Compiler-Based Efficient CNN Model Construction for 5G Edge Devices

Kun Wan, Xiaolei Liu, Jianyu Yu, Xiaosong Zhang, Xiaojiang Du, *Fellow, IEEE*, and Nadra Guizani

***Abstract*— With the increasing demand to deploy convolutional neural networks (CNNs) on 5G mobile platforms, architecture designs with efficient sparse kernels (SKs) were proposed, which can save more parameters than the standard convolution while maintaining the high accuracy. Despite the great potential, neural network designs with SKs still require a lot of expert knowledge and take ample time. In this paper, we first propose a *search scheme* that effectively reduces the SK design space based on three aspects: *composition*, *performance*, and *efficiency*. Meanwhile, we completely eliminate the model training from our search scheme. Instead, an easily measurable quantity, the *information field*, is identified and used to predict the model accuracy in the searching process. Additionally, we provide a detailed *efficiency analysis* on the final designs found by our scheme. Second, based on the analysis we propose a *model transformation scheme* to better utilize the SK designs on existing models to either reduce the number of parameters or increase the accuracy. Last, considering the extra programming overhead and the expert knowledge required by the model transformation scheme, we develop a *compiler prototype* to automate the entire process, given the source code of an existing model. Experimental results show that models composed of the sparse kernel designs searched by our search scheme can beat state-of-the-art networks such as ResNets in terms of the accuracy and the efficiency. Also by using our model transformation scheme we can easily improve the accuracy (the same number of parameters) or the efficiency (the same accuracy) upon existing state-of-the-art models.**

***Index Terms*— 5G edge devices, efficient CNNs.**

I. Introduction

CNNs have achieved unprecedented success in visual recognition tasks. The development of 5G mobile devices drives the increasing demand to deploy these deep networks on mobile platforms such as cell phones and self-driving cars. However, CNNs are usually resource-intensive, making them difficult to deploy on these memory-constrained and energy-limited platforms. For example, the size of the AlexNet model [2] is over 200MB and the VGG-16 [3] is over 500MB [4].

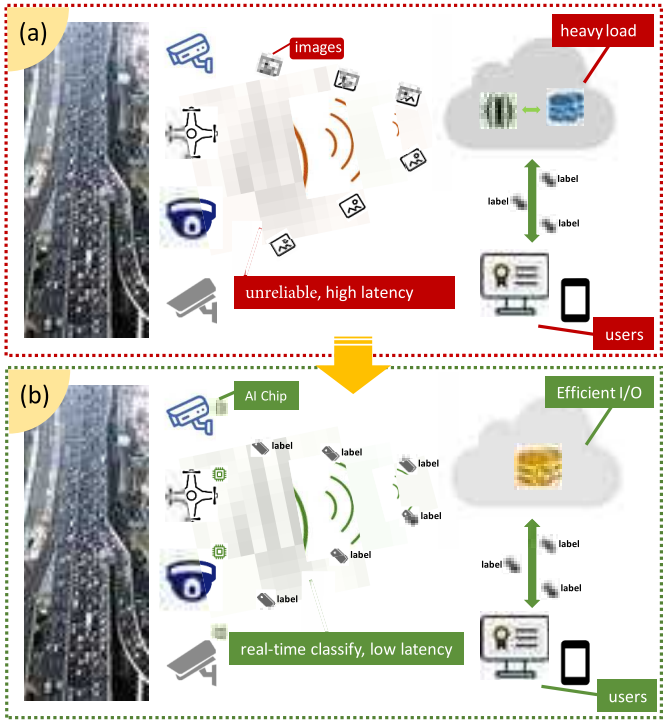


Fig. 1. Solutions for applying deep learning on the 5G edge devices. a) Traditional public cloud solution, b) Edge-learning-based public cloud solution.

To take advantage of the power of deep learning, a traditional solution is to rely on the public cloud. As shown in Figure 1a, 5G edge devices like cell phones send images/data to the public cloud which contains lots of computing power, then the public cloud will send the desired labels/results back to the devices. Nonetheless, such a solution is extremely time-consuming, tedious, and error prone due to the unreliable network between 5G edge devices and the public cloud [5].

To solve the problem, we decide to directly deploy pre-trained models on 5G edge devices like Figure 1b. To realize it, we need to compress the model while keeping its accuracy/performance as much as possible. Previously, several techniques have been proposed, including pruning [4],

Manuscript received March 28, 2020; revised December 8, 2020; accepted January 28, 2021. This work was supported in part by the Director of Computer Application Research Institute Foundation under Grant SJ2020A08 and in part by the China Academy of Engineering Physics Innovation and Development Fund Cultivation Project under Grant PY20210160. The Associate Editor for this article was S. Garg. *(Corresponding author: Xiaolei Liu.)*

Kun Wan and Jianyu Yu are with the Department of Computer Science, University of California at Santa Barbara, Santa Barbara, CA 93106 USA (e-mail: kun@cs.ucsb.edu; jianyuyu@ucsb.edu).

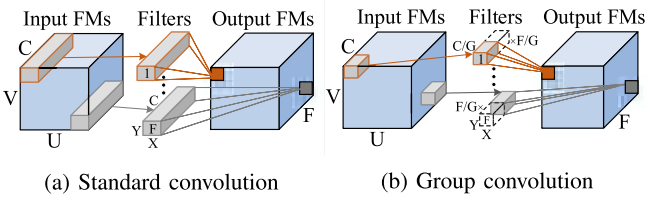
Xiaolei Liu is with the Institute of Computer Application, China Academy of Engineering Physics, Mianyang 621900, China (e-mail: liuxiaolei@caep.cn).

Xiaosong Zhang is with the Cyberspace Security Research Center, University of Electronic Science and Technology of China, Chengdu 611731, China (e-mail: johnsonzxs@uestc.edu.cn).

Xiaojiang Du is with the Department of Computer and Information Sciences, Temple University, Philadelphia, PA 19122 USA (e-mail: dxj@ieee.org).

Nadra Guizani is with the School of Electrical Engineering & Computer Science, Washington State University, Pullman, WA 99163 USA (e-mail: nadra.guizani@wsu.edu).

Digital Object Identifier 10.1109/TITS.2021.3056426



(a) Standard convolution (b) Group convolution

Fig. 2. Comparison between the standard convolution and the group convolution.

[6]–[19], quantization [20]–[32], and low rank approximation [33]–[43]. Though these approaches can offer a reasonable parameter reduction with minor accuracy degradation, they suffer from the following three drawbacks: 1) irregular network structure after compression, which limits performance and throughput on GPU; 2) increased training complexity due to the additional compression or re-training process; and 3) heuristic compression ratios depending on networks, which cannot be precisely controlled.

Recently, the SK approach was proposed to mitigate these problems by directly training networks using structural (large granularity) sparse convolutional kernels with fixed compression ratios. As an example, group convolution [44] achieves sparsity over the channel dimension by limiting the number of input channels each output channel connects to, which is illustrated clearly in Figure 2b. The idea of SK was originally proposed as different types of convolutional approaches. Later researchers explored their usages in the context of CNNs by combining some of these SKs to save parameters/computation against the standard convolution. For example, MobileNets [45] realized 7× parameter savings with only 1% accuracy loss by adopting the combination of two SKs, depthwise convolution [46], and pointwise convolution [47], to replace the standard convolution in their networks.

However, despite the great potential with the SK approach to save parameters/computation while maintaining accuracy, it is still unknown how to craft an SK design with such potential (i.e., effective SK design). Prior works like MobileNet [45] and Xception [48] adopted simple combinations of existing SKs, and did not state the reasons why they choose such a design. Meanwhile, it has been a long-existing question in the field whether there is any other SK design that is more efficient than all the state-of-the-art ones while still maintaining a similar accuracy with the standard convolution.

To answer this question, a native idea is to try all possible combinations and get the final accuracy for each of them. Unfortunately, the number of combinations will grow exponentially with the number of kernels in a design, and thus it is infeasible to train each of them. Specifically, even if we limit the design space to four common types of SKs – group convolution [44], depthwise convolution [46], pointwise convolution [47], and pointwise group convolution [49] – the total number of possible combinations would be 4^k, given that k is the number of SKs we use in a design (note that each SK can appear more than once in a design).

In this paper, we first craft an effective SK design by efficiently eliminating poor candidates from the large design space. Specifically, we reduce the design space from three aspects: composition, performance, and efficiency. First, observing that in normal CNNs, it is quite common to have multiple blocks which contain repeated patterns such as layers or structures, we eliminate the design space by ignoring the combinations including repeated patterns. Second, realizing that removing designs with large accuracy degradation would significantly reduce the design space, we identify an easily measurable quantity named *information field* behind various SK designs, which is closely related to the model accuracy. We get rid of designs that lead to a smaller *information field* compared to the standard convolution model. Last, in order to achieve a better parameter efficiency, we remove designs containing redundant SKs which do not contribute to the size of the *information field*. With all aforementioned knowledge, we present an SK scheme that incorporates the final four different designs automatically reduced from a large original design space.

Additionally, in practice, researchers would also like to select the most parameter/computation efficient SK designs based on their needs, which drives the demand to study the efficiency for different SK designs. Previously, no research has investigated the efficiency for any SK design. In this paper, three aspects of efficiency are addressed for each of the designs in our SK scheme: 1) what are the factors that could affect the efficiency in different scenarios? 2) what is the best efficiency that could be achieved combining all these factors? 3) what is the condition for the best efficiency to be achieved?

Besides, by considering a case in real life instead of building a model completely from scratch, researchers are more likely to make modifications on existing models to realize a specific resource constraint. In this paper we propose a model transformation scheme to better utilize the SK designs to achieve this goal. Specifically, given an existing model, the transformation scheme will iterate through each layer and find the most efficient design in our SK scheme, then replace it based on the layer information and the efficiency analysis of different designs.

However, this process would require a lot of expert knowledge, e.g., selecting the most efficient SK design. In addition, the extra programming overhead would also be a problem. Unlike the transition within different variants of one type of network (e.g. from VGG-16 to VGG-19), simply modifying the loop count will not work for the model transformation since different layers of a network could be replaced with totally different SK designs, not to mention different types of networks.

To mitigate this problem, inspired by the concept of intent-based networking [50], [51], in this paper we integrate the model transformation scheme with a compiler prototype to automate the entire process. Specifically, given the code of an existing model, the compiler will first convert the code into an abstract syntax tree (AST), and for each layer the layer information can be extracted from the tree. Then as per the information, the best SK design is chosen and the replacement can be conducted by directly modifying the AST. Last, after replacing all convolutional layers, the AST will be converted back to the new source code.

The contributions of our paper can be summarized as follows:

- We are the first in the field to point out that the *information field* is the key for the SK designs. Meanwhile we observe that the model accuracy is positively correlated to the size of the *information field*.
- We present a SK scheme to illustrate how to eliminate the original design space from three aspects and incorporate the final 4 types of designs along with rigorous mathematical foundation on the efficiency.
- We provide some network designs which are in the scope of our SK scheme and have not been explored yet and show that they could have superior performance compared to state-of-the-art network units.
- We are the first to propose a model transformation scheme to provide guidance on how to apply SK designs on existing models to either reduce the number of parameters or improve the accuracy.
- We develop a compiler prototype to automate the entire process of model transformation and experiments show that transformation on some models could even result in smaller networks with better accuracy than other existing efficient models.

II. Preliminaries

We first give a brief introduction to the standard convolution and the four common types of SKs.

A. Standard Convolution

Standard convolution is the basic component in most CNN models, kernels of which can be described as a 4-dimensional tensor: $W \in \mathbb{R}^{C\times X\times Y\times F}$, where C and F are the numbers of the input and the output channels, respectively, and X and Y are the spatial dimensions of the kernels. Let $I \in \mathbb{R}^{C\times U\times V}$ be the input tensor, where U and V denote the spatial dimensions of the feature maps. Therefore, the output activation at the output feature map f and the spatial location (x, y) can be expressed as,

$$T(f,x,y)=\sum_{c=1}^{C}\sum_{x'=1}^{X}\sum_{y'=1}^{Y} I(c,x-x',y-y')\cdot W(c,x',y',f)$$

B. Group Convolution

Group convolution was first used in AlexNet [44] for distributing the model over two GPUs. The idea of group convolution is to split both the input and the output channels into disjoint groups and each output group is connected to a single input group and vice versa. By doing so, each output channel will only depend on a fraction of input channels instead of the entire ones, thus a large amount of parameters and computation could be saved. Considering the number of groups as M, the output activation (f, x, y) can be calculated as,

$$T(f,x,y)=\sum_{c'=1}^{C/M}\sum_{x'=1}^{X}\sum_{y'=1}^{Y} I(\frac{C}{M}\lfloor\frac{f-1}{\frac{F}{M}}\rfloor+c',x-x',y-y') \cdot W(c',x',y',f)$$

C. Depthwise Convolution

The idea of depthwise convolution is similar to the group convolution, both of which sparsify kernels in the channel extent. In fact, depthwise convolution can be regarded as an extreme case of group convolution in which the number of groups is exactly the same as the number of input channels. Also notice that in practice, usually the number of channels does not change after the depthwise convolution is applied. Thus, the equation above can be further rewritten as,

$$T(f,x,y)=\sum_{x'=1}^{X}\sum_{y'=1}^{Y} I(f,x-x',y-y')\cdot W(x',y',f)$$

D. Pointwise Convolution

Pointwise convolution is actually a 1×1 standard convolution. Different from group convolution, pointwise convolution achieves the sparsity over the spatial extent by using kernels with 1×1 spatial size. Similarly, the equation below shows how to calculate one output activation from the pointwise convolution in detail,

$$T(f,x,y)=\sum_{c=1}^{C} I(c,x,y)\cdot W(c,f)$$

E. Pointwise Group Convolution

To sparsify kernels in both the channel and the spatial extents, the group convolution can be combined together with the pointwise convolution, i.e., pointwise group convolution. Besides the use of 1×1 spatial kernel size, in pointwise group convolution, each output channel will also depend on a portion of input channels. The specific calculations for one output activation can be found from the equation below.

$$T(f,x,y)=\sum_{c'=1}^{C/M} I(\frac{C}{M}\lfloor\frac{f-1}{\frac{F}{M}}\rfloor+c',x,y)\cdot W(c',f)$$

III. Sparse Kernel Scheme

Recall that the total number of combinations will grow exponentially with the number of kernels in a design, which could result in a large design space. In this paper, we craft the effective SK design (i.e., design that consumes less parameters but maintains accuracy with the standard convolution) by efficiently eliminating the design space.

Specifically, we first determine the initial design space by setting the maximum number of SKs (length). To decide this number, two aspects are considered: 1) in order to give the potential to find more efficient designs which have not been explored yet, the maximum length of the SK design should be greater than the number of all the state-of-the-art ones; 2) it is also obvious that the greater length is more likely to consume more parameters, which contradicts our goal to find more efficient designs. Therefore, combining the two aspects together, we set the maximum length to 6, which is not only greater than the largest number (i.e., 3) in all current

designs, but also ensures that designs with the maximum length will still be more efficient than the standard convolution. As a result, the original design space can be expressed as $(4^1 + 4^2 + \cdots + 4^6)$.

Algorithm 1 Design Space Reduction

```
S: the original design space
CalculateInfo(): calculate the new information field
STANDARDINFO: information field of standard conv
import re                          ▷ standard regular expression module
for each design d in S do
    if re.match("(.+?)\1+", d) then
        S.remove(d)
    end if
end for
currInfo ← (1, 1, 1)               ▷ initialize the information field
for each design d in S do
    flag ← 0                       ▷ flag to indicate the early-stop
    for each kernel k in d do
        preInfo = currInfo
        currInfo = CalculateInfo(preInfo, k)
        if currInfo == preInfo then
            flag ← 1
            break
        end if
        if currInfo <= STANDARDINFO then
            continue
        else
            flag ← 1
            break
        end if
    end for
    if flag == 1 then
        S.remove(d)
        continue
    end if
    if currInfo < STANDARDINFO then
        S.remove(d)
        continue
    end if
end for
return S
```

A. Reduce the Design Space

We then start to reduce the design space from three aspects: composition, performance, and efficiency. In the following paragraphs, we will introduce the three aspects in detail. The overall flow to reduce the design space from the three aspects is shown in Algorithm 1.

1) Composition: The overall layout of CNNs provides a good insight for us to quickly reduce the design space. Specifically, in normal CNNs, it is quite common to have multiple stages/blocks which contain repeated patterns such as layers or structures. For example, in both VGG [3] and ResNet [1] there are 4 stages with several repeated layers inside each stage. Inspired by this fact, when we replace the standard convolution using various SK designs intuitively there is no need to incorporate these repeated patterns into the original place of each standard convolutional layer. For example, if there are three types of SKs, A, B, and C, then the following combinations should be removed as containing repeated patterns: AAAAAA, ABABAB, and ABCABC. AAAAAA contains the repeated pattern of A, while ABABAB and ABCABC have the patterns of AB and ABC, respectively.

Repeated patterns are also easy to detect, which makes the entire process extremely fast. To find such patterns, we can use the regular expression matching. The corresponding expression for the matched combinations should be $(.+?)\backslash 1+$, where $(.+?)$ denotes the first capturing group which contains at least one character, but as few as possible, and $\backslash 1+$ tries to match the same character(s) as most recently matched by the first group as many times as possible. As a result, we can efficiently eliminate the design space with the help of the regular expression.

2) Performance: There are many SK designs that could result in large accuracy degradation, which gives us another opportunity to greatly reduce the design space. To get rid of them, we need an easily measurable (i.e., no training) property behind various designs that could directly indicate the final accuracy. Fortunately, after analyzing many prior works and conducting many experimental studies, we do find such a property. We name it the *information field*.

Definition 1 (Information Field): Information field is the area in the input tensor where one or more convolutional layers are used to generate one output activation. For one output tensor, sizes of information fields for all activations are usually the same.

Figure 2a shows the data dependency for the standard convolution, from which we can also find out the size of the *information field*. Assuming the spatial kernel size is 3×3, starting from any output node in the figure, we can see that in terms of the channel dimension, each output channel will connect to all input channels. For the spatial dimensions, one output activation will depend on activations inside a 3×3 spatial area. Therefore the *information field* for the standard convolution will be (3, 3, C) where C is the number of input channels.

We find that *information field* is the key behind all SK designs, and also observe that the model accuracy is positively correlated to the size of the *information field*, the idea of which is also validated by later experiments in Section VII-A.

With the help of *information field*, SK designs that would result in large accuracy degradation can be easily removed from the original design space without actually training the models. Specifically, for each design, we first calculate the size of the *information field* by adding it up sequentially from the leftmost kernel to the rightmost one. For example, we use a three-dimensional vector, (1, 1, 1), to represent the initial values of *information field* on three different dimensions (i.e., two spatial dimensions and one channel dimension), then corresponding values of the vector will be updated based

on the known properties of the SK encountered.[1] After the rightmost kernel, the final vector we get will be the size of the *information field* for the design. Finally, we compare it with that of the standard convolution. If the two sizes are the same, we will keep the design, otherwise we will simply discard it. For instance, the design composed of one depthwise convolution will be removed since its *information field* only contains one channel area instead of the full channel space from the standard convolution.

3) Efficiency: In order to achieve better parameters and computation efficiency, we remove designs that include SKs that do not contribute to the *information field*. Specifically, there are two cases that could worsen the efficiency and should be regarded as inferior designs: 1) it can be easily verified that the size of the *information field* will never decrease when passing through SKs in a design, thus in a situation where the *information field* remains constant across one kernel, the kernel is not helpful, even if the final size of the *information field* is the same as the standard convolution; 2) it is also possible that an *information field* that is the same size as the standard convolution is retained by a fraction of SKs in a design; in this case, other kernels do not contribute to the *information field*. Although other SKs can still enlarge the *information field* to improve accuracy by increasing the spatial dimension, in Section V-B, we argue that such a method is not parameter-efficient. As a result, in terms of efficiency, the designs in both cases contain non-contributing kernels, so we can remove them from the original design space.

To effectively detect and delete such inferior designs within the two cases, we introduce an *early-stop mechanism* during the process to check the size of the *information field* above. Specifically, as per the two cases, we check two things when adding up the *information field* from the leftmost kernel in a design: 1) we record the size of the *information field* before entering the next kernel and compare it with the new size calculated after that kernel. If the two sizes are the same, we immediately mark the current design as inferior; 2) we compare the new size of the *information field* with that of the standard convolution. If the size is smaller, we will continue to add up the *information field* from the next kernel; otherwise we will skip to the next design.

With all aforementioned knowledge, we write a program to automatically reduce the original design space. Also note that other techniques to save parameters such as bottleneck structure [1] appear to be complimentary to the SK approach, which can be combined together to further improve parameter efficiency while maintaining accuracy. To validate this idea and also increase the chance to explore more efficient SK designs, we take the bottleneck structure into consideration when reducing the design space.

Finally, the original design space ($4^1 + 4^2 + \cdots + 4^6$) can be reduced to 4 different types of SK designs.[2] In the next section, we will present the 4 designs.

B. Final Sparse Kernel Designs

1) Depthwise Convolution + Pointwise Convolution: Unlike the standard convolution, which combines the spatial and the channel information together to calculate the output, the combination of depthwise convolution (DW) and pointwise convolution (PW) splits the two kinds of information and deals with them separately. The dependency of such a design is depicted in Figure 3a, from which we can easily verify that the size of the *information field* is the same as the standard convolution.

2) Group Convolution + Pointwise Group Convolution: The combination of group convolution (GC) and pointwise group convolution (PWG) can be regarded as an extension from the design above, where group convolution is applied on the pointwise convolution. However, simply using the pointwise group convolution would reduce the size of the *information field* on the channel dimension since the depthwise convolution cannot fuse the channel information. To recover the *information field*, depthwise convolution is replaced with a standard group convolution, and channel permutation should be added between the two layers. When using such an SK design, we always assume the number of channels does not change after the first group convolution. Figure 3b clearly shows the *information field* of this design.

3) Pointwise Convolution + Depthwise Convolution + Pointwise Convolution: Although two pointwise convolutions do not ensure better efficiency in our SK scheme, the combination with the bottleneck structure can help ease the problem, letting it survive as one of the last designs. Following the common practice, we set the bottleneck ratio as $1:4$, which implies the ratio of the bottleneck channels to the output channels. Figure 3c shows that the *information field* of such a design is the same as the standard convolution.

4) Pointwise Group Convolution + Depthwise Convolution + Pointwise Group Convolution: The combination of two pointwise group convolutions and one depthwise convolution can also ensure that the *information field* is the same size as the standard convolution. Similarly, channel permutation is needed. The bottleneck structure is also adopted here to achieve better efficiency. Figure 3d can verify the size of the *information field* compared with the standard convolution.

IV. Efficiency Analysis

We notice that the conditions for the best efficiency of different designs in our SK scheme do not always overlap. Thus to ease the pain for researchers to find the most parameter/computation efficient designs based on their needs,

[1]The spatial size of the new *information field* can be calculated simply by adding the corresponding kernel size minus 1. However, the calculation of the channel size depends on whether the group convolution is used. If one design does not contain any group convolution, the channel size will always be equal to the number of input channels, otherwise the channel dependency with regards to the input channels must be recorded for all output channels every time a new SK encountered.

[2]During the process to eliminate the design space, we allow channel permutation (including reshaping, matrix transposing and flattening back) within the designs, and when a group convolution is encountered, we will try all possible numbers of groups to calculate the size of *information field*. As long as there is one group number that can pass the entire process, we will keep the design. In case there are multiple group numbers passing the process, we will consider them as the same design.

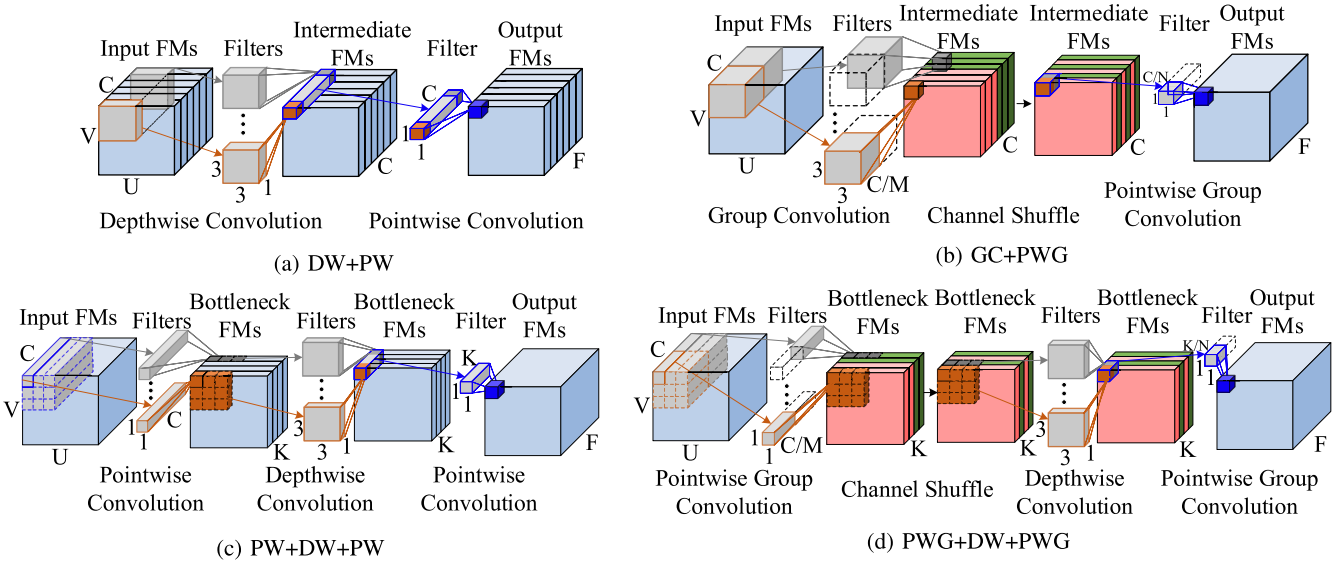


Fig. 3. Data dependency of different SK designs. Here the spatial kernel size is 3×3.

we analyze the efficiency for each of the designs in detail. Specifically, we consider two scenarios which are frequently encountered by researchers when applying SK designs: 1) given the input and the output for a layer; 2) given the total number of parameters for a layer. In the first scenario, we show the parameter efficiency for a SK design by comparing the total number of parameters before and after applying such a design. In the second one, since the total number of parameters is fixed, we compare the efficiency of different designs by observing the greatest widths they can achieve. In this section, the spatial kernel size 3×3 is used unless otherwise stated.

A. DW + PW

1) Efficiency Given the Input and the Output: Given the numbers of input and output channels C and F, the total number of parameters after applying this design is $9C + CF$, and the number of parameters for the standard convolution is $9CF$. Therefore the parameter efficiency of such a method is $1/F + 1/9$, represented by the ratio of parameters after and before applying such a design. Clearly, given C and F, the parameter efficiency is always the same.

2) Efficiency Given the Total Amount of Parameters: It can be easily verified that given the total number of parameters, the greatest width is reached when the best efficiency is achieved. Thus, the condition for the best efficiency given the total amount of parameters should be the same as the one in which the greatest width is reached.

The total number of parameters P for the design can be expressed as

$$P = 3 \cdot 3 \cdot C + 1 \cdot 1 \cdot C \cdot F.$$

When studying the greatest width, we need to assume the ratio between C and F does not change, thus the number of output channels F can be written as $F = \alpha \cdot C$ where usually $\alpha \in \mathbb{N}^+$. As a result, when P from the equation above is fixed, the greatest width G (i.e., $\frac{-9+\sqrt{81+4\alpha P}}{2\alpha}$) will also be fixed, which indicates that the parameter efficiency is always the same.

B. GC + PWG

1) Efficiency Given the Input and the Output: Similarly, we use the ratio of parameters to show the parameter efficiency of this design. Given C and F, the number of parameters after using such a design can be written as $3 \cdot 3 \cdot \frac{C}{M} \cdot C + 1 \cdot 1 \cdot \frac{C}{N} \cdot F = \frac{9C^2}{M} + \frac{CF}{N}$ where M and N represent the group numbers for GC and PWG, respectively. Since the number of parameters for standard convolution is $9CF$, the ratio will become $\frac{C}{MF} + \frac{1}{9N}$. Notice that to ensure the same size of *information field* with standard convolution, in any input group of the second layer there should be at least one output channel from each one of the output groups of the first layer, therefore $M \cdot N$ should be less than or equal to the number of output channels from the first layer, i.e., $M \cdot N \leq C$. To further illustrate the relationship between the best parameter efficiency and the choices of M and N, we have the following theorem:

Theorem 1: With the same size of information field*, the best parameter efficiency is achieved if and only if the product of the two group numbers equals the channel number of the intermediate layer.*

As per the theorem, the best parameter efficiency can be achieved only when $M \cdot N = C$. Thus the ratio will become $\frac{N}{F} + \frac{1}{9N}$. When F is a fixed number, N is the only variable which can affect the efficiency. Since $\frac{N}{F} + \frac{1}{9N} \geq \frac{2}{3}\sqrt{\frac{1}{F}}$, the best efficiency can be achieved when $\frac{N}{F} = \frac{1}{9N}$, or $N = \frac{\sqrt{F}}{3}$.

2) Efficiency Given the Total Amount of Parameters: Given the total number of parameters P for one design, both M and N can affect the width of the network. As per Theorem 1, the greatest C can be reached only when $C = M \cdot N$. When $F = \alpha \cdot C$, P can be written as

$$P = 3 \cdot 3 \cdot N \cdot M \cdot N + 1 \cdot 1 \cdot M \cdot \alpha \cdot M \cdot N = MN(9N + \alpha M)$$

$$\geq MN \cdot 2\sqrt{9\alpha MN} = 6\sqrt{\alpha}C^{\frac{3}{2}}$$

Given the number of parameters P, width C has an upper bound when $9N = \alpha M$, which is also the condition for the best efficiency. The greatest width G is $(\frac{P}{6\sqrt{\alpha}})^{\frac{2}{3}}$.

C. $PW + DW + PW$

1) Efficiency Given the Input and the Output: Same as before, given the number of input channels C, bottleneck channels K and output channels F. After applying the design, the total amount of parameters is reduced to $1 \cdot 1 \cdot C \cdot K + 3 \cdot 3 \cdot K + 1 \cdot 1 \cdot K \cdot F = K(C + F + 9)$. The number of parameters for the standard convolution is still $9CF$. Notice that $K = F/4$, therefore the ratio can be further expressed as $\frac{C+F+9}{36C}$. Clearly, given C, K, and F, such a design will also result in a fixed efficiency.

2) Efficiency Given the Total Amount of Parameters: When $F = \alpha \cdot C$ and $K = F/4$, the total number of parameters P will be

$$P = 1 \cdot 1 \cdot C \cdot \frac{\alpha C}{4} + 3 \cdot 3 \cdot \frac{\alpha C}{4} + 1 \cdot 1 \cdot \frac{\alpha C}{4} \cdot \alpha C.$$

When P is fixed, the greatest width G is also fixed, i.e.,

$$\frac{-9\alpha + \sqrt{81\alpha^2 + 16\alpha^2 P + 16\alpha P}}{2(\alpha^2 + \alpha)}.$$

D. $PWG + DW + PWG$

1) Efficiency Given the Input and the Output: We use the same method to evaluate parameter efficiency for this design. First, the number of parameters after applying such a method is $1 \cdot 1 \cdot \frac{C}{M} \cdot K + 3 \cdot 3 \cdot K + 1 \cdot 1 \cdot \frac{K}{N} \cdot F = K(\frac{C}{M} + \frac{F}{N} + 9)$ where M and N represent the group numbers for the two PWGs. The number for the standard convolution is $9CF$. Since $K = F/4$ and as per Theorem 1, the best parameter efficiency can be achieved only when $K = M \cdot N$, and the ratio of parameters can then be represented as $\frac{\frac{C}{M}+4M+9}{36C}$. Thus given C, K, and F, the best parameter efficiency can be reached by setting $\frac{C}{M} = 4M$, or $M = \frac{\sqrt{C}}{2}$.

2) Efficiency Given the Total Amount of Parameters: Similarly, according to the Theorem 1, the greatest C can be reached only when the number of bottleneck channels $K = M \cdot N$. Since $F = \alpha \cdot C$ and $K = F/4$, the total number of parameters of one design P can be expressed as

$$\begin{aligned} P &= 1 \cdot 1 \cdot \frac{4N}{\alpha} \cdot MN + 3 \cdot 3 \cdot MN + 1 \cdot 1 \cdot M \cdot 4MN \\ &= MN(\frac{4N}{\alpha} + 9 + 4M) \\ &\geq MN(9 + 2\sqrt{\frac{16MN}{\alpha}}) = \frac{\alpha}{4}C(9 + 4\sqrt{C}) \end{aligned}$$

Given the number of parameters P, the greatest width G exists when $\alpha M = N$.

V. Model Transformation Scheme

Although our SK scheme can generate SK designs with various efficiency, it remains a question how to apply them to construct a network under a resource constraint. Since people usually rely on an existing model as a reference instead of having a specific resource constraint, we develop a model transformation scheme to utilize SK designs on existing models. According to different functionalities, the scheme can be mainly divided into two parts. The first part (Reduce Parameters/Computation) aims to reduce the parameters/computation from a existing model while preserving the accuracy. The second part (Improve Accuracy) aims to achieve higher accuracy while using a similar amount of parameters/computation as an existing model.

A. Reduce Parameters/Computation

Given an existing model, we reduce the parameters/computation in it by replacing the standard convolution with designs in our SK scheme since all of the designs in our scheme can ensure a same-sized information field with the standard convolution. Specifically, our scheme will start from the first convolutional layer in the model and based on the layer information (the numbers of input and output channels) and the efficiency analysis (Section IV), calculate the efficiency for different SK designs. Then, the design with the best efficiency will be chosen as the replacement for the layer. Similarly, for all the following layers, the corresponding SK designs are chosen. The new model is then constructed based on these SK designs. Notice that in practice, we usually do not replace the first convolutional layer in a model since it does not contain too many parameters.

B. Improve Accuracy

Improving accuracy in an existing model with a close amount of parameters/computation is also an interesting topic in the field. In Section V-A, we already show that by keeping the network width (*information field*) and utilizing SK designs, we can construct a new model with a similar accuracy and much fewer parameters/computation compared to an existing network. Also recall that in Section III, we observe that the model accuracy is positively correlated to the size of the *information field*. Thus, increasing the accuracy with a close amount of parameters/computation could in turn enlarge the *information field* with the saved parameters/computation after using the SK methods. Among various approaches to increase the *information field*, we adopt the most straightforward one in our scheme, which increases the width (the number of channels) of the network. Based on the analysis above, the scheme to improve accuracy can be regarded as an extension for the one to reduce parameters/computation. However, different from that scheme to reduce parameters/computation, here we cannot increase the width of a layer solely based on the information of that single layer since the change of the channel numbers should always be consistent across all adjacent layers. For example, considering two adjacent layers, if we increase the width of the two layers separately based

Algorithm 2 Accuracy Improving Upon an Existing Model

Take as input the layer information in the original model, and we are supposed to generate the new layer information to improve accuracy
$T_{old} \leftarrow 0$
for L in $layers$ **do**
  if L is a convolutional layer **then**
    $T_{old} \leftarrow T_{old} + parameters\_of\_L$
  end if
end for
for L in $layers$ **do**
  if L is a convolutional layer **then**
    Get the channel numbers C and F
    for i in $sparse\_kernel\_designs$ **do**
      Calculate the parameter efficiency
    end for
    Replace L with most efficient SK design
  end if
end for
$T_{new} \leftarrow 0$
for L in $layers$ **do**
  if L is a convolutional layer **then**
    $T_{new} \leftarrow T_{new} + parameters\_of\_L$
  end if
end for
$s \leftarrow \sqrt{\frac{T_{old}}{T_{new}}}$
for L **in** $layers$ **do**
  if L is a convolutional layer **then**
    Get the channel numbers C and F
    $C_{new} \leftarrow s \cdot C$
    $F_{new} \leftarrow s \cdot F$
    Replace L with a layer based on C_{new} and F_{new}
  end if
end for
return $layers$.

on the total number of parameters of each layer, it could result in a situation where the output channel number of the preceding layer is not equal to the input channel number of the succeeding layer. Obviously, this would cause a problem in the network since the output from the preceding layer would be used as the input to the succeeding layer. Also notice that when increasing the number of channels, since we do not want to modify the overall network layout, we would like to increase both the input and the output channels of a layer proportionally. This means the ratio between the number of the input and the output channels should always remain the same. Taking all the circumstances above into consideration, when designing the scheme, we introduce a scale factor s for the width. It can be easily proved that for all the SK methods mentioned in Section III, by increasing the width of a layer by s times the total number of parameters/computation for that layer, the width will become roughly s^2 times of the original layer. Therefore the scheme to improve accuracy, which is also outlined in Algorithm 2, can be summarized as follows. First we calculate the total number of parameters of all convolutional layers T_{old} in the original network. Second, we adopt the scheme of reducing parameters/computation in Section V-A on the original network. After that, the numbers of channels for all layers will remain the same, but the total number of parameters will be reduced to T_{new}. Third, to enlarge the *information field* we apply a same scale factor s across the entire reduced network, then the total number of parameters will become roughly $s^2 \times T_{new}$. Last, since the new network will have a similar amount of parameters as the original one, we can solve for the value of s from the equation $T_{old} = s^2 \times T_{new}$, and according to the value of s, channel numbers for all convolutional layers can be calculated. Finally a new network can be constructed.

VI. Integration With Compilers

After the analysis of our transformation scheme, the potential programming overhead of it is also an important aspect to be considered. Realizing that replacing standard convolutions with SK methods in existing neural networks will also incur extra programming complexities, we integrate our algorithms into a compiler prototype which could automatically calculate the best SK design and finish the code transformation. Our compiler prototype is designed based on the famous machine learning framework TensorFlow and its most popular Python interface.

Generally, as per the transformation scheme, our compiler prototype supports two modes. Mode 1 is to reduce parameters/computation while preserving a similar accuracy, and Mode 2 is to increase accuracy while using a close amount of parameters/computation. The choice of the mode is provided from users as an argument when running our program. Both of the two modes will only demand the source code from users. Then based on users' inputs, the compiler will automatically collect the information needed from the original neural network and according to the information comparison among different SK designs, it will find the best one. Finally, the compiler will also automate the entire code transformation process by replacing standard convolutions with the corresponding best SK methods.

Despite the two different modes supported by the compiler prototype, the main procedure can always be divided into the following four steps. First, generate the AST from the source code. Second, retrieve the information needed for comparisons of different SK designs from the AST. Third, based on comparison results, modify the corresponding AST nodes. Last, convert the modified AST back to readable Python code. Figure 4 shows the overview of our compiler prototype. Sections below will illustrate the four steps in detail.

A. Generating the AST

In both of the two aforementioned modes, information about the channel numbers for all convolutional layers is required for future comparisons of different SK designs. However, directly searching such information in the source code is not applicable since programming languages are ambiguous by nature. Fortunately, AST - which is widely used in compilers to represent the abstract syntactic structure of source code - is

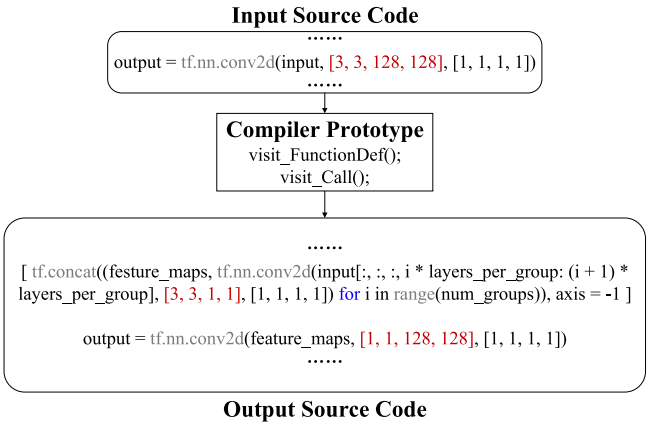


Fig. 4. Overview of the compiler prototype. Red text indicates the kernel size.

introduced to solve this problem. Once built, users can extract information they need from the tree by simply traversing it. In our compiler prototype, we adopt the API *ast.parse()* in the python *ast* module to build an AST from source code stored as a string. The root node on the tree will be returned by the function and further steps can be realized on the tree.

B. Acquiring Information From the AST

According to the model transformation scheme in Section V, the two modes may rely on the same information on the AST. We find this information through pattern matching. The code pattern the compiler looks for is the specific function calls (tf.nn.conv2d()) representing the standard convolutions in TensorFlow.

Specifically, we build a detection gadget by defining a subclass of *ast.NodeVisitor* and overriding *visit* methods inside it. *ast.NodeVisitor* is the primary tool for 'scanning' the tree, and most importantly, it can ensure the same order of function calls as in the source code. An AST node matcher (*ast.Call*) is used inside the method *visit_Call()* to do the matching since a function call will be represented as a *Call* node on the tree, i.e., an instance of a subclass of *AST* module. Once the code pattern is found, the *eval()* function is adopted to collect channel numbers and parameter information of a layer from the argument of *tf.nn.conv2d()*, representing the shape of the kernel.

However, checking the code pattern for *tf.nn.conv2d()* has some intricacies in practice. For the compiler to find specific function calls correctly, an important case should be considered. Specifically, when *tf.nn.conv2d()* is wrapped inside other functions, it will become hard to detect the code pattern. To solve the issue, in our compiler prototype we build a call graph by overriding another *visit_FunctionDef()* method. Basically, the call graph will contain the function dependencies and the relationships between arguments of different functions. Therefore whenever a function call is found during the pattern matching, the compiler is always aware of the potential existence and the corresponding information of *tf.nn.conv2d()* inside it.

C. Modifying the AST

With all the information collected from AST, calculations will be conducted by our compiler prototype to determine the best SK methods. Both of the two modes will involve modifications of the AST nodes.

Similar to Section VI-B, we find all convolutional layers for replacement through pattern matching. Here, we build a detection gadget by defining a subclass of *ast.NodeTransformer* instead of *ast.NodeVisitor* and overriding the method *visit_Call()* inside it. The method should return the original node, a replacement node, or None to remove that node from the tree. The AST node matcher (*ast.Call*) is also used to do the matching. Once the layer is found, the replacement can be realized by constructing a new node inside the same method.

However, for the replacement to work soundly, the case in which *tf.nn.conv2d()* is wrapped should also be considered. Replacing *tf.nn.conv2d()* in this case is complicated since the compiler cannot detect its explicit function call. To solve this problem, we need the call graph built in Section VI-B. Meanwhile we define another *visit_Call()* method to first find all function call nodes through pattern matching and then replace them with nodes containing several sub-nodes, each of which represent a wrapped function call. Afterwards, the compiler should be able to replace all the function calls normally.

D. Converting Back to Python Code

Once we finish modifying the corresponding AST nodes, the next step is to convert the modified AST back to the Python code. Fortunately, with the help from some open source packages or libraries, this step can be easily realized. In our compiler prototype, we adopt the API *unparse.Unparser()* in the *unparse* module for this step. The new Python code file will be saved at the same directory as the original source code. Users will be able to train and evaluate the new model in the file same as before.

VII. Evaluation

In this section, we first conduct an empirical study to validate the idea regarding the *information field* and the final accuracy from two aspects: 1) we show that the relationship between the two properties exists by changing the number of groups within one type of SK design; 2) we observe that the same idea also applies to different SK designs given the same network layout. Second, via experiments, we verify the correctness of our SK scheme by showing that some designs explored by the scheme can achieve performance superior to the state-of-the-art network units. Last, we validate the effectiveness of our compiler by applying it on several state-of-the-art models and then comparing performances of the generated networks along with other existing efficient models.

A. Empirical Study

1) Implementation Details: In this section, to ensure the fair comparison, we adopt the same overall network layout

TABLE I

OVERALL NETWORK LAYOUT. *B* IS THE NUMBER OF BLOCKS AT EACH STAGE. AT THE FIRST BLOCK OF EACH STAGE EXCEPT THE FIRST ONE, DOWN-SAMPLING IS PERFORMED AND THE CHANNEL NUMBER IS DOUBLED

Layer	Output size	KSize	Strides	Repeat
Image	224×224			
Conv1	112×112	3×3	2	1
Max Pool	56×56	3×3	2	1
Stage 1	56×56		1	B
Stage 2	28×28		2	1
	28×28		1	$B-1$
Stage 3	14×14		2	1
	14×14		1	$B-1$
Stage 4	7×7		2	1
	7×7		1	$B-1$
Average Pool	1×1	7×7		1
1000-D FC, Softmax				

TABLE II

COMPARISONS TO ILLUSTRATE THE RELATIONSHIP BETWEEN THE *Information Field* AND THE MODEL ACCURACY WITHIN ONE SK DESIGN. WE TUNE THE NUMBER OF GROUPS TO ACHIEVE DIFFERENT PARAMETER EFFICIENCIES. WIDTH INDICATES THE NUMBER OF INPUT CHANNELS TO THE FIRST STAGE OF THE NETWORK. NUMBERS WITHIN THE PARENTHESES REPRESENT THE NUMBER OF GROUPS

Network Unit	#Params ($\times$M)	Depth	Width	Error (%)
PW+GConv(1)+PW	13.9	98	128	30.0
PW+GConv(32)+PW	13.9	98	256	29.2
PW+GConv(1)+PW	28.4	194	128	29.7
PW+GConv(1)+PW	28.4	98	200	29.3
PW+GConv(2)+PW	28.4	98	256	28.7
PW+GConv(64)+PW	28.4	98	512	**28.4**

for different SK designs. Following the common practice, the design of the layout mainly follows two rules: 1) the overall network contains several stages, and within each stage the output feature map size is the same, and the layers have the same number of filters; 2) at the beginning of each stage, the feature map size is halved, and the number of filters is doubled.

The overall network layout is shown in Table I. Identity mapping [52] is used over each block. When building the models, we can simply replace each block in the layout with the SK designs mentioned in Section III or other state-of-the-art efficient network units. Batch normalization (BN) [53] is adopted right after each layer in the block and as suggested by [48], nonlinear activation ReLU is only performed after the summation of the identity shortcut and the output of each block.

We evaluate our models on the ImageNet 2012 dataset [54], [55], which contains 1.2 million training images and 50k validation images from 1k categories. We follow the same data augmentation scheme in [1], [52], which includes randomized cropping, color jittering, and horizontal flipping. All models are trained for 100 epochs with a batch size of 256. The SGD optimizer is used with the Nesterov momentum. The weight decay is set to 0.0001 and the momentum is 0.9. We adopt a similar weight initialization method from [1], [56], [57]. The learning rate starts at 0.1 and is divided by 10 every 30 epochs. All results reported are single center crop top-1 performances.

2) Information Field Within One Type of SK Design: Recall that in Section III we find an easily measurable property behind various SK designs, the *information field*, that could directly indicate the final accuracy, and observe that the model accuracy is positively correlated to the size of the *information field*. To verify this idea, we choose a bottleneck-like design and conduct some comparisons by tuning different numbers of groups for the intermediate group convolution. It can be easily verified that given the same input, the change of the group number in this design will not affect the size of the *information field* in the output. We adopt the same overall network layout in Table I. Results are shown in Table II.

Specifically, comparing the results in rows 2 and 5, we can see that considering the same-sized input and output for each layer by increasing the number of group from 2 to 32, more than a half of the number of parameters can be saved with only slightly decreased accuracy. Meanwhile, further comparison of rows 5 and 6 indicate that if we apply the saved parameters to increase the model width, the accuracy can be improved. Analyzing the three results together, we can easily find that the first two models contain the same-sized *information field*, whereas the model in row 6 has a larger one, and the reported accuracy is positively correlated to the size of it, which coincides with our idea regarding the *information field*. Also since both of the two models in rows 5 and 6 contain the same amount of parameters, overall network layout, and type of SK design, the accuracy improvement only comes from the increased width (*information field*). A similar phenomenon can also be found via comparison of the results in rows 1 and 2.

We also investigate different usages of parameters. The comparison of results in rows 3 and 4 shows that increasing model width results in higher potential for accuracy improvement than increasing depth. Such results also suggest that the size of *information field* could play a more important role in the model accuracy. Additionally, results in Table II can further explain the correctness of the SK design (PW + DW + PW) in Section III-B3. It directly adopts depthwise convolution in the middle since it can not only ensure an *information field* equal to other group numbers, but is also the most parameter-efficient choice.

3) Information Field Across Different SK Designs: We further validate the idea of *information field* across different SK designs. Specifically, we compare SK designs mentioned in Section III-B. Results are reported in Table III. As mentioned in Section III-B, all designs will share the same-sized *information field* given the same input. For fair comparison, we set a

TABLE III

COMPARISONS TO ILLUSTRATE THE RELATIONSHIP BETWEEN THE *Information Field* AND THE MODEL ACCURACY ACROSS DIFFERENT SK DESIGNS. WIDTH INDICATES THE NUMBER OF INPUT CHANNELS TO THE FIRST STAGE OF THE NETWORK. NUMBERS WITHIN THE PARENTHESES REPRESENT THE NUMBER OF GROUPS

Network Unit	#Params (×M)	Width	Error (%)
Standard Convolution	11.2	64	31.1
DW+PW	0.8	72	31.7
DW+PW	11.2	280	28.5
GConv(4)+PWGConv(32)	11.2	128	30.8
GConv(16)+PWGConv(16)	11.3	256	29.4
PW+DW+PW	11.0	400	26.9
PWGConv(4)+DW+ PWGConv(4)	11.3	560	**25.6**

TABLE IV

COMPARISONS WITH STATE-OF-THE-ART NETWORK UNITS. WIDTH INDICATES THE NUMBER OF INPUT CHANNELS TO THE FIRST STAGE OF THE NETWORK. NUMBERS WITHIN THE PARENTHESES REPRESENT THE NUMBER OF GROUPS. ALL SETTINGS ARE RESTORED FROM THE ORIGINAL PAPERS. IN PARTICULAR, THE BOTTLENECK RATIO IS 1 : 4 FOR RESNET, AND RESNEXT ADOPTS A CARDINALITY OF 16 AND A BOTTLENECK RATIO OF 1 : 2. SHUFFLENET USES A GROUP NUMBER OF 4

Network Unit	#Params (×M)	Width	Error (%)
ResNet [1]	11.2	64	31.3
ResNet with bottleneck [1]	11.3	192	29.9
ResNeXt [58]	11.1	192	29.8
Xception [48]	11.2	280	28.5
ShuffleNet [49]	11.3	560	25.6
GConv(100)+PWGConv(2)	8.6	200	27.0
PWGConv(100)+DW+ PWGConv(2)	10.4	700	**24.9**

close amount of parameters for different designs and compare the final accuracy. From Table III, we can see that due to the differences of parameter efficiency among various SK designs, models with different widths can be constructed, and the accuracy is always positively correlated to the width (size of the *information field*), which again coincides with our previous idea. Meanwhile, the comparison between rows 1 and 2 shows that (DW + PW) could construct a model with a width (*information field*) similar to the standard convolution using less than 1/10 parameter consumption, while still maintaining similar accuracy. This comparison also indicates the importance of the *information field* to the final accuracy. Notice that the results here do not necessarily indicate that one type of SK design is always better than the other in terms of the parameter efficiency since - as per the analysis in Section IV - the efficiency also depends on other factors like the number of groups. For example, considering the same number of parameters and network layout, there could be a combination of group numbers M and N such that the network with the design (GConv(M) + PWGConv(N)) is wider than that of (DW + PW).

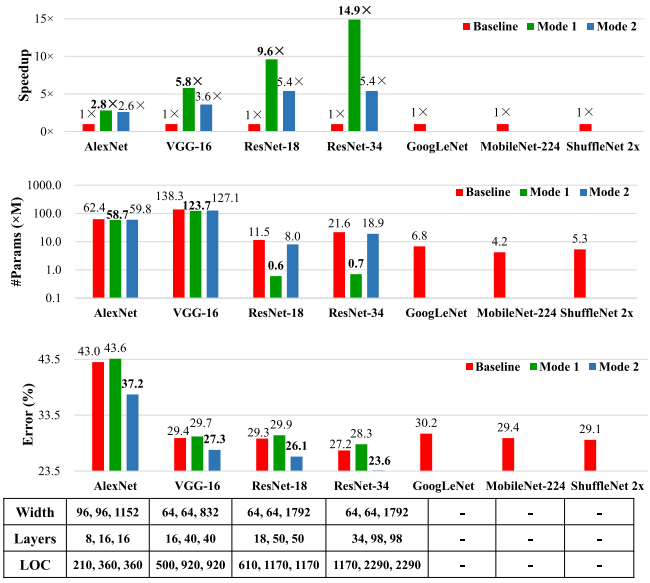


Fig. 5. Comparisons of models before and after the transformation. Width indicates the number of output channels from the first convolutional layer in the network. LOC represents the total line of code used for the model graph.

B. Effectiveness of the Sparse Kernel Scheme

We validate the effectiveness of our spare kernel scheme by comparing the performances of the SK designs to the state-of-the-art network units. All these units chosen can not only achieve state-of-the-art accuracies but also have a relatively small model sizes. Table IV shows the results. Specifically, for fair comparison, we use the same network layout as shown in Table I and replace blocks in it with corresponding designs or units. The model size around 11.0M is selected, as it is the size that different models (e.g., Xception, ResNeXt and ShuffleNet) can be easily configured to simultaneously. Results in rows 6 and 7 show that the SK designs found by our scheme can yield better accuracy with a smaller model size, which then validates the effectiveness of the scheme. Also notice that the choices of group numbers used in the SK designs are chosen to help accommodate both the similar model size and the overall network layout, which may not be the most efficient ones that are expected to result in a wider network (larger *information field*) with better accuracy under the same limitation of parameters.

C. Effectiveness of the Compiler Prototype

To validate the effectiveness of the compiler prototype, we choose 4 different state-of-the-art models: AlexNet, VGG-16, ResNet-18, and ResNet-34. Each model will be applied with two different modes of the compiler. We also report

performances from some existing efficient models for comparison. Results are shown in Figure 5.

From Figure 5, we can see that generally, all models using Mode 1 will have slightly decreased accuracy due to the reduction of parameters, while models with Mode 2 will show consistent accuracy improvement thanks to the increased model width (*information field*). In particular, ResNet-34 with Mode 1 can achieve better accuracy than all existing efficient models with much less parameter consumption. Such results validate the effectiveness of our compiler prototype and also coincide with our idea regarding the *information field*.

Looking at the number of parameters from different models, we can observe that by using Mode 1, AlexNet and VGG-16 can only reduce a small amount of parameters while ResNets show a significant parameter reduction. We believe this huge difference is caused by the fully-connected (FC) layers. Both AlexNet and VGG-16 have three FC layers which consumes much more parameters than the convolutional layers. However, ResNets only have one FC layer, and its parameters take up a small portion of the entire network.

Comparisons of the speed-ups from different models show that the speed-up of SK designs is not scalable to the parameter reduction. We argue that this is due to the lack of optimized implementations of SK designs in TensorFlow library, therefore the codes of new models generated by our compiler prototype may not guarantee the best run-time performances. Future optimization on the implementations of SK designs should expect better performances.

VIII. Related Work

A. Pruning

Pruning [4], [6]–[19] is used to compress a model while preserving the accuracy by reducing redundant weights, network connections or channels in a pre-trained model. However, sometimes pruning can face difficulties when deploying on hardware like GPUs since, as indicated in [4], some pruning approaches like [6] may only be effective when the weight matrices are sufficiently sparse.

B. Quantization

Quantization [20]–[32] is another method to compress the model without losing the accuracy; it reduces the number of bits required to represent the weights. However, this technique will require specialized hardware support for the actual speedup on device.

C. Low Rank Approximation

Low rank approximation [33]–[43] uses two or more matrices to approximate the original matrix values in a pre-trained model. For example, one can decompose a pre-trained 4D kernel into the product of two 3D matrices. By replacing the original matrix with more low-rank matrices, some parameters/computation can be saved. However, since the decomposition is just an approximation of the original matrix values, maintaining a similar accuracy will always need additional model re-training.

IX. Conclusion

In this paper, to facilitate the direct deployment of deep learning models on 5G edge devices, we first present an SK scheme to craft an effective SK design by eliminating the large design space from three aspects: composition, performance, and efficiency. During the process to reduce the design space, we find a unified property named the *information field* behind various designs, which can directly indicate the final accuracy. Meanwhile, we give the detailed efficiency analysis for the final 4 designs in the scheme. Second, based on the analysis, we propose a model transformation scheme to utilize the SK designs on existing models. Last, considering the extra programming overhead and the expert knowledge required by the model transformation scheme, we develop a compiler prototype to automate the entire process. The empirical studies show that under the same overall network layouts models composed of the sparse kernel designs searched by our search scheme can beat the state-of-the-art models in terms of the accuracy and the efficiency. And the model transformation scheme can easily improve either the model accuracy (the same number of parameters) or the efficiency (the same accuracy) upon existing state-of-the-art models.

For future studies, we plan to propose a more accurate efficiency analysis by adopting a better representation of the computational efficiency such as the runtime instead of the computational FLOPs. However, measuring the actual runtime for each design takes ample time, which contradicts the goal of the paper. Recent studies show that it is possible to utilize the power of the machine learning to predict the actual runtime by training a neural network [59]. Besides, we are also interested in predicting the runtime on various platforms and devices both efficiently and accurately.

References

[1] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Jun. 2016, pp. 770–778.

[2] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "ImageNet classification with deep convolutional neural networks," *Commun. ACM*, vol. 60, no. 6, pp. 84–90, May 2017.

[3] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," 2014, *arXiv:1409.1556*. [Online]. Available: http://arxiv.org/abs/1409.1556

[4] S. Han, H. Mao, and W. J. Dally, "Deep compression: Compressing deep neural networks with pruning, trained quantization and Huffman coding," 2015, *arXiv:1510.00149*. [Online]. Available: http://arxiv.org/abs/1510.00149

[5] L. Xiao, X. Wan, C. Dai, X. Du, X. Chen, and M. Guizani, "Security in mobile edge caching with reinforcement learning," *IEEE Wireless Commun.*, vol. 25, no. 3, pp. 116–122, Jun. 2018.

[6] Y. LeCun, J. S. Denker, and S. A. Solla, "Optimal brain damage," in *Proc. Adv. Neural Inf. Process. Syst.*, 1990, pp. 598–605.

[7] S. Han, J. Pool, J. Tran, and W. Dally, "Learning both weights and connections for efficient neural network," in *Proc. Adv. Neural Inf. Process. Syst.*, 2015, pp. 1135–1143.

[8] W. Wen, C. Wu, Y. Wang, Y. Chen, and H. Li, "Learning structured sparsity in deep neural networks," in *Proc. Adv. Neural Inf. Process. Syst.*, 2016, pp. 2074–2082.

[9] A. Ardakani, C. Condo, and W. J. Gross, "Sparsely-connected neural networks: Towards efficient VLSI implementation of deep neural networks," 2016, *arXiv:1611.01427*. [Online]. Available: http://arxiv.org/abs/1611.01427

[10] Z. Liu, J. Li, Z. Shen, G. Huang, S. Yan, and C. Zhang, "Learning efficient convolutional networks through network slimming," in *Proc. IEEE Int. Conf. Comput. Vis. (ICCV)*, Oct. 2017, pp. 2755–2763.

[11] H. Li, A. Kadav, I. Durdanovic, H. Samet, and H. Peter Graf, "Pruning filters for efficient ConvNets," 2016, *arXiv:1608.08710*. [Online]. Available: http://arxiv.org/abs/1608.08710

[12] Y. He, X. Zhang, and J. Sun, "Channel pruning for accelerating very deep neural networks," in *Proc. IEEE Int. Conf. Comput. Vis. (ICCV)*, Oct. 2017, vol. 2, no. 6, pp. 1398–1406.

[13] B. Liu, M. Wang, H. Foroosh, M. Tappen, and M. Penksy, "Sparse convolutional neural networks," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Jun. 2015, pp. 806–814.

[14] J.-H. Luo, J. Wu, and W. Lin, "ThiNet: A filter level pruning method for deep neural network compression," 2017, *arXiv:1707.06342*. [Online]. Available: http://arxiv.org/abs/1707.06342

[15] W. Wen, C. Xu, C. Wu, Y. Wang, Y. Chen, and H. Li, "Coordinating filters for faster deep neural networks," in *Proc. IEEE Int. Conf. Comput. Vis. (ICCV)*, Oct. 2017, pp. 658–666.

[16] P. Molchanov, S. Tyree, T. Karras, T. Aila, and J. Kautz, "Pruning convolutional neural networks for resource efficient inference," 2016, *arXiv:1611.06440*. [Online]. Available: http://arxiv.org/abs/1611.06440

[17] X. Sun, X. Ren, S. Ma, and H. Wang, "meProp: Sparsified back propagation for accelerated deep learning with reduced overfitting," 2017, *arXiv:1706.06197*. [Online]. Available: http://arxiv.org/abs/1706.06197

[18] R. Spring and A. Shrivastava, "Scalable and sustainable deep learning via randomized hashing," in *Proc. 23rd ACM SIGKDD Int. Conf. Knowl. Discovery Data Mining*, Aug. 2017, pp. 445–454.

[19] Y. Lin, C. Sakr, Y. Kim, and N. Shanbhag, "PredictiveNet: An energy-efficient convolutional neural network via zero prediction," in *Proc. IEEE Int. Symp. Circuits Syst. (ISCAS)*, May 2017, pp. 1–4.

[20] D. Soudry, I. Hubara, and R. Meir, "Expectation backpropagation: Parameter-free training of multilayer neural networks with continuous or discrete weights," in *Proc. Adv. Neural Inf. Process. Syst.*, 2014, pp. 963–971.

[21] M. Rastegari *et al.*, "Xnor-net: Imagenet classification using binary convolutional neural networks," in *Proc. Eur. Conf. Comput. Vis.* Cham, Switzerland: Springer, 2016, pp. 525–542.

[22] J. Wu, C. Leng, Y. Wang, Q. Hu, and J. Cheng, "Quantized convolutional neural networks for mobile devices," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Jun. 2016, pp. 4820–4828.

[23] S. Zhou, Y. Wu, Z. Ni, X. Zhou, H. Wen, and Y. Zou, "DoReFa-net: Training low bitwidth convolutional neural networks with low bitwidth gradients," 2016, *arXiv:1606.06160*. [Online]. Available: http://arxiv.org/abs/1606.06160

[24] A. Zhou, A. Yao, Y. Guo, L. Xu, and Y. Chen, "Incremental network quantization: Towards lossless CNNs with low-precision weights," 2017, *arXiv:1702.03044*. [Online]. Available: http://arxiv.org/abs/1702.03044

[25] M. Courbariaux, Y. Bengio, and J.-P. David, "Binaryconnect: Training deep neural networks with binary weights during propagations," in *Proc. Adv. Neural Inf. Process. Syst.*, 2015, pp. 3123–3131.

[26] M. Courbariaux, I. Hubara, D. Soudry, R. El-Yaniv, and Y. Bengio, "Binarized neural networks: Training deep neural networks with weights and activations constrained to $+1$ or -1," 2016, *arXiv:1602.02830*. [Online]. Available: http://arxiv.org/abs/1602.02830

[27] L. Deng, P. Jiao, J. Pei, Z. Wu, and G. Li, "GXNOR-net: Training deep neural networks with ternary weights and activations without full-precision memory under a unified discretization framework," *Neural Netw.*, vol. 100, pp. 49–58, Apr. 2018.

[28] P. Micikevicius *et al.*, "Mixed precision training," 2017, *arXiv:1710.03740*. [Online]. Available: http://arxiv.org/abs/1710.03740

[29] C. Leng, H. Li, S. Zhu, and R. Jin, "Extremely low bit neural network: Squeeze the last bit out with ADMM," 2017, *arXiv:1707.09870*. [Online]. Available: http://arxiv.org/abs/1707.09870

[30] W. Wen *et al.*, "Terngrad: Ternary gradients to reduce communication in distributed deep learning," in *Proc. Adv. Neural Inf. Process. Syst.*, 2017, pp. 1509–1519.

[31] C. Xu *et al.*, "Alternating multi-bit quantization for recurrent neural networks," 2018, *arXiv:1802.00150*. [Online]. Available: http://arxiv.org/abs/1802.00150

[32] S. Wu, G. Li, F. Chen, and L. Shi, "Training and inference with integers in deep neural networks," 2018, *arXiv:1802.04680*. [Online]. Available: http://arxiv.org/abs/1802.04680

[33] E. L. Denton, W. Zaremba, J. Bruna, Y. LeCun, and R. Fergus, "Exploiting linear structure within convolutional networks for efficient evaluation," in *Proc. Adv. Neural Inf. Process. Syst.*, 2014, pp. 1269–1277.

[34] M. Jaderberg, A. Vedaldi, and A. Zisserman, "Speeding up convolutional neural networks with low rank expansions," 2014, *arXiv:1405.3866*. [Online]. Available: http://arxiv.org/abs/1405.3866

[35] V. Lebedev, Y. Ganin, M. Rakhuba, I. Oseledets, and V. Lempitsky, "Speeding-up convolutional neural networks using fine-tuned CP-decomposition," 2014, *arXiv:1412.6553*. [Online]. Available: http://arxiv.org/abs/1412.6553

[36] J. Jin, A. Dundar, and E. Culurciello, "Flattened convolutional neural networks for feedforward acceleration," 2014, *arXiv:1412.5474*. [Online]. Available: http://arxiv.org/abs/1412.5474

[37] M. Wang, B. Liu, and H. Foroosh, "Design of efficient convolutional layers using single intra-channel convolution, topological subdivisioning and spatial 'Bottleneck' structure," 2016, *arXiv:1608.04337*. [Online]. Available: http://arxiv.org/abs/1608.04337

[38] J. Xue, J. Li, D. Yu, M. Seltzer, and Y. Gong, "Singular value decomposition based low-footprint speaker adaptation and personalization for deep neural network," in *Proc. IEEE Int. Conf. Acoust., Speech Signal Process. (ICASSP)*, May 2014, pp. 6359–6363.

[39] A. Novikov, D. Podoprikhin, A. Osokin, and D. P. Vetrov, "Tensorizing neural networks," in *Proc. Adv. Neural Inf. Process. Syst.*, 2015, pp. 442–450.

[40] T. Garipov, D. Podoprikhin, A. Novikov, and D. Vetrov, "Ultimate tensorization: Compressing convolutional and FC layers alike," 2016, *arXiv:1611.03214*. [Online]. Available: http://arxiv.org/abs/1611.03214

[41] Y.-D. Kim, E. Park, S. Yoo, T. Choi, L. Yang, and D. Shin, "Compression of deep convolutional neural networks for fast and low power mobile applications," 2015, *arXiv:1511.06530*. [Online]. Available: http://arxiv.org/abs/1511.06530

[42] Y. Yang, D. Krompass, and V. Tresp, "Tensor-train recurrent neural networks for video classification," 2017, *arXiv:1707.01786*. [Online]. Available: http://arxiv.org/abs/1707.01786

[43] J. M. Alvarez and M. Salzmann, "Compression-aware training of deep networks," in *Proc. Adv. Neural Inf. Process. Syst.*, 2017, pp. 856–867.

[44] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," in *Proc. Adv. Neural Inf. Process. Syst.*, 2012, pp. 1097–1105.

[45] A. G. Howard *et al.*, "MobileNets: Efficient convolutional neural networks for mobile vision applications," 2017, *arXiv:1704.04861*. [Online]. Available: http://arxiv.org/abs/1704.04861

[46] L. Sifre and S. Mallat, "Rigid-motion scattering for texture classification," 2014, *arXiv:1403.1687*. [Online]. Available: http://arxiv.org/abs/1403.1687

[47] M. Lin, Q. Chen, and S. Yan, "Network in network," 2013, *arXiv:1312.4400*. [Online]. Available: http://arxiv.org/abs/1312.4400

[48] F. Chollet, "Xception: Deep learning with depthwise separable convolutions," 2016, *arXiv:1610.02357*. [Online]. Available: http://arxiv.org/abs/1610.02357

[49] X. Zhang, X. Zhou, M. Lin, and J. Sun, "ShuffleNet: An extremely efficient convolutional neural network for mobile devices," 2017, *arXiv:1707.01083*. [Online]. Available: http://arxiv.org/abs/1707.01083

[50] T. Subramanya, R. Riggio, and T. Rasheed, "Intent-based mobile backhauling for 5G networks," in *Proc. 12th Int. Conf. Netw. Service Manage. (CNSM)*, Oct. 2016, pp. 348–352.

[51] L. Wu, X. Du, W. Wang, and B. Lin, "An out-of-band authentication scheme for Internet of Things using blockchain technology," in *Proc. Int. Conf. Comput., Netw. Commun. (ICNC)*, Mar. 2018, pp. 769–773.

[52] K. He *et al.*, "Identity mappings in deep residual networks," in *Proc. Eur. Conf. Comput. Vis.* Cham, Switzerland: Springer, 2016, pp. 630–645.

[53] S. Ioffe and C. Szegedy, "Batch normalization: Accelerating deep network training by reducing internal covariate shift," 2015, *arXiv:1502.03167*. [Online]. Available: http://arxiv.org/abs/1502.03167

[54] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei, "ImageNet: A large-scale hierarchical image database," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, Jun. 2009, pp. 248–255.

[55] O. Russakovsky *et al.*, "ImageNet large scale visual recognition challenge," *Int. J. Comput. Vis.*, vol. 115, no. 3, pp. 211–252, Dec. 2015.

[56] K. He, X. Zhang, S. Ren, and J. Sun, "Delving deep into rectifiers: Surpassing human-level performance on ImageNet classification," in *Proc. IEEE Int. Conf. Comput. Vis. (ICCV)*, Dec. 2015, pp. 1026–1034.

[57] G. Huang, Z. Liu, L. van der Maaten, and K. Q. Weinberger, "Densely connected convolutional networks," 2016, *arXiv:1608.06993*. [Online]. Available: http://arxiv.org/abs/1608.06993

[58] S. Xie, R. Girshick, P. Dollár, Z. Tu, and K. He, "Aggregated residual transformations for deep neural networks," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Jul. 2017, pp. 5987–5995.

[59] H. Cai, L. Zhu, and S. Han, "ProxylessNAS: Direct neural architecture search on target task and hardware," 2018, *arXiv:1812.00332*. [Online]. Available: http://arxiv.org/abs/1812.00332

Kun Wan received the bachelor's degree from the Harbin Institute of Technology. He is currently pursuing the Ph.D. degree with the Department of Computer Science, University of California at Santa Barbarba. His research interests include machine learning, computer vision, and program systems, especially building intelligent programming systems for resource-efficient learning.

Xiaolei Liu received the Ph.D. degree in software engineering from the University of Electronic and Technology of China (UESTC). He is currently an Assistant Research Fellow with the Institute of Computer Application, China Academy of Engineering Physics. His research interests include AI security, adversarial example, and swarm intelligence optimization algorithm.

Jianyu Yu is currently pursuing the bachelor's degree with the Department of Computer Science, University of California at Santa Barbarba. His research interests include machine learning, computer vision, and data privacy.

Xiaosong Zhang received the B.S. degree in dynamics engineering from Shanghai Jiao Tong University, Shanghai in 1990, and the M.S. and Ph.D. degrees in computer science from the University of Electronic and Technology of China (UESTC), Chengdu, in 2011. He is currently a Professor in Computer Science at UESTC. He has worked on numerous projects in both research and development roles. These projects include device security, intrusion detection, malware analysis, software testing, and software verification. He has coauthored a number of research articles on computer security. His current research interests include software reliability, software vulnerability discovering, software test case generation, and reverse engineering.

Xiaojiang Du (Fellow, IEEE) received the B.S. and M.S. degrees in electrical engineering from the Automation Department, Tsinghua University, Beijing, China, in 1996 and 1998, respectively, and the M.S. and Ph.D. degrees in electrical engineering from the University of Maryland, College Park, in 2002 and 2003, respectively. He is currently a tenured Full Professor and the Director of the Security and Networking (SAN) Laboratory, Department of Computer and Information Sciences, Temple University, Philadelphia, PA, USA. His research interests include security, wireless networks, and systems. He has authored over 400 journal and conference papers in these areas, as well as a book published by Springer. He has been awarded more than six million U.S. Dollars research grants from the U.S. National Science Foundation (NSF), Army Research Office, Air Force Research Lab, NASA, the State of Pennsylvania, and Amazon. He won the Best Paper Award at IEEE GLOBECOM 2014 and the best poster runner-up award at the ACM MobiHoc 2014. He is a Life Member of ACM. He is (was) a Technical Program Committee (TPC) member of several premier ACM/IEEE conferences, such as INFOCOM (2007–2020), IM, NOMS, ICC, GLOBECOM, WCNC, BroadNet, and IPCCC. He serves on the editorial boards of two international journals. He served as the Lead Chair for the Communication and Information Security Symposium of the IEEE International Communication Conference (ICC) 2015, and the Co-Chair of Mobile and Wireless Networks Track of IEEE Wireless Communications and Networking Conference (WCNC) 2015.

Nadra Guizani received the Ph.D. degree from Purdue University, USA. Her Ph.D. research work revolved around prediction and access control of disease spread data on dynamic network topologies. She is currently an Assistant Professor with the School of Electrical Engineering & Computer Science, Washington State University, USA. Her research interests include machine learning, mobile networking, large data analysis, and prediction techniques. She is an Active Member of both the Women in Engineering Program and the Computing Research Association (CRA).