GhostShiftAddNet: More Features from Energy-Efficient Operations

Jia Bi
J.Bi@soton.ac.uk
Jonathon Hare
jsh2@ecs.soton.ac.uk
Geoff V. Merrett
gvm@ecs.soton.ac.uk

Electronics and Computer Science
University of Southampton
Southampton, UK

Abstract

Deep convolutional neural networks (CNNs) are computationally and memory intensive. In CNNs, intensive multiplication can have resource implications that may challenge the ability for effective deployment of inference on resource-constrained edge devices. This paper proposes GhostShiftAddNet, where the motivation is to implement a hardware-efficient deep network: a multiplication-free CNN with fewer redundant features. We introduce a new bottleneck block, GhostSA, that converts all multiplications in the block to cheap operations. The bottleneck uses an appropriate number of bit-shift filters to process intrinsic feature maps, then applies a series of transformations that consist of bit-wise shifts with addition operations to generate more feature maps that fully learn to capture information underlying intrinsic features. We schedule the number of bit-shift and addition operations for different hardware platforms. We conduct extensive experiments and ablation studies with desktop and embedded (Jetson Nano) devices for implementation and measurements. We demonstrate the proposed GhostSA block can replace bottleneck blocks in the backbone of state-of-the-art networks architectures and gives improved performance on image classification benchmarks. Further, our GhostShiftAddNet can achieve higher classification accuracy with fewer FLOPs and parameters (reduced by up to 3×) than GhostNet. When compared to GhostNet, inference latency on the Jetson Nano is improved by 1.3× and 2× on the GPU and CPU respectively. Code is available open-source on https://github.com/JIABI/GhostShiftAddNet.

1 Introduction

Deep Convolutional Neural Networks (CNNs) have become more accurate and faster for image classification applications with large image datasets. However, traditional CNN networks with many parameters and floating point operations (FLOPs) are problematic to deploy on resource-constrained hardware platforms within specific application scenarios (e.g. low latency requirements are a priority for autonomous driving systems). To address this problem, many portable networks, e.g., ShuffleNet [13, 23], MobileNet [8, 9, 16], and CNNs

based on shift or addition operation (e.g., ShiftNet [21], AdderNet [2]) have been proposed, requiring fewer FLOPs and parameters. However, in practice, these methods are inefficient to implement onto different hardware platforms for two reasons: the structure of convolutional components and operation selection in CNNs.

Firstly, three important factors in achieving a hardware efficient network are a small number of FLOPs and parameters, and a low inference latency. However, common convolution components in CNNs, including Spatial Convolution (SConv), Depth Separable Convolution (DWSConv), and Shift Convolution (ShiftConv), cannot satisfy these three factors concurrently. For example, because SConv requires a considerable number of FLOPs, they are inefficient and limited to use on compute-bound hardware platforms (e.g. CPU) [3]. Compared with SConv, DWSConv and ShiftConv require fewer FLOPs and parameters but cannot effectively reduce inference latency, especially on memory-bound platforms (e.g. GPU). The main reason for this is that DWSConv and ShiftConv require more memory accesses than computation [3, 7, 13]. As a result, a first question naturally arises: *What is the best structure of the convolutional component in CNNs, which allows the network to run efficiently on CPU- and memory-bound platforms?*

Secondly, CNNs contain many multiplications, and their high computational load prevents them from running on embedded platforms with limited power budgets. Therefore, considerable research has aimed to replace multiplication operations with "cheap" operations (e.g., shift and addition operations) [22]. Although addition operations have significantly fewer FLOPs than multiplications, we observe that CNNs based on addition operations may have a longer training process and higher inference latency than CNNs based on multiplication. This is for two possible reasons: Firstly, during the training process, the loss function of CNNs that use addition is measured by the ℓ_1 distance and the magnitude of the variance of the gradient of the loss function is larger than that of multiplication. This leads to slower convergence of the network and lower accuracy [2]. Secondly, addition operations in floating point (FP) format may actually have higher inference latency than multiplication operations; following the IEEE754 FP standard [10], according to the difference between the two exponents, the FP addition operation needs to align the two mantissas to be added. This may require multiple variable shifts before the adder. The result of mantissa addition then needs to be renormalized, also requiring many shifts to format the result correctly. Therefore, compared to multiplication, adding two mantissas may require higher gate delays and line delays. As a result, if aiming to use cheap operations to replace multiplications in CNNs, a second question is posed: *what is the efficient structure of operations in a convolutional component?*

These two issues prompted us to propose a novel CNN *GhostShiftAddNet* (GhostSANet), a lightweight network topology with hardware-friendly convolution operations, which can be efficiently implement on embedded CPUs and GPUs. In particular, we designed a new module called *GhostShiftAdd* (GhostSA), partly inspired by GhostNet [6] and ShiftAddNet [22], which divides the convolutional output layer into two parts, as shown in red and blue in Figure 1. In this figure, some bit-wise shift operations identify the entire features of the input layer (called "intrinsic" features) and convolve them into the red part. Given the intrinsic features from the red part, the blue part can be obtained by using the modified DWSConv and cheap operations to generate more "ghost" features based on the intrinsic features. The output feature consists of concatenating two parts, the size of which has not changed. Given the size of the output layer, a hyper-parameter γ balances the number of intrinsic and ghost features in the output layer so that the number of ghost features can be controlled to enrich the intrinsic feature information from the input layer, and feature redundancy can also be

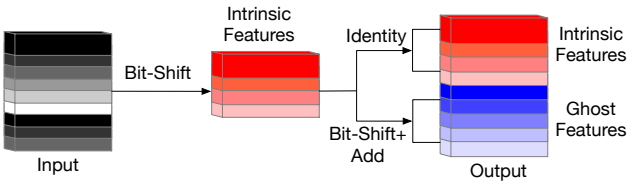

Figure 1: An illustration of proposed GhostSA module for outputting the same number of features. SConv applies the number of bit-shift filters to generate intrinsic features from the input features. Then a modified DWSConv utilises bit-wise shift and addition operations [22] to generate a series of ghost features.

avoided. Consequently, our contributions can be summarized as follow:

1. A novel multiplication-free module *GhostShiftAdd* (GhostSA) uses a hyper-parameter γ to control the number of bit-wise shift and addition operations in a convolutional layer. The module can be easily applied to any CNN architecture. This is the first work to concurrently reduce the FLOPs, memory requirements and inference latency of both traditional and portable CNNs on general-purpose and embedded hardware platforms.

2. The GhostSA module provides a modified DWSConvt. The improved DWSConvt uses bit-wise operation in the depth-wise layer to quickly parallel shift features of separable channels. Then, it uses addition operations in the point-wise layer to combine the feature information from all channels and adjust the parameters to avoid the accuracy loss through the shift operation in the depth layer. Our theoretical results show that the GhostSA module can significantly reduce FLOPs, memory requirements, and inference latency of multiplication-based spatial convolutions by $k^2\gamma\times$ (typical values are normally a filter kernel size $k = 3$ and $\gamma >= 2$).

3. Based on the GhostSA module, we present *GhostShiftAddNet* (GhostSANet), a novel hardware-efficient neural architecture. We demonstrate the effectiveness and efficiency of GhostSANet on visual datasets, comparing with SOTA benchmarks including both traditional CNNs (such as VGG, ResNets) and portable CNN (such as ShuffleNet, MobileNet, GhostNet). Our experimental results show that the GhostSA module reduces the number of parameters and FLOPs (reduced by $3\times$) in the SOTA backbone network, reduces training time and inference latency (reduced GPU latency by $1.5\times$), and maintains or even improves image classification accuracy (>1%). In addition, the CPU and GPU latency of GhostSANet on an embedded platform (Nvidia Jetson Nano) is reduced by $2\times$ and $1.3\times$, respectively.

2 Related Work

Multiplication-less Deep Networks. DWSConv is a method to reduce the FLOPs by decoupling SConv into spatial and cross-channel convolution processes, which requires fewer multiplications. Alternatively, there are many works which use other operations with fewer FLOPs to replace multiplications in CNNs. For example, ShiftNet [21] generates feature

maps by assigning one value in a $k \times k$ convolution kernel as 1, and the rest as 0, thereby using the shift operation to perform convolution. Chen et al. [3] proposes to further reduce FLOPs by removing unnecessary shifts in ShiftNet. DeepShift [4] uses bit-wise and sign-shifting instead of a standard shift during training, and only requires at most 5 bits to represent the weight. However, DeepShift reduces the relatively high classification accuracy of inference, especially on large-scale tasks (e.g., ImageNet datasets). To solve this problem, ShiftAddNet [22] maintains an accuracy similar to the original network by combining the bit-shift and addition operations proposed by AdderNet [2]. He et al. [7] observe that shift operations with fewer FLOPs do not always lead to a shorter inference latency, and hence three shift-based primitives are proposed to reduce GPU inference time. However, these methods only focus on traditional CNNs, and did not experimentally evaluate with portable CNNs due to the difficulty in improving their efficiency. This motivates us further to reduce FLOPs, parameters, and inference latency of all types of CNNs, without incurring accuracy loss.

Lightweight Network Topology Design. In recent years, a series of efficient convolutional architectures have been proposed for lightweight network topologies. Aside from the convolution components already introduced (e.g., DWSConv, group convolution and channel shuffle [13, 23]), GhostNet [5] aims to reduce input redundancy by using fewer convolutional filters to process input features, which is beneficial to reduce the FLOPs, parameters and memory movements in networks. Fewer memory movements can reduce inference latency in a network. Therefore, drawing on the concept of GhostNet, our fundamental idea in this paper is to identify cheap combinations of operations and efficient convolutional components architectures to build our new CNN network.

3 The GhostShiftAdd Module: Less is More

Assume input data $I \in \mathbb{R}^{c_i \times h_i \times w_i}$, where c_i is the number of input channels, h_i and w_i are the height and width of the input data respectively. The operation of a standard convolutional layer for producing features can be formulated as $O = I * f + b$, where $*$ is the convolution operation, b is the bias term. $O \in \mathbb{R}^{h_o \times w_o \times c_o}$ is the output feature map with c_o channels, and $f \in \mathbb{R}^{c_i \times k \times k \times c_o}$ is the convolution filter in this layer. Moreover, h_o and w_o are the height and width of the output data, and $k \times k$ is the kernel size. During this convolution procedure, a standard convolutional network requires $c_i \cdot h_o \cdot w_o \cdot c_o \cdot k \cdot k$ FLOPs, which is often massive since the number of output filters c_o and the input channel number c_i are generally very large (e.g. 256 or 512) [5]. In addition, the large number of c_i may increase memory movements resulting in high latency.

3.1 Bit-wise shifts for Generating Intrinsic features

Unlike GhostNet [5], which divides the output features into two equal parts $m1 = m2$ in all cases, we provide a more general form that uses the ratio γ to balance the two parts, such that $m_1 = c_o/\gamma$ and $m_2 = m_1(\gamma - 1)$ ($\gamma >= 2$). In Fig 1, the red part represents the number of intrinsic features m_1 in output layer. The output intrinsic feature $O_{Int} \in \mathbb{R}^{(m_1 \times h_o \times w_o)}$ can be formulated as

$$O_{Int} = f_s(I, w_s), \quad \text{where} \quad f_s(I, w_s) = \sum I^T * w_s \,, \tag{1}$$

where $f_s(\cdot,\cdot)$ is a bit-wise filter that performs an inner product, and $w_s = s \cdot 2^p$ are weights in the bit-wise shift, where p is an integer parameter, and $s \in \{-1,0,1\}$ is a sign flip operator. In particular, the bit-wise shift operation for multiplication (left shift) or division (right shift) depends on the parameter p. If p is positive, the input value $I << p$ means left-shift; if p is negative, then right-shift. Since bit-wise shifts are always positive and there is a lack of a negative search space, the sign flip operation can expand the bit shift search space to provide a negative value (see Elhoushi et al. [4] for full details). Compared to GhostNet, the computational cost of bit-wise shift operations is much cheaper than multiplication.

3.2 Depth-wise Separable Bit-wise shifts with Adders for Generating Ghost features

The blue part in Fig 1 represents the number of m_2 output features, namely ghost features. The ghost feature size is $O_{ghost} \in \mathbb{R}^{(m_2 \times h_o \times w_o)}$. These are generated by the intrinsic feature through the DWSConv structure, where the depth-wise layer uses bit-wise shift operations ($k = 3$), and the point-wise layer uses addition operations ($k = 1$). This idea was motivated by two reasons. Firstly, in the FP format, we find that the addition operation applied to the deep convolution filter is slower than using shift and multiplication, especially the large filter kernel size (for example, $k = 3$ or 5). However, when $k = 1$, addition is as fast as shifting or multiplication. Secondly, using shift operations throughout DWSConv results in a relatively high loss in accuracy. This is because, compared with the standard convolution spanning the entire continuous space of the multiplication map in DWSConv, the bit-wise shift can only represent a subset of the power of 2 multiplication [22]. However, addition can effectively expand the shift parameter mapping space. Therefore, after the depth-wise layer based on bit-wise operations, we use the addition operation in the 1×1 point-wise layer to avoid high latencies and improve the accuracy of the bit-wise shift operations. Combining with Eq. 1, the ghost features part can be mathematically described as

$$O_{ghost} = f_a(f_s(O_{Int}, s \cdot 2^p), w_a), \quad \text{where} \quad f_a(f_s, w_a) = -\sum \|f_s - w_a\|_1 \,, \tag{2}$$

where $f_a(\cdot,\cdot)$ is the adder filter, and w_a is the addition weights that is measured by ℓ_1 distance. Finally, we concatenate the intrinsic and ghost features as output layer $Y = [O_{Int}, O_{ghost}]$.

3.3 Optimization of GhostSA

In an addition operation based CNN, the relatively large variance of the gradient would increase the FLOPs during the training process. Therefore, in our case, choosing an effective optimization method becomes very important. Stochastic Gradient Descent (SGD) and Adam are widely used in many addition networks [2, 18, 22]. The Adam method uses an adaptive learning rate to converge faster than SGD in some cases. However, the adaptive learning rate computed using the exponential moving average of the squared gradient cannot guarantee convergence in certain conditions, for example, high-dimensional settings when the variance of the gradient to time is large [14]. Therefore, we applied the Variance Controlled Stochastic Method (VCSG) [1] to effectively reduce the variance. This offers two improvements: the update of the adaptive learning rate switches between fixed and decayed value depending on the current variance, and a hyper-parameter controls the variance of VCSG at different stages of the training process.

3.4 Complexity of GhostSA

This section analyzes the performance of the GhostSA module through acceleration, model compression and memory access rates. The output channels of the intrinsic part is m_1, and the kernel size is d (it is usually recommended that $d = 1$). The intrinsic part requires zero FLOPs to calculate the power-of-2 function in bit-wise shift operations. Since an identity mapping has $m_1(m_2 - 1)$ filters for ghost features, it requires $h_o \cdot w_o \cdot m_1 \cdot m_2$ FLOPs. Finally, the theoretical speedup ratio of upgrading spatial convolution with our GhostSA module is

$$r_s = \frac{c_i \cdot c_o \cdot h_o \cdot w_o \cdot k \cdot k}{h_o \cdot w_o \cdot m_1 m_2} = \frac{c_i \cdot c_o \cdot k \cdot k}{m_1 \cdot m_2} = \frac{k \cdot k \cdot c_i \cdot c_o}{\frac{c_o}{\gamma} \cdot \frac{c_o(\gamma-1)}{\gamma}} \approx \frac{\gamma k \cdot k \cdot c_i}{c_o} \approx k^2 \gamma \,, \tag{3}$$

where $d \times d = 1 \times 1$, the last approximation is obtained by assuming $\gamma << c_i$. Moreover, intrinsic and ghost part require $log(c_i \cdot d \cdot d \cdot m_1)$ and $log(m_1 \cdot k \cdot k) + m_1 m_2$ parameters, respectively. The compression ratio of GhostSA can be formulated as

$$r_c = \frac{c_i \cdot c_o \cdot k \cdot k}{log(c_i \cdot m_1 \cdot d \cdot d) + log(m_1 \cdot k \cdot k) + m_1 \cdot m_2} = \frac{c_i \cdot c_o \cdot k \cdot k}{2log(c_i \cdot \frac{c_o}{\gamma} \cdot k \cdot k) + m_1 \cdot m_2} \approx k^2 \gamma \,. \tag{4}$$

Finally, the memory access ratio from input to output feature maps is

$$r_m = \frac{c_i \cdot c_o \cdot k \cdot k \cdot h_o \cdot w_o + c_o \cdot h_o \cdot w_o}{h_o \cdot w_o (c_i \cdot m_1 + m_1 + m_1 \cdot k \cdot k + m_1 + m_1 \cdot m_2 + m_2)} = \frac{k \cdot k \cdot \gamma \cdot c_i + c_o}{c_i + c_o + k \cdot k + 1} \approx k^2 \gamma \,. \tag{5}$$

In the three ratios of Eq 3, 4 and 5, the kernel size k is a constant value. The hyper-parameter γ can be flexibly adjusted to control the number of shift and addition operations in each convolution layer: a key factor affecting acceleration, compression, and memory access ratios. When increasing γ, the division of ghost features becomes larger, resulting in more addition operations than shift operations in total. In this case, the same trend on both CPU and GPU is observed: that the number of parameters, FLOPs and accuracy are increased. However, the latency of the model on GPU and CPU show different trends. A larger number of shift operations on the GPU will increase the inference latency, but on the CPU will decrease latency.

3.5 GhostSANet: Efficient CNNs

GhostSA Bottlenecks. Based on the GhostSA module, we proposed the GhostSA bottlenecks that integrate GhostSA modules and shortcuts, as shown in Fig 2. There are two versions of the GhostSA bottleneck when stride=1 and stride=2. In the case of stride=1, the GhostSA bottleneck consists of two stacked GhostSA modules. The first GhostSA module acts as an expansion layer that uses *expansion ratio* to balance input and output channels. The second GhostSA module reduces the number of channels from the expansion layer to match the shortcut path. In the middle of two GhostSA modules, Batch normalization (BN) [11] is used after each layer, but only used with Relu non-linear activation layers after the first GhostSA module. For the case of stride=2, we implement the shortcut path by inserting a down-sampling layer (*max-pool*) between the two GhostSA modules, which is an alternative to the depth-wise convolution structure in the Ghost bottleneck [6]. The motivation is to simplify the network architecture and control flow, especially on edge hardware platforms.

GhostSANet We use the GhostSA bottleneck to build a new efficient and lightweight CNN architecture, called *GhostShiftAddNet* (GhostSANet). Following from architectures of GhostNet, MobileNet v3, GhostSANet uses bit-wise shift operations in all convolutional layers, and applies GhostSA bottlenecks in the block layer, and use the ReLU non-linear activation in the first convolution layer and the GhostSA bottleneck, and apply hard-swish [8] in the last convolutional layer before the fully connected layer. To adapt to the application of different scales, we use the width multiplier α in the model, expressed as GhostSANet-$\alpha\times$, which can scale the width of the entire network.

4 Experiments and Results Analysis

Datasets and Architecture Settings Our experiments are performed on image classification datasets, including the CIFAR10 [12] and ImageNet ILSVRC 2012 [15] datasets. The CIFAR10 dataset is used to analyze the attributes of the proposed method, which consists of 60,000 32×32 colour images in 10 classes, including 50,000 training images and 10,000 test images. ImageNet is a relatively large-scale image dataset containing 1.2M training images and 50K validation images in 1,000 classes. Our experiments were conducted on three hardware platforms: Intel Core i7-10700 CPU, NVIDIA GeForce RTX 2080 Ti GPU, and NVIDIA Jetson Nano (an embedded platform containing a quad-core Arm CPU, and a 128-core GPU based on the NVIDIA Maxwell architecture). For direct comparisons, we re-implement all benchmarks in our experiments following our architecture settings.

Analysis of Hyper-parameters of GhostSA Module In order to evaluate the behaviour of shift and addition operations on heterogeneous computing devices (such as CPU and GPU), we analyze the values of hyper-parameters γ in the GhostSA module on popular deep CNNs (VGG-16 [17] and ResNet-20 [6]) with the CIFAR-10 dataset. The GhostSA module replaces all convolutional layers of the reference network, and the new models are named GhostSA-VGG-16 and GhostSA-ResNet-20.

In Table 1, we test the performance of Ghost modules applied to two backbone models named Ghost-ResNet-20 and Ghost-VGG-16, and set appropriate kernel sizes $k = 3$ and $d = 1$. The experimental results verify our theoretical analysis. As γ is increased from 2 to 6, the partition of intrinsic features becomes smaller, resulting in the output layer containing fewer intrinsic features to represent the input features and more ghost features. More ghost features supplement information from the intrinsic part, which can increase the accuracy. In such a

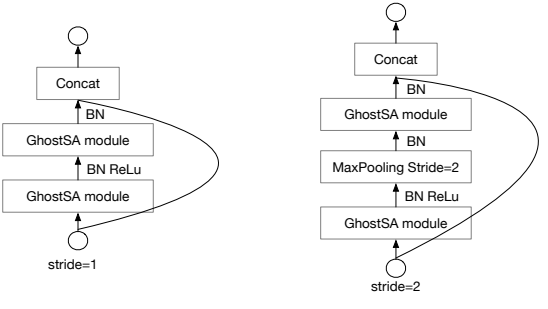

Figure 2: GhostSA bottleneck: stride=1 (left) and stride=2 (right).

Model	γ	Weights (M)	FLOPs (M)	Top1-Acc. (%)	GPU-Latency (ms)	CPU-Latency (ms)
ResNet-20 [8]	-	0.27	41	92.2	7.5	83
	$\gamma=2$	0.007	1.12	87.7	8.7	67
	$\gamma=3$	0.009	1.23	88.2	8.2	81
GhostSA-ResNet-20	$\gamma=4$	0.016	1.64	89.3	8.0	99
	$\gamma=5$	0.021	2.36	90.3	7.6	113
	$\gamma=6$	0.045	4.67	92.4	7.2	127
Ghost-ResNet-20 [6]	-	0.14	20	92.3	7.3	70
VGG-16 [6, 12]	-	15	313	93.6	86	2.1
	$\gamma=2$	0.35	7.12	87.3	99	1.8
	$\gamma=3$	0.46	8.72	90.4	91	2.0
GhostSA-VGG-16	$\gamma=4$	0.57	12.1	91.6	87	2.5
	$\gamma=5$	0.92	17.8	92.1	83	3.1
	$\gamma=6$	1.8	35.6	93.5	78	3.6
Ghost-VGG-16 [6]	-	7.7	158	93.7	80	1.9

Table 1: Comparison the performance of Ghost modules and GhostSA modules with different γ for compressing ResNet20 and VGG-16 on CIFAR10.

case, the greater number of addition operations will increase parameters and FLOPs, which places a burden on compute-bound platforms (e.g. CPU). Accordingly, the total number of bit-shift operations is relatively small, reducing memory movement on GPU. In addition, compared to the Ghost module, our module has significantly fewer FLOPs and parameters, and the accuracy loss is kept within $< 1\%$, even slightly higher than the Ghost module. After many experiments and comparisons, we choose $\gamma = 4$ for GPU and $\gamma = 2$ for CPU in the following experiments.

GhostSA Bottleneck on State-of-the-arts We compare the performance of three bottlenecks including the GhostSA bottleneck, GhostNet and ShiftAddNet, by applying them in SOTA architectures, e.g., MobileNet 2 [16], 3 [8], ShuffleNet 2 [13] and GhostNet [6] on the CIFAR-10 dataset, as shown in Fig 3. The GhostSA bottleneck can reduce FLOPS by up to $50\times$, and GPU latency by up to $1.5\times$ while achieving similar or even higher accuracy. Among these backbones, GhostNet achieves the highest accuracy of 94.1% within 18.1 milliseconds on the GPU, requiring more than 6.9 million FLOPs and 5.2 million parameters. After applying GhostSA in GhostNet, it only takes 13.5 ms to obtain an accuracy of 93.2% on the GPU with 0.18M FLOPs and 1.6M parameters. We also compared the GhostSA module with ShiftAddNet [22], and the results show that the GhostSA module is superior to the ShiftAddNet method.

GhostSANet for ImageNet Classification To evaluate the performance of the proposed GhostSANet model, we compared it to the SOTA models introduced in Fig 3, and add two other models into the comparison including FBNet V2 [20] and MnasNet [19] on the ImageNet classification task. Following common practice, all networks have three levels of computational complexity, i.e., 40, 140, and 200-300 MFLOPs. For the sake of fairness, we test these models and GhostSANet under the same initial training settings. We set the DWS kernel size $k = 3$ for the ghost part. The comparison results are summarised in Table 2, showing that greater FLOPs can lead to higher accuracy with a larger number of parameters. Our model provides improved performance compared to the SOTA, as shown in blue.

Evaluation on Embedded Platform. In the final experiment, we test the GhostSA bottleneck and GhostSANet performance on the Jetson Nano. With limited computational re-

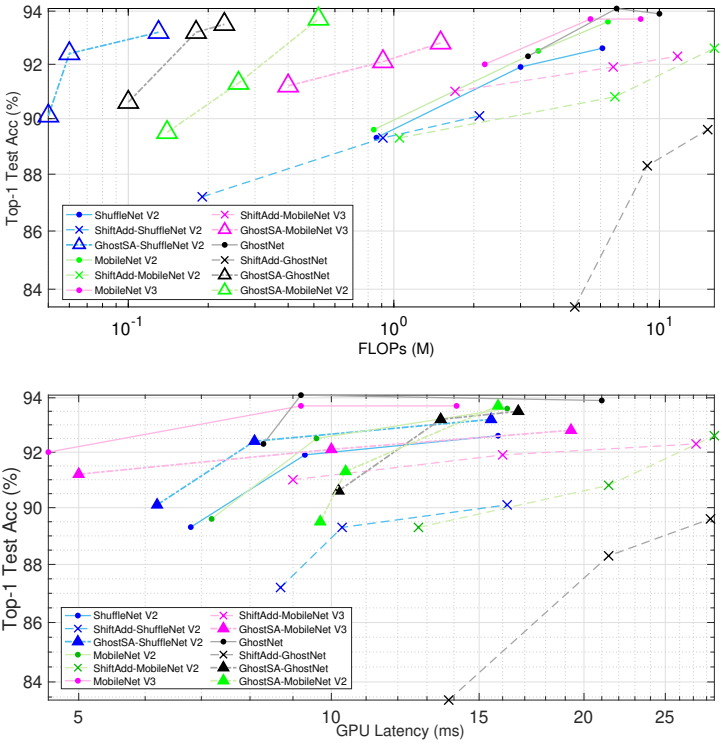

Figure 3: Comparison of FLOPs (top) and latency (bottom) of state-of-the-art lightweight networks against accuracy on the CIFAR10 dataset.

Model	Weights(M)	FLOPs(M)	Top-1 Acc. (%)
ShuffleNetV2 0.5x [23]	1.4	42	61.1
MobileNetV2 0.35x [16]	1.7	59	60.3
MobileNetV3 Small 0.75x [8]	2.4	44	64.5
GhostNet 0.5x [6]	2.6	42	66.2
GhostSANet 0.5x	2.3	20	66.9
MobileNetV2 0.65X [16]	2.3	172	67.2
ShuffleNetV2 1.0X [23]	2.3	146	67.8
MobileNetV3 Large 0.75X [8]	4.0	155	73.3
GhostNet 1.0X [6]	5.2	141	73.9
GhostSANet 1.0X	3.0	40	73.4
MobileNetV2 1.0x [16]	3.4	300	71.8
ShuffleNetV2 1.5x [23]	3.5	299	72.6
FBNet-B [20]	4.5	295	74.1
MnasNet-A1 [19]	3.9	312	75.2
MobileNetV3 Large 1X [8]	5.4	219	75.2
GhostNet 1.3x [6]	7.3	226	75.7
GhostSANet 1.3x	3.7	69	76.4

Table 2: Comparison of state-of-the-art networks over classification accuracy, the number of weights and FLOPs on the ImageNet dataset.

Model name	Backbone **G-Backbones**	Top1-Acc. (%)	GPU Speed (Batches/ms)	CPU Speed (Images/sec)
MobileNet-V3 [9]	0.25x(S)	87.6	1207	114
	G-0.25x(S)	**88.3**	**1454**	**160**
	0.75x(S)	91.3	1000	36
	G-0.75x(S)	**91.5**	**1084**	**74**
	0.75x(Large)	91.9	888	24
	G-0.75x(L)	**91.9**	**1000**	**58**
MobileNet-V2 [14]	0.25x	87.3	1049	81
	G-0.25x	**87.8**	**1230**	**112**
	0.65x	89.5	429	36
	G-0.65x	**90.3**	**500**	**38**
	1x	92.4	365	27
	G-1x	**91.9**	**441**	**33**
ShuffleNet-V2 [23]	0.5x	87.5	1032	96
	G-0.5x	**88.6**	**1306**	**152**
	1x	91.3	780	62
	G-1x	**91.5**	**901**	**78**
GhostNet [8]	0.5x	91.6	727	28
	1x	94.8	627	19
GhostSANet	0.5x	92.1	1027	69
	1x	95.1	819	56

Table 3: GhostSA applied backbone models on Jetson Nano for CIFAR10 that is shown as G-backbones. All benchmarks are implemented by us.

source, low latency requirements become essential for real-time data analysis. Therefore, our experiment focuses on evaluating the GPU latency (ms) and CPU latency (s) of the network. As summarized in Table 3, we test the GhostSA bottleneck applied to each SOTA network, highlighted in bold. After applying the GhostSA bottleneck, the GPU/CPU speed of the backbone network can be increased by up to 1.3× and 1.6×, respectively. Moreover, we test GhostSANet (shown in blue) showing that it can obtain a 0.3% higher Top-1 test accuracy compared to GhostNet, with 1.3× and 2× lower GPU and CPU latency. In addition, GhostSANet requires less time to achieve similar performance, compared to other SOTA methods. For example, GhostSA-MobileNet V3 0.25x has lower GPU and CPU latency (44ms and 0.2s respectively), but the accuracy is only 88.3%, while GhostSANet with the same accuracy only requires 26ms and 0.08s. Overall, our network is more effective and efficient on embedded devices than other current SOTA networks.

5 Conclusions

This paper proposes a new GhostSA module for building an efficient neural architecture, and shows that it can significantly concurrently reduce the computational cost, number of parameters, and inference latency. Moreover, the GhostSA module can flexibly adjust the number of shift and addition operations in a convolution layer through the hyper-parameter γ, which allows it to adapt to different hardware platforms. Experiments show that the proposed GhostSANet has excellent efficiency and accuracy on both desktop GPU/CPU and embedded platforms (Nvidia Jetson Nano). We anticipate that this work can inspire future work to design network architectures for embedded hardware systems that are both energy-saving and platform-aware.

6 Acknowledgements

This work was supported in part by the Engineering and Physical Sciences Research Council (EPSRC) under Grant EP/S030069/1.

References

[1] Jia Bi and Steve R. Gunn. A variance controlled stochastic method with biased estimation for faster non-convex optimization. In *European Conference on Machine Learning and Principles and Practice of Knowledge Discovery in Databases (ECML PKDD 2021)*, Sep 2021.

[2] Hanting Chen, Yunhe Wang, Chunjing Xu, Boxin Shi, Chao Xu, Qi Tian, and Chang Xu. Addernet: Do we really need multiplications in deep learning? In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2020.

[3] Weijie Chen, Di Xie, Y. Zhang, and Shiliang Pu. All you need is a few shifts: Designing efficient convolutional neural networks for image classification. *In IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 7234–7243, 2019.

[4] Mostafa Elhoushi, Zihao Chen, Farhan Shafiq, Ye Henry Tian, and Joey Yiwei Li. Deepshift: Towards multiplication-less neural networks. In *2021 IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops (CVPRW)*, pages 2359–2368, 2021. doi: 10.1109/CVPRW53098.2021.00268.

[5] Kai Han, Yunhe Wang, Qi Tian, Jianyuan Guo, Chunjing Xu, and Chang Xu. Ghostnet: More features from cheap operations. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2020.

[6] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 770–778, 2016. doi: 10.1109/CVPR.2016.90.

[7] Yihui He, Xianggen Liu, Huasong Zhong, and Yuchun Ma. Addressnet: Shift-based primitives for efficient convolutional neural networks. In *IEEE Winter Conference on Applications of Computer Vision, WACV 2019, Waikoloa Village, HI, USA, January 7-11, 2019*, pages 1213–1222. IEEE, 2019. doi: 10.1109/WACV.2019.00134.

[8] Andrew Howard, Mark Sandler, Grace Chu, Liang-Chieh Chen, Bo Chen, Mingxing Tan, Weijun Wang, Yukun Zhu, Ruoming Pang, Vijay Vasudevan, Quoc V. Le, and Hartwig Adam. Searching for mobilenetv3. In *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)*, October 2019.

[9] Andrew G. Howard, Menglong Zhu, Bo Chen, Dmitry Kalenichenko, Weijun Wang, Tobias Weyand, Marco Andreetto, and Hartwig Adam. Mobilenets: Efficient convolutional neural networks for mobile vision applications. *CoRR*, abs/1704.04861, 2017.

[10] IEEE. Ieee standard for floating-point arithmetic. *IEEE Std 754-2019 (Revision of IEEE 754-2008)*, pages 1–84, 2019. doi: 10.1109/IEEESTD.2019.8766229.

[11] Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In Francis Bach and David Blei, editors, *Proceedings of the 32nd International Conference on Machine Learning*, volume 37 of *Proceedings of Machine Learning Research*, pages 448–456, Lille, France, 07–09 Jul 2015. PMLR.

[12] A. Krizhevsky and G. Hinton. Learning multiple layers of features from tiny images. *Master's thesis, Department of Computer Science, University of Toronto*, 2009.

[13] Ningning Ma, Xiangyu Zhang, Hai-Tao Zheng, and Jian Sun. Shufflenet v2: Practical guidelines for efficient cnn architecture design. In *Proceedings of the European Conference on Computer Vision (ECCV)*, September 2018.

[14] Sashank J. Reddi, Satyen Kale, and Sanjiv Kumar. On the convergence of adam and beyond. In *6th International Conference on Learning Representations, ICLR 2018, Vancouver, BC, Canada, April 30 - May 3, 2018, Conference Track Proceedings*, 2018.

[15] Olga Russakovsky, Jia Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, Michael Bernstein, Alexander C. Berg, and Li Fei-Fei. ImageNet Large Scale Visual Recognition Challenge. *International Journal of Computer Vision (IJCV)*, 115(3):211–252, 2015. doi: 10.1007/s11263-015-0816-y.

[16] Mark Sandler, Andrew Howard, Menglong Zhu, Andrey Zhmoginov, and Liang-Chieh Chen. Mobilenetv2: Inverted residuals and linear bottlenecks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2018.

[17] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. In *International Conference on Learning Representations(ICLR)*, 2015.

[18] Dehua Song, Yunhe Wang, Hanting Chen, Chang Xu, Chunjing Xu, and Dacheng Tao. Addersr: Towards energy efficient image super-resolution. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 15648–15657, June 2021.

[19] Mingxing Tan, Bo Chen, Ruoming Pang, Vijay Vasudevan, Mark Sandler, Andrew Howard, and Quoc V. Le. Mnasnet: Platform-aware neural architecture search for mobile. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2019.

[20] Alvin Wan, Xiaoliang Dai, Peizhao Zhang, Zijian He, Yuandong Tian, Saining Xie, Bichen Wu, Matthew Yu, Tao Xu, Kan Chen, Peter Vajda, and Joseph E. Gonzalez. Fbnetv2: Differentiable neural architecture search for spatial and channel dimensions. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2020.

[21] B. Wu, Alvin Wan, Xiangyu Yue, Peter H. Jin, S. Zhao, Noah Golmant, A. Gholaminejad, Joseph E. Gonzalez, and K. Keutzer. Shift: A zero flop, zero parameter alternative to spatial convolutions. *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 9127–9135, 2018.

[22] Haoran You, Xiaohan Chen, Yongan Zhang, Chaojian Li, Sicheng Li, Zihao Liu, Zhangyang Wang, and Yingyan Lin. Shiftaddnet: A hardware-inspired deep network. In H. Larochelle, M. Ranzato, R. Hadsell, M. F. Balcan, and H. Lin, editors, *Advances in Neural Information Processing Systems (NeurIPS)*, volume 33, pages 2771–2783, 2020.

[23] Xiangyu Zhang, Xinyu Zhou, Mengxiao Lin, and Jian Sun. Shufflenet: An extremely efficient convolutional neural network for mobile devices. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2018.