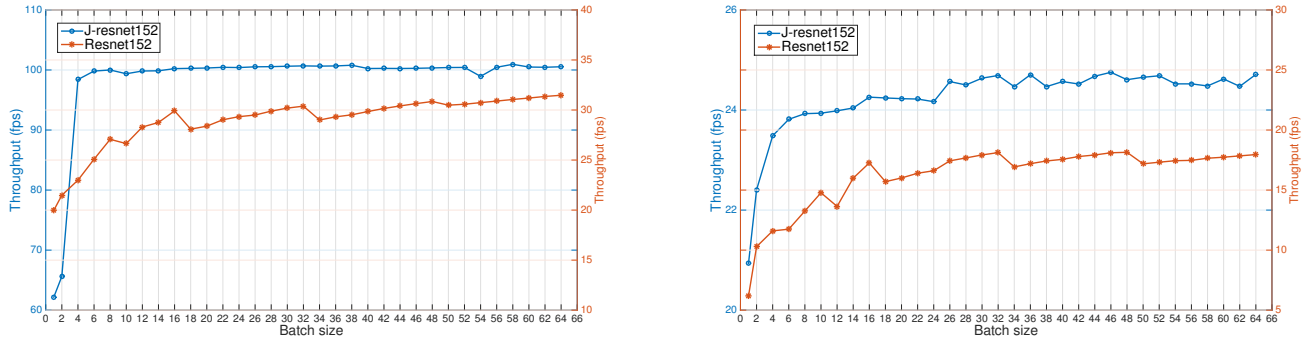
A. Residual networks in SDIN

Network intrusion detection system (NIDS) has been evolving over years. The current-generation NIDS employs Residual networks [1], [2], [3], [30] (e.g *resnet34*,*resnet50*) to improve the detection performance on various attacks includes DOS, U2R, R2L and PROBING. By transforming the NSL-KDD dataset into images [1], residual networks can learn critical features of different network attacks. Previous studies have shown that residual networks achieve excellent results on the KDD-NSL dataset that outperforms traditional machine learning (e.g., SVM, decision tree) based solutions. However, the inference delay of residual networks is significantly longer than ML solutions, which increases the interval between each network monitoring. Therefore, the aforementioned works still endure the bottleneck of inference delay of residual networks. To timely identify attacks and prompt counter-attack actions, it is necessary to accelerate inference speed.

B. Efficient accleration for residual networks

One way to accelerate inference speed is to schedule CNN operators to increase parallelism. Tang *et al.* [12] propose a greedy strategy that directly executes all available CNN operators on multicore CPU to fully utilize the hardware. TensorFlow and PyTorch execute arithmetic operations within a single operator in parallel. Another way is transforming computing graphs to optimize CNN architectures. Jia *et al.* [13] merges multiple operators into a larger one to enable more

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TII.2021.3121783, IEEE Transactions on Industrial Informatics



(a) The throughput on NVIDIA V100

(b) The throughput on NVIDIA Tx2

Fig. 9: The throughput of Juggler-ResNet152 and ResNet152 on different hardware platforms under different batch size settings. The result is the average of 100 runs after warming up the hardware with input images of 3 channels and 256 resolutions on V100 and 56 resolutions on Tx2.

parallelism. However, the study described above is performing optimization on multi-branch architecture CNNs. They do not change the CNN overall architecture, that is, the multi-branch architecture remains unchanged after optimization. Few works focus on decoupling the network architecture in the inference phase from the training phase to accelerate inference efficiency.

VI. Conclusion

This paper proposed Juggler-ResNet, a flexible and efficient residual network architecture. First, it preserves the feature extraction ability of multi-branch architecture with fusible residual structure in the training phase. Second, it can be equivalent transform to liner-topology architecture by a set of fusion operators, thus taking the advantages of liner-topology architecture in inference. Experimental results with a wide range of workloads show that Juggler-ResNet can achieve acceleration on the different hardware platforms and neural network inference computing frameworks. Compared with traditional network architecture, Juggler-ResNet takes both advantages of multi-branch and liner-topology architecture. Juggler-ResNet can effectively shorten the intrusion detection interval in IDS of SDIN while maintaining SOTA-level accuracy performance. In future work, we will consider combining Juggler-ResNet with other model compression methods to broaden the scope of application to storage resources constrained platforms.

Acknowledgment

This work was partially supported by National Natural Science Foundation of China (62102390, 62102179, 62176027), China Postdoctoral Science Foundation (No. 2020M671637), National Science Youth Fund of Jiangsu Province (No. BK20190224), and Jiangsu Postdoctoral Science Foundation (No. 2019K224).

References

[1] M. Masum, H. Shahriar, and H. M. Haddad, "A transfer learning with deep neural network approach for network intrusion detection," 2021.

[2] J. Song, B. Li, Y. Wu, Y. Shi, and A. Li, "Real: A new resnet-alstm based intrusion detection system for the internet of energy," in *2020 IEEE 45th Conference on Local Computer Networks (LCN)*. IEEE, 2020, pp. 491–496.

[3] J. Man and G. Sun, "A residual learning-based network intrusion detection system," *Security and Communication Networks*, vol. 2021, 2021.

[4] M. Latah and L. Toker, "Artificial intelligence enabled software defined networking: A comprehensive overview," *Iet Networks*, 2018.

[5] M. Zhao, S. Zhong, X. Fu, B. Tang, and M. Pecht, "Deep residual shrinkage networks for fault diagnosis," *IEEE Transactions on Industrial Informatics*, vol. 16, no. 7, pp. 4681–4690, 2020.

[6] J. Liu, S. Zhao, Y. Xie, W. Gui, Z. Tang, T. Ma, and J. P. Niyoyita, "Learning local gabor pattern-based discriminative dictionary of froth images for flotation process working condition monitoring," *IEEE Transactions on Industrial Informatics*, vol. 17, no. 7, pp. 4437–4448, 2021.

[7] Y. Fu and C. Aldrich, "Flotation froth image recognition with convolutional neural networks," *Minerals Engineering*, vol. 132, pp. 183–190, 2019.

[8] G. Liu, L. Zhu, W. Yu, and W. Yu, "Image formation, deep learning, and physical implication of multiple time-series one-dimensional signals: Method and application," *IEEE Transactions on Industrial Informatics*, vol. 17, no. 7, pp. 4566–4574, 2021.

[9] M. D. Putro, L. Kurnianggoro, and K. H. Jo, "High performance and efficient real-time face detector on central processing unit based on convolutional neural network," *IEEE Transactions on Industrial Informatics*, vol. 17, no. 7, pp. 4449–4457, 2021.

[10] S. Jiang, Y. Qi, H. Zhang, Z. Bai, X. Lu, and P. Wang, "D3d: Dual 3-d convolutional network for real-time action recognition," *IEEE Transactions on Industrial Informatics*, vol. 17, no. 7, 2021.

[11] Y. Ding, L. Zhu, Z. Jia, G. Pekhimenko, and S. Han, "Ios: Inter-operator scheduler for cnn acceleration," 2020.

[12] Y. Liu, Y. Wang, R. Yu, M. Li, V. Sharma, and Y. Wang, "Optimizing cnn model inference on cpus," 2018.

[13] Z. Jia, "Optimizing dnn computation with relaxed graph substitutions."

[14] M. Abadi, P. Barham, J. Chen, Z. Chen, and X. Zhang, "Tensorflow: A system for large-scale machine learning," *USENIX Association*, 2016.

[15] PyTorch, "Tensors and Dynamic neural networks in Python with strong Gpu acceleration," Website, 2017, https://pytorch. org.

[16] T. Chen, T. Moreau, Z. Jiang, H. Shen, and A. Krishnamurthy, "Tvm: End-to-end optimization stack for deep learning," 2018.

[17] NVIDIA, "Nvidia tensorrt: Programmable inference accelerator," Website, 2017, https://developer.nvidia.com/tensorrt.

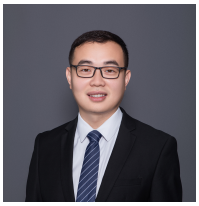
[18] A. Veit, M. Wilber, and S. Belongie, "Residual networks are exponential ensembles of relatively shallow networks," *Advances in Neural Information Processing Systems*, 2016.

[19] "Imagenet large scale visual recognition challenge," Website, 2017, ISVRC, 2017.

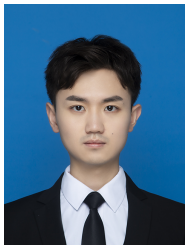
[20] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," in *International Conference on Neural Information Processing Systems*, 2012.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TII.2021.3121783, IEEE Transactions on Industrial Informatics

[21] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," *Computer Science*, 2014.
[22] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, and A. Rabinovich, "Going deeper with convolutions," *IEEE Computer Society*, 2014.
[23] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," *IEEE*, 2016.
[24] H. Jie, S. Li, S. Gang, and S. Albanie, "Squeeze-and-excitation networks," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. PP, no. 99, 2017.
[25] A. Lavin and S. Gray, "Fast algorithms for convolutional neural networks," in *2016 IEEE CVPR 2016*. IEEE Computer Society, 2016, pp. 4013–4021.
[26] X. Ding, Y. Guo, G. Ding, and J. Han, "Acnet: Strengthening the kernel skeletons for powerful cnn via asymmetric convolution blocks," 2019.
[27] X. Ding, X. Zhang, J. Han, and G. Ding, "Diverse branch block: Building a convolution as an inception-like unit," 2021.
[28] C. Szegedy, V. Vanhoucke, S. Ioffe, J. Shlens, and Z. Wojna, "Rethinking the inception architecture for computer vision," *IEEE*, pp. 2818–2826, 2016.
[29] A. Krizhevsky, G. Hinton *et al.*, "Learning multiple layers of features from tiny images," 2009.
[30] R. Yu, D. Ye, Z. Wang, B. Zhang, A. M. Oguti, J. Li, B. Jin, and F. Kurdahi, "Cffnn: Cross feature fusion neural network for collaborative filtering," *IEEE Transactions on Knowledge and Data Engineering*, 2021.



Zongwei Zhu received his M.S. and Ph.D. degrees in Computer Science from the University of Science and Technology of China (USTC), in 2011 and 2014, respectively. From 2014 to 2016, he was a research assistant at the IoT Perception Mine Research Center of China University of Mining and Technology. From 2016 to 2018, he worked as a senior engineer in the Huawei Company. Currently, he is a research assistant at the Suzhou Institute for Advanced Research of USTC. His research focuses on resource scheduling, memory, power and operating systems.



Wenjie Zhai received his B.S. degree from the School of Computer Science, Southwest Petroleum University, in 2019. He is currently working toward the M.S. degree in the Department of Software Engineering at the University of Science and Technology of China (USTC). His research interests include machine vision, nonvolatile memory, and hybrid memory system.

Huanghe Liu received his B.S. degree in Computer Science from Sichuan University in 2017 and M.S. degree from University of Science and Technology of China (USTC) in 2021. His research focuses on machine vision, embedded systems and operating systems.

Jiawei Geng received his B.S. degree from the School of Management Science and Engineering, Dongbei University Of Finance And Economics, in 2020. He is currently working toward the M.S. degree in the Department of Software Engineering at the University of Science and Technology of China (USTC). His research interests include edge computing, deep neural networks, and resource scheduling.

Mingliang Zhou received the Ph.D. degree in computer science from Beihang University, Beijing, China, in 2017. He was a post-doctoral in the Department of Computer Science, City University of Hong Kong, Hong Kong, China, from Sep. 2017 to Sep. 2019. He is currently a lecturer with the School of Computer Science, Chongqing University. His research interests include image and video coding, perceptual image processing, multimedia signal processing, rate control, multimedia communication, machine learning, and optimization.

Cheng Ji received his M.E. degree from the School of Computer Science and Technology, University of Science and Technology of China (USTC) in 2014, and the Ph.D. degree from the Department of Computer Science, City University of Hong Kong, in 2018. He is now an associate professor with the School of Computer Science and Engineering, Nanjing University of Science and Technology. His research interests include embedded systems, nonvolatile memory, and operating systems.

Gangyong Jia (Member, IEEE) received Ph.D. degree from the Department of Computer Science, University of Science and Technology of China (USTC), Hefei, in 2013. He is currently an Associate Professor with the Department of Computer Science, Hangzhou Dianzi University, Hangzhou, China. His current research interests are IoT, cloud computing, edge computing, power management, and operating system.