# Modified Deep Neural Network for Object Recognition

Dulari Bhatt[1], Chirag Patel[2*], Madhuri Chopade[3], Madhvi Dave[4], Chintan Patel[5]
Research Scholar, Computer Science & Engineering, Parul University, Vadodara, Gujarat, India[1]
Associate Professor, Computer Science, Charusat University, Gujarat, India[2]
Assistant Professor, Information Technology, Gandhinagar University, Gujarat, India[3]
Associate Professor, Computer Science, Adani Institute of Digital Technology Management, Gujarat, India[4]
Academic Associate, IIM Ahmedabad, Gujarat, India[5]

***Abstract*—Object recognition has gained significance due to the rise in CCTV surveillance and the need for automated detection of objects or activities in images and videos. Lightweight process frameworks are in demand for sensor networks. While Convolutional Neural Networks (CNNs) are widely used in computer vision, many existing architectures are specialized. This paper introduces the Dimension-Based Generic Convolution Block (DBGC), enhancing CNNs with dimension-wise selection of kernels for improved performance. The DBGC offers flexibility for height, width, and depth kernels and can be applied to different dimension combinations. A key feature is the dimension selector block. Unoptimized kernel dimensions reduce computational operations and accuracy, while semi-optimized ones maintain accuracy with fewer operations. Optimized dimensions provide 5-6% higher accuracy and reduced operations. This work addresses the challenge of generic architecture in object recognition research.**

***Keywords*—Convolutional Neural Network (CNN); depth-wise separable convolution; dimension-based generic convolution unit (DBGC); CNN architecture**

## I. Introduction

The Convolutional Neural Network (CNN) stands as a widely adopted architecture within the realm of computer vision, serving pivotal roles in tasks like object recognition and detection [1]. Researchers have delved into a myriad of modifications for CNNs in the past, encompassing areas such as activation functions, regularization techniques, parameter tuning, and architectural advancements. In the following section, we will delve into the latest developments in this ever evolving field.

This paper introduces a groundbreaking innovation known as the Dimension-Based Generic Convolution Unit (DBGC), which operates as a dimension selector module. Remarkably versatile, the DBGC can be seamlessly integrated into various architectural frameworks, yielding reductions in Floating Point Operations Per Second (FLOPs) without compromising accuracy. The research makes a significant contribution by presenting both semi-optimized and fully optimized kernel methods, effectively curtailing FLOPs while either maintaining or improving model accuracy.

The applications of computer vision and image processing extend across a multitude of domains, including but not limited to traffic surveillance, object detection, autonomous vehicles, agriculture, and healthcare [2], [3]. A crucial aspect of computer vision tasks is precise feature extraction. Fusion methods for feature extraction, as mentioned in b4 [4], enhance performance. The paper identifies inspiration from networks like ShuffleNetv2 [5], ESPNetv2 [6], DiCENet [7], and MobileNetv2 [8]. The subsequent sections elaborate on these networks' core architectures, strengths, and weaknesses.

### A. ShuffleNetv2

ShuffleNetv2, presented in the 2018 paper "ShuffleNetv2: Practical Guidelines for Efficient CNN Architecture Design" by Ma, Zhang, Zheng, and Sun, introduces a novel approach for evaluating computational complexity [5]. In addition to the conventional FLOPs metric, ShuffleNetv2 considers factors like speed, memory access costs, and platform-specific parallelism. Unlike FLOPs alone, which may not accurately reflect network speed due to variables such as memory access and target platform differences, this approach provides a more comprehensive assessment of efficiency. To address these limitations, ShuffleNetv2 introduces four key principles for network design:

- Proportional Input and Output Channels: Maintaining a 1:1 ratio between input and output channels minimizes memory access costs.
- Optimal Group Convolution: Carefully selecting the number of groups in convolutions prevents excessive memory access costs.
- Reduced Network Fragmentation: Minimizing network fragmentation enhances parallelism efficiency for better parallel computations.
- Significance of Element-wise Operations: Despite low FLOPs, element-wise operations can significantly increase memory access time.

The combined application of these principles enhances CNN architecture efficiency beyond FLOPs considerations. ShuffleNetv2 integrates these principles into its design to improve network efficiency, as illustrated in Fig. 1.

### B. ESPNetv2

ESPNetv2 is a progression from ESPNetv1, initially designed for semantic segmentation [6]. It broadens the application of ESPNetv1's ideas to diverse computer vision tasks, even encompassing language modeling. The central aim

*Corresponding Author

is refining the separation of dilated convolutions in a depth-wise fashion. This is realized through the introduction of a new module called EESP (Extremely Efficient Spatial Pyramid), outlined in Fig. 2.

The EESP block in ESPNetv2 divides channels into two groups, using one as an identity and maintaining channel balance during convolutions. It avoids group-wise convolutions and includes ReLU concatenations and depth-wise convolutions [6]. The architecture begins with stride-2 convolutions and has three phases, each starting with a stride EESP block, altering spatial dimensions and channels. Afterward, there are convolution layers, global average pooling, and a classifier. Strided EESP uses dilated convolutions and concatenation for long-range connections, supported by depth-wise convolutions for spatial information. ESPNetv2 achieves about 74% Top1 accuracy with 5.9 million parameters.

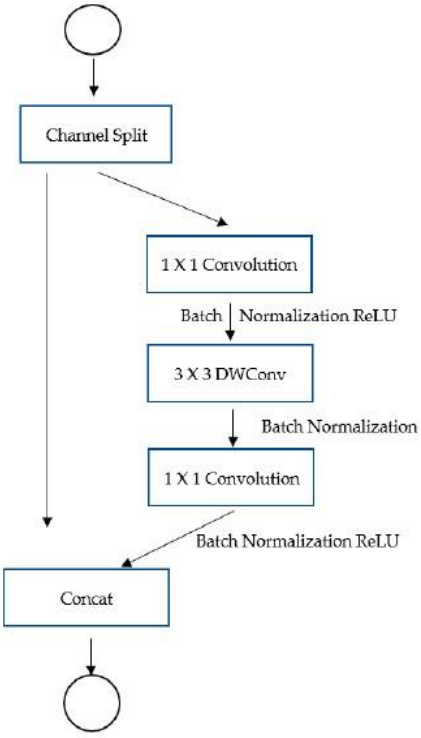


Fig. 1. ShuffleNetv2 design.

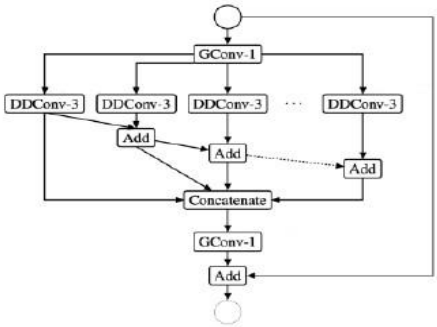


Fig. 2. Building block of EESP.

### C. DiCENet

DiCENet introduces the concept of dimension-wise convolution and fusion to replace regular convolution. Depth wise and group convolutions divide the input tensor along the channel dimension, allowing each filter to operate on specific channel subsets [7]. DiCENet suggests extending this slicing approach to width and height dimensions. Depth-wise convolution has a drawback of requiring multiple convolutions to effectively mix channels, constituting a substantial portion of operations in models like MobileNet.

ShuffleNet attempted to mitigate this with grouped convolution for speed, but further improvement might be possible. DiCENet proposes dimension-wise convolution (DimConv) across all three possible axes (DHW): depth-wise (HW), widthwise (DH), and height-wise (DW). Depth-wise covers spatial dimensions, widthwise involves splitting along channels while sliding over image width, and height-wise does the same along the height axis. This approach aims to enhance convolution efficiency and performance.

### D. MobileNetv2

MobileNetv2 maintains the use of depth-wise convolution, a characteristic shared with its predecessor, MobileNetv1 [8]. However, in MobileNetv2, the arrangement of blocks is restructured, notably placing the depth-wise convolution block at the center, as depicted in Fig. 3. Preceding the depth wise layer is a 1x1 convolution referred to as the expansion layer, which augments the channel count. Following the depth wise layer, there is another 1x1 convolution, identified as the projection layer or bottleneck layer. This final convolution reduces the number of channels back to a more optimal configuration. This architectural adjustment aims to enhance the model's performance and efficiency.

MobileNetv2 retains the depth-wise convolution approach from its predecessor, MobileNetv1 [8]. However, MobileNetv2 introduces a novel block arrangement. Notably, it places the depth-wise convolution block at the center, as shown in Fig. 3. Preceding the depth-wise layer is a 1x1 convolution known as the expansion layer, which increases channel numbers. Subsequent to the depth-wise layer, another 1x1 convolution, called the projection or bottleneck layer, is employed. This final convolution reduces channel counts to an optimal level. This architectural shift is designed to boost both the performance and efficiency of the model.

MobileNetv2 preserves ReLU6 as its activation function, as in its predecessor. Yet, beyond the bottleneck layer, it opts not to use any activation function [8], termed "linear bottlenecks." This choice is informed by the risk of losing vital information due to non-linearity in this low-dimensional data context.

Together, these elements constitute the architecture of MobileNetV2. Subsequently, the architecture involves a conventional 1x1 convolution layer, global average pooling, and a classification layer, as depicted in Fig. 3. Fig. 4 furnishes the Top1 and Top5 accuracy metrics for a range of lightweight CNN architectures.

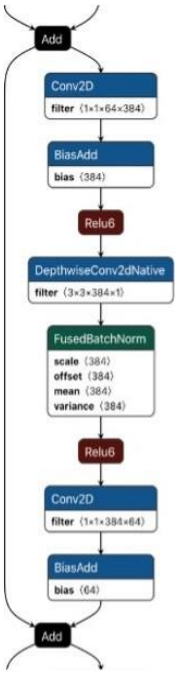


Fig. 3. MobileNetv2.

| Architecture | Year | Parameters | Top1 | Top5 |
|---|---|---|---|---|
| ShuffleNetv2 [6] | 2018 | 2.3 M | 69.4 | 88.9 |
| MobileNetv2 [7] | 2018 | 3.47 M | 71.8 | 91.0 |
| ESPNetv2 [8] | 2019 | 3.49 M | 72.06 | 90.39 |
| DiCENet [9] | 2020 | 2.65 M | 69.05 | 88.8 |

Fig. 4. A comparison of different lightweight architectural designs.

In Section III of the manuscript, the authors present the DBGC (Dimension-based Generic Convolution) block. A comprehensive grasp of the concepts expounded upon in Section II is essential for a complete comprehension of this section. Section II is bifurcated into two key segments: the initial part elucidates the separable convolution technique and its diverse modifications, while the subsequent part delves into an array of convolutional kernels. Moving forward, Section IV provides a detailed examination of the results and analysis pertaining to the recently introduced DBGC block.

## II. Materials and Methods

This section provides a clear explanation of crucial terminologies and their relevance in the context of the proposed DBGC block. The section is structured into two subsections: the first one covers separable convolution, while the second one delves into depth-wise separable convolution.

### A. Introduction to Separable Convolution

The inception of separable convolution dates back to the Xception model of 2016 [9]. In response to the growing trend of deep learning (DL) moving towards edge computing, especially on smart phones and IoT devices, researchers proposed various techniques to enhance inferential computation efficiency. These techniques encompass network pruning, parameter compression, and more. One particularly effective approach is quantization, which streamlines DL processes by enabling them to operate on fixed-point pipelines. The renowned lightweight architecture, MobileNetv1, has notably leveraged separable convolution to substantially reduce parameter size and computation latency [9].

Separable convolutions can be categorized into two main types: (1) Spatial separable convolution and (2) Depth-wise separable convolution.

*1) Spatial separable convolution*: Spatial separable convolution simplifies convolution by breaking it into two separate convolutions [18]. While conceptually straightforward, its practical application in deep learning is limited due to certain drawbacks. The term "spatial separable convolution" reflects its focus on width and height dimensions in images and kernels [10], with image channels representing the "depth" dimension. This approach splits a kernel into smaller parts; for example, a 3x3 kernel is divided into 3x1 and 1x3 kernels, as shown in Fig. 5.

$$\begin{bmatrix} 2 & 4 & 6 \\ 3 & 6 & 9 \\ 4 & 8 & 12 \end{bmatrix} = \begin{bmatrix} 2 \\ 3 \\ 4 \end{bmatrix} X \begin{bmatrix} 1 & 2 & 3 \end{bmatrix}$$

Fig. 5. Dividing a 3x3 kernel into spatial.

Achieving the same outcome involves conducting two convolutions, each with three multiplications, totaling six, as opposed to a single convolution with nine multiplications. This approach reduces computational complexity by minimizing the number of multiplications, resulting in enhanced network efficiency, as illustrated in Fig. 6.

A notable limitation of a spatial separable kernel is its inability to effectively divide all kernels into two parts. This limitation becomes particularly problematic during training, as it utilizes only a small fraction of the entire network.

*2) Depth-wise separable convolution*: When working with kernels that cannot be decomposed into smaller components, the adoption of depth-wise separable.

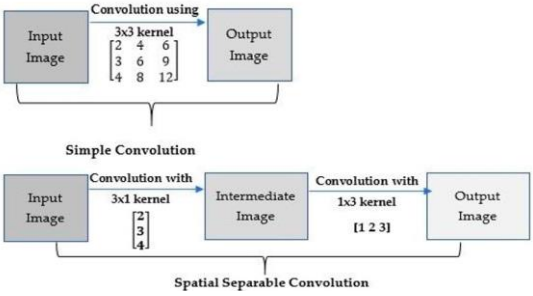


Fig. 6. Basic and spatially separable convolution.

Convolution becomes increasingly popular. Facilitating its implementation are tools such as keras.layers.SeparableConv2D or tf.layers.separable_conv2d. The term "depth-wise separable convolution" reflects its attention to both spatial dimensions and the depth dimension (number of channels) [11]. In an input image with RGB channels, each channel (R, G, B) provides a distinct interpretation of the image. After multiple convolutions, the image can comprise a variety of channels [12]. For instance, the "red" channel focuses on redness, "blue" on blueness, and

"green" on greenness. A 64-channel image can be understood in 64 unique ways. In depth-wise separable convolution, similarly to spatial separable convolution, a kernel is divided into two separate kernels. One kernel handles depth-wise convolution, while the other manages point-wise convolution.

*a) Depth-Wise Convolution*: Depth-wise convolution applies convolutional kernels to an input image without modifying its depth, as elucidated in [13]. This process, illustrated in Fig. 7, is achieved through the utilization of three distinct kernels. For example, when dealing with a 12x12x3 input image, three 5x5x1 kernels are employed. Each 5x5x1 kernel traverses a channel of the input image, computing scalar products for every group of 25 pixels (5x5). Consequently, this generates an output image of dimensions 8x8x1, as visually depicted in Fig. 7.

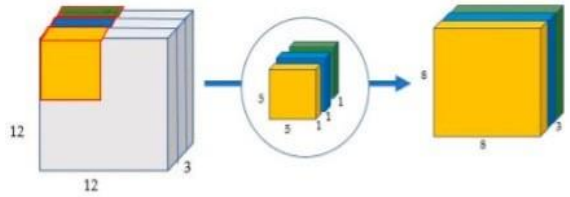


Fig. 7. Depth-wise convolution employs three kernels to transform a 12 × 12 × 1 image into an 8 × 8 × 1 image.

*b) Point-Wise Convolution*: Following depth-wise convolution, the image transforms from 12x12x3 to 8x8x3, necessitating a channel expansion. "Point-wise convolution" uses a 1x1 kernel, processing individual points. The kernel's depth matches input image channels [14]. For an 8x8x1 result, a 1x1x3 kernel moves through the 8x8x3 image, as shown in Fig. 8.

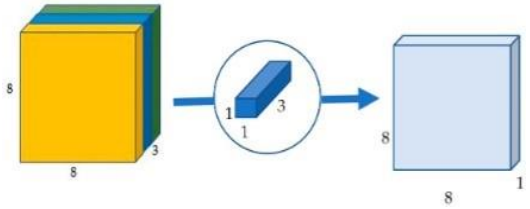


Fig. 8. Point-wise convolution reduces a three-channel image to a single-channel image.

In a normal convolution, considering 256 kernels of 5x5x3 over 8x8 movements requires 1,228,800 multiplications. In separable convolution, using three 5x5x1 kernels for depth wise convolution in 8x8 movements entails 4,800 multiplications, and point-wise convolution with 256 kernels of 1x1x1x3 needs 49,152 multiplications. The total is 53,952.

Comparatively, 1,228,800 is significantly higher than 53,952. Fewer calculations enable efficient data processing. Standard convolution alters the image 256 times, demanding 4,800 multiplications each. In contrast, separable convolution modifies the image once and extends it to 256 channels, conserving resources. Separable convolution's limitation is minimizing parameters, potentially affecting small networks. Nonetheless, it boosts efficiency without compromising effectiveness, making it a popular choice20.

## B. Introduction to Convolution Kernels

Convolution utilizes a matrix-like "kernel" to extract targeted "features" from an input image, enhancing the output through multiplication, as shown in Fig. 9. For instance, a kernel can sharpen the input image, yielding a desired output representation [20].

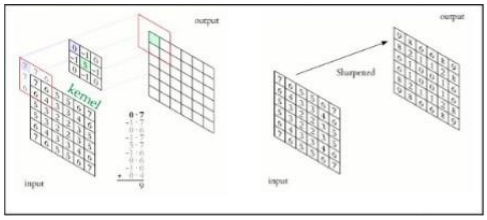


Fig. 9. Example for convolutional use kernel.

Convolution involves using a "kernel," a matrix of weighted values, to extract important features from input data. The term "convolution" is related to the dimensions of the kernel, such as in 2D convolutions where the kernel matrix is two dimensional. On the other hand, a "filter" is a collection of multiple kernels, each applied to a specific input channel. Filters have an extra dimension compared to kernels, particularly in 2D convolutions, where filters become 3D matrices. For a CNN layer with kernel dimensions of h*w and input channels of k, filter dimensions become k*h*w. Convolutions come in three types based on kernel nature: 1D, 2D and 3D convolutions.

*1) 1D convolution*: 1D convolutions are highly useful for processing time series data, especially with one-dimensional inputs, even when containing multiple channels [15]. These convolutions operate in a single direction, resulting in one-dimensional output, as demonstrated in Fig. 10.

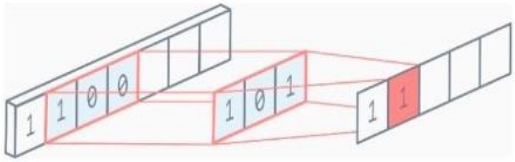


Fig. 10. 1D convolution.

*2) 2D Convolution*: In 2D convolutions, as shown in Fig. 11, the kernel size is 3x3. Multiple kernels, highlighted in yellow, form a filter that corresponds to various input channels indicated in blue. Each channel aligns with a distinct kernel. The filter moves in two directions across the input, resulting in a two-dimensional output. 2D convolutions are widely used, particularly in computer vision.

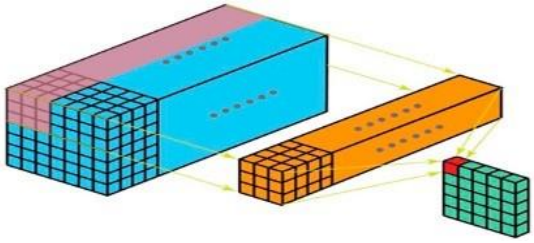

Fig. 11. 2D convolution.

*3) 3D Convolution*: Visualizing a 3D filter (4D matrix) can be complex. However, we'll focus on single-channel 3D convolution for simplicity. As shown in Fig. 12, the kernel moves in three directions, leading to a 3D output [16]. It's notable that most customization and modification research for CNN layers has primarily concentrated on 2D convolutions.

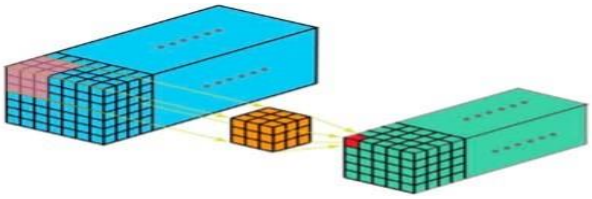

Fig. 12. 3D convolution.

## III. DBGC: Dimension-based Generic Convolution

Standard convolutions simultaneously handle spatial and channel data but can be computationally demanding. To improve efficiency, separable convolutions split spatial and channel data via depth-wise and point-wise convolutions, yet this can lead to computational bottlenecks in point-wise convolutions. The DBGC unit addresses this by using a dimension selector to efficiently encode spatial and dimension information, reducing computational load. Fig. 13 shows DBGC architecture. This involves two stages illustrated in Fig. 9: dimension-based convolution Section III(A) and dimension-wise fusion Section III(C). By replacing point-wise convolutions with dimension-based ones, the DBGC unit alleviates computational bottlenecks.

### A. Convolution based on Dimension (ConvDim)

The ConvDim block processes information distinctly along the height-wise, depth-wise, and width-wise dimensions. To achieve this, it extends the concept of depth-wise separable convolutions to encompass all dimensions of the input tensor I, which is characterized by dimensions H × D × W (representing height, depth, and width). As illustrated in Figure 12, ConvDim employs three sets of kernels for each dimension: KH for height-wise convolution, KD for depth wise convolution, and KW for width-wise convolution. These dimension-specific kernels generate outputs denoted as YH, YD, and YW, each having dimensions H × D × W, effectively capturing information from the input tensor. Subsequently, these outputs are amalgamated within the dimension selector block, merging spatial planes to yield the ultimate output, YDim.

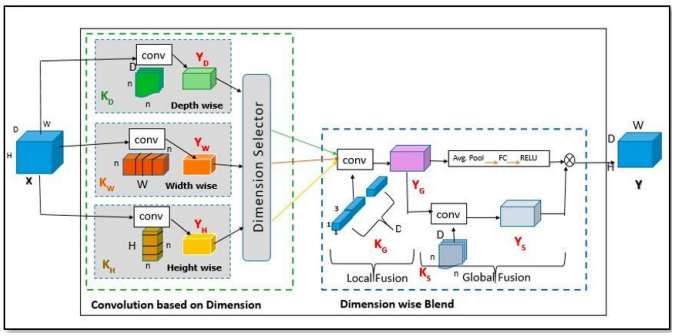


Fig. 13. DBGC architecture.

### B. Dimension Selector (Ds)

The dimension selector block, labeled as Ds, allows dimension selection tailored to specific application needs. Depending on requirements, users can opt to use only height, width, or depth dimensions during training. Parameters in this block specify chosen dimensions: Ds = KD, KW, KH. Combinations of two kernels are also possible, such as Ds = KD, KW, Ds = KH, KW, Ds = KD, KH, and Ds = KD, KW, KH. Thus, the dimension selector presents seven possibilities: height-only, width-only, depth-only, height and width, width and depth, height and depth, and all three dimensions. Selected kernels influence YDim integration, showcased in various dimension combinations in Fig. 14.

### C. Dimension-Wise Blends (DimBlend)

Dimension-wise convolutions capture local information from input tensor dimensions, but not global information. While point-wise convolutions in CNNs commonly combine global insights, they can be time-consuming. To address this, the dimension-wise blend module (DimBlend) divides pointwise convolution into local and global fusion phases, effectively merging dimension-wise representations from YDim into output Y [6]. The dimension-wise blend module, DimBlend, offers an alternative by splitting the point-wise convolution into two phases, as shown in Fig. 14.

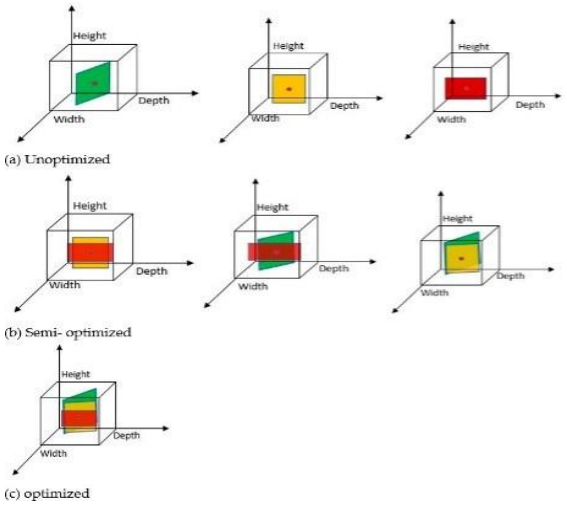


Fig. 14. ConvDim implementation progresses from individual kernel application (a) to simultaneous application of two kernels (b) and finally to concurrent application of all kernels (c), efficiently aggregating information through height-wise, depth-wise, and width-wise convolution.

The DBGC unit efficiently encodes spatial and dimension wise information using a multi-stage approach. The dimension selector (Ds) module allows flexible dimension choice, and the dimension-wise blend module (DimBlend) refines the process. In local fusion, YDim consolidates outputs from Ds, while DimBlend employs a group point-wise convolution layer (Kg) for dimension-wise merging. Global fusion follows, with DimBlend learning channel-wise and spatial representations, compressing spatial dimensions into channel-wise via fully connected layers. Integrating local and global fusion, DimBlend produces the final output Y, effectively capturing both spatial and dimension-wise information.

## IV. Result Analysis

### A. Implementation of DBGC

The integration of the DBGC unit within a CNN architecture involves the utilization of conventional CNN layers. The architecture's depiction is presented in Fig. 15, providing an overview of the overall arrangement.

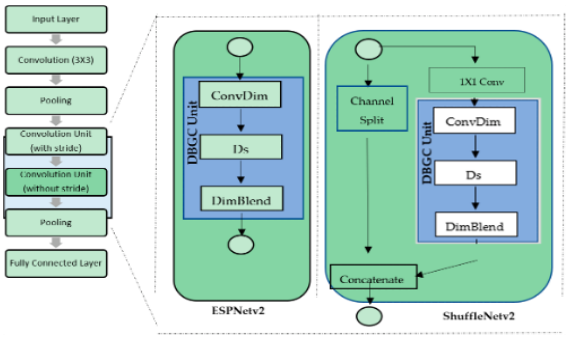


Fig. 15. The DBGC unit in different architecture (ESPNetv2 and ShuffleNetv2) designs.

### B. Experimental Setup

This section provides implementation details of the DBGC block and analyzes results. The DBGC unit was tested on ESPNetv2 and ShuffleNetv2 architectures, evaluating its impact on computational load and accuracy using the PASCAL VOC dataset. The integration of the DBGC unit with different architectures is discussed in Section V.

### C. Dataset Details

To demonstrate the DBGC unit's efficacy across diverse models, a standardized dataset was selected to ensure consistent comparison among architectures. The PASCAL VOC (Visual Object Classes Challenge) dataset was employed for this purpose. This dataset includes 20 object categories such as vehicles, animals, and furniture, with pixel-level segmentation annotations and bounding boxes. It is divided into training, validation, and private testing subsets, serving as a benchmark for object detection, semantic segmentation, and classification tasks.

### D. Results Analysis

This section presents the outcomes of integrating the DBGC unit into ESPNetv2 and ShuffleNetv2 architectures, focusing on the impact of reduced FLOPs on object detection and semantic segmentation accuracy. ESPNetv2 was used for object detection, while ShuffleNetv2 was employed for semantic segmentation. The distinction between FLOPs and FLOPS is explained: FLOPs represents the computational complexity of a model, quantifying the floating-point operations during inference [17]. On the other hand, FLOPS measures hardware processing speed. Faster hardware with higher FLOPS leads to quicker inference times. The equations to calculate FLOPs in the model are detailed in Fig. 16.

Within this section, the analysis of the results is structured into three distinct categories: (1) unoptimized kernel dimensions, (2) semi-optimized kernel dimensions, and (3) optimized kernel dimensions.

| Sr. No | Layer | The Equation to Calculate FLOPs |
|---|---|---|
| 1 | Convolution Layer | 2 × No. of Kernel × Kernel's Shape × Output Shape × Repeat Count (if available) |
| 2 | Pooling Layer (Without stride) | Height × Width × Depth of an input Image |
| 3 | Pooling Layer (With stride) | (Height/Stride) × Depth × (Width/Stride) of an input Image |
| 4 | Fully Connected Layer (FC Layer) | 2 × Input Size × Output Size |

Fig. 16. Output shape of a convolutional layer: Output = (Input - Kernel) + 1.

*1) Unoptimized kernel dimensions*: In the context of unoptimized kernel dimensions, employing a single kernel dimension for the output channel can lead to decreased object detection accuracy. This evaluation used ESPNetv2 and ShuffleNetv2 architectures for object detection and semantic segmentation tasks with the PASCAL VOC dataset. Initial models were created following existing guidelines, and then the same models were integrated with the proposed DBGC block using the width parameter in the dimension selector module from the DBGC architecture. The objective was to observe FLOPs variations and assess Top1 and Top5 accuracies. Results are presented in Tables I, II, and III for width-based, height-based, and depth-based kernels, respectively.

TABLE I. Exclusively Width-based Kernel

| Model | Dataset | Image Size | FLOP (In Millions) | Top1 | Top5 |
|---|---|---|---|---|---|
| ESPNet v2 | PASCAL | 224 × 224 | 86 | 66.1 | 70.02 |
| ShuffleNetv2 | | | 71 | 63.9 | 62.30 |
| ESPNetv2(DBGC-KH) | | | 24 | 35.64 | 43.86 |
| ShuffleNetv2 (DBGC-KH) | | | 21 | 34.5 | 39.54 |

TABLE II. Solely Height-based Kernel

| Model | Dataset | Image Size | FLOP (In Millions) | Top1 | Top5 |
|---|---|---|---|---|---|
| ESPNet v2 | PASCAL | 224 × 224 | 86 | 66.1 | 70.02 |
| ShuffleNetv2 | | | 71 | 63.9 | 62.30 |
| ESPNetv2(DBGC-KH) | | | 24 | 33.4 | 37.66 |
| ShuffleNetv2 (DBGC-KH) | | | 21 | 32.15 | 36.84 |

TABLE III. Only Depth-based Kernel

| Model | Dataset | Image Size | FLOP (In Millions) | Top1 | Top5 |
|---|---|---|---|---|---|
| ESPNet v2 | PASCAL | 224 × 224 | 86 | 66.1 | 70.02 |
| ShuffleNetv2 | | | 71 | 63.9 | 62.30 |
| ESPNetv2(DBGC-KD) | | | 24 | 33.34 | 36.66 |
| ShuffleNetv2 (DBGC-KD) | | | 21 | 31.95 | 35.84 |

The results show that unoptimized kernel dimensions reduce FLOPs by about one third compared to the original ESPNetv2 and ShuffleNetv2 architectures. However, using only a single dimension for kernel selection notably decreases accuracy by around 30%, as demonstrated in Fig. 17 to Fig. 19, highlighting the trade-off between computational efficiency and model performance.

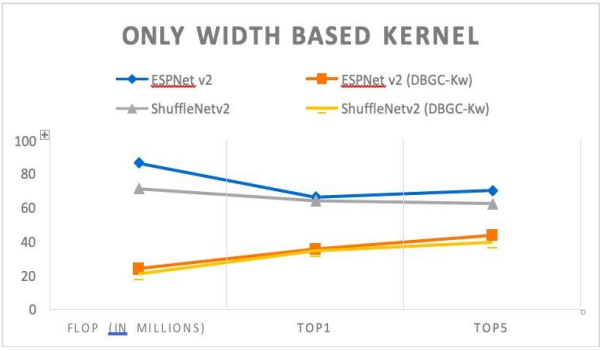


Fig. 17. Only height-based kernel.

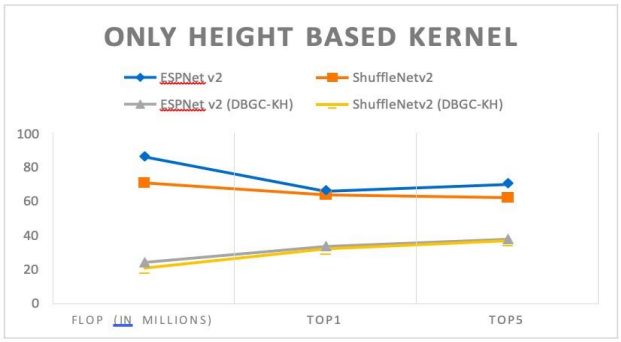


Fig. 18. Only height-based kernel.

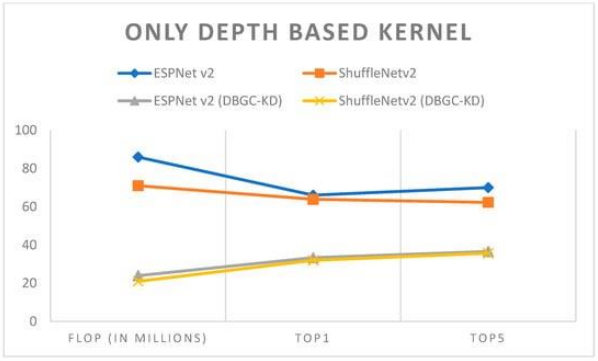


Fig. 19. Only depth-based kernel.

*2) Semi-optimized kernel dimensions*: In the semi-optimized kernel dimensions analysis, two kernel dimension combinations were employed for the output channel, resulting in reduced FLOPs with minimal accuracy loss. ESPNetv2 and ShuffleNetv2 were utilized for object detection and semantic segmentation on the PASCAL VOC dataset. The DBGC block was integrated with various kernel combinations (DBGC-Kwh, DBGC-Kdh, DBGC-Kwd) in dimension selector modules. Results in Tables IV to VI demonstrate lowered FLOPs and slight accuracy effects.

TABLE IV. DEPTH & WIDTH-BASED KERNEL

| Model | Dataset | Image Size | FLOP (In Millions) | Top1 | Top5 |
|---|---|---|---|---|---|
| ESPNet v2 | PASCAL | 224 x 224 | 86 | 70.02 | 66.1 |
| Shuffle Netv2 | | | 71 | 62.30 | 63.9 |
| ESPNetv2(DBGC-KDW) | | | 48 | 71.59 | 66.41 |
| Shuffle Netv2 (DBGC-KDW) | | | 42 | 64.83 | 69.75 |

TABLE V. DEPTH & HEIGHT-BASED KERNEL

| Model | Dataset | Image Size | FLOP (In Millions) | Top1 | Top5 |
|---|---|---|---|---|---|
| ESPNet v2 | PASCAL | 224 x 224 | 86 | 70.02 | 66.1 |
| Shuffle Netv2 | | | 71 | 62.30 | 63.9 |
| ESPNetv2(DBGC-KDH) | | | 48 | 65.57 | 68.42 |
| Shuffle Netv2 (DBGC-KDH) | | | 42 | 65.82 | 67.73 |

TABLE VI. HEIGHT & WIDTH-BASED KERNEL

| Model | Dataset | Image Size | FLOP (In Millions) | Top1 | Top5 |
|---|---|---|---|---|---|
| ESPNet v2 | PASCAL | 224 x 224 | 86 | 70.02 | 66.1 |
| Shuffle Netv2 | | | 71 | 62.30 | 63.9 |
| ESPNetv2(DBGC-KHW) | | | 48 | 65.52 | 71.47 |
| Shuffle Netv2 (DBGC-KHW) | | | 42 | 64.87 | 68.77 |

Semi-optimized kernel dimensions in DBGC halve FLOPs while maintaining good accuracy; object detection sustains accuracy and semantic segmentation gains 1-2%, as shown in Fig. 20 to Fig. 22.

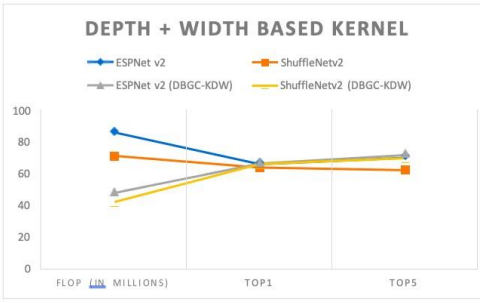


Fig. 20. Combining depth-based and width-based kernels.

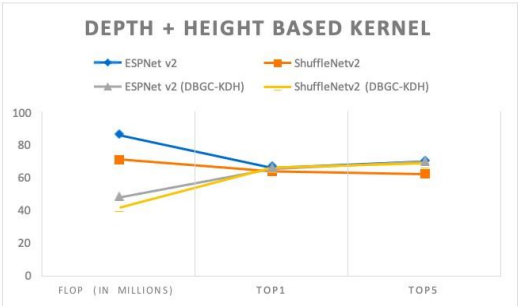


Fig. 21. Combining depth-based and height-based kernel.

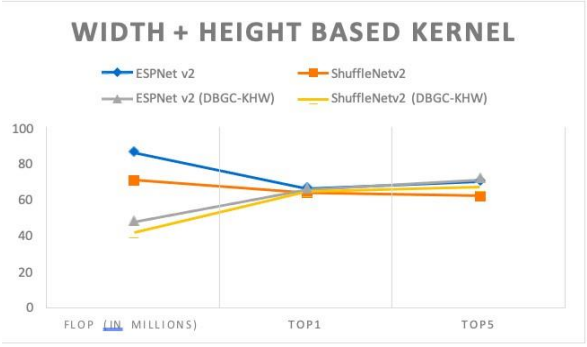


Fig. 22. Combining height-based and width-based kernel.

Fig. 23 illustrates the analysis of results, showing that using two dimensions reduces FLOPs while maintaining accuracy close to the original architecture.

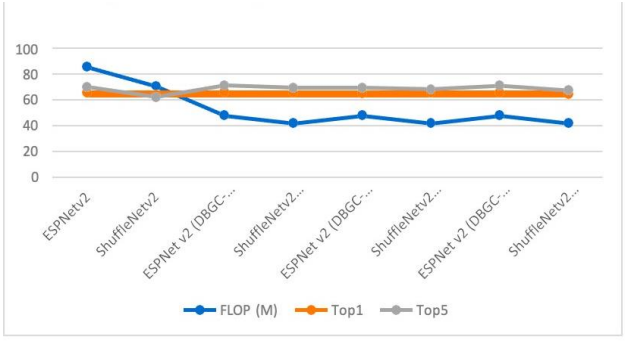


Fig. 23. Semi-optimized kernel dimension.

### E. Optimized Kernel Dimension

To improve accuracy while considering computational efficiency, a combination of all three kernel dimensions (width, height, and depth) was employed for the output channel. This approach increased FLOPs but enhanced accuracy. After implementing ESPNetv2 and ShuffleNetv2 models following the methodology in [7], these models were further enhanced with the incorporation of the proposed DBGC-Khwd block. The DBGC block's parameters for width, height, and depth, as described in Section 3 of the DBGC architecture, were chosen. The goal was to compare FLOPs and evaluate Top1 and Top5 accuracies. The outcomes are presented in Table VII and Fig. 24 for kernels based on depth, width, and height.

TABLE VII. HEIGHT & WIDTH & DEPTH-BASED KERNEL

| Model | Dataset | Image Size | FLOP (In Millions) | Top1 | Top5 |
|---|---|---|---|---|---|
| ESPNet v2 | PASCAL | 224 x 224 | 86 | 70.02 | 66.1 |
| Shuffle Netv2 | | | 71 | 62.30 | 63.9 |
| ESPNetv2(DBGC-KHW) | | | 48 | 70.52 | 74.48 |
| Shuffle Netv2 (DBGC-KHW) | | | 42 | 69.97 | 74.77 |

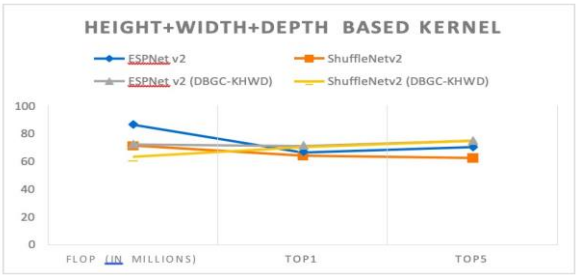


Fig. 24. Optimized kernel dimension.

Fig. 25 highlights that selecting all three dimensions led to a 4 to 5% accuracy increase while reducing FLOPs. showcasing the superiority of optimized kernels. For a comprehensive comparison, Fig. 25 and Fig. 26 display the performance of unoptimized, semi-optimized, and optimized kernel dimensions in ESPNetv2 and ShuffleNetv2 architectures, respectively.

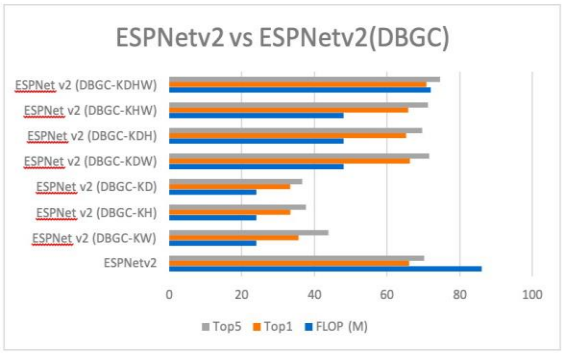


Fig. 25. ESPNetv2 versus ESPNetv2 (DBGC).

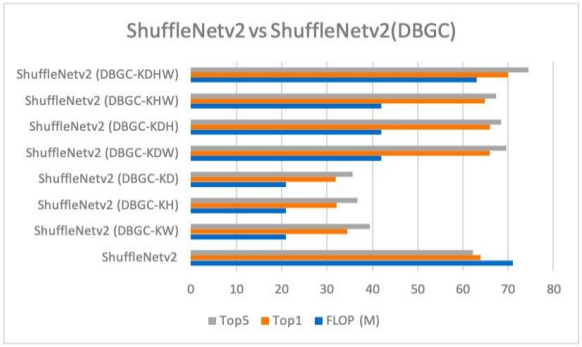


Fig. 26. ShuffleNetv2 versus ShuffleNetv2 (DBGC).

Fig. 27 presents a box plot summarizing the various methods employed on the PASCAL VOC dataset. "Ev2" represents ESPNetv2, and "Sv2" stands for ShuffleNetv2 in the chart.

A box plot visualizes the performance of unoptimized, semi-optimized, and optimized kernels for ESPNetv2 versus ESPNetV2 (DBGC) and ShuffleNetv2 versus ShuffleNetV2 (DBGC). The effectiveness of DBGC is evaluated using the MS COCO dataset [19], showcasing the performance differences in Fig. 28.

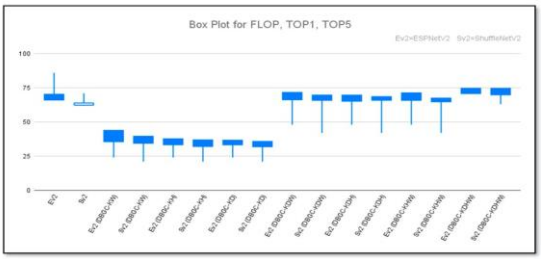


Fig. 27. Box Plot for kernel types.

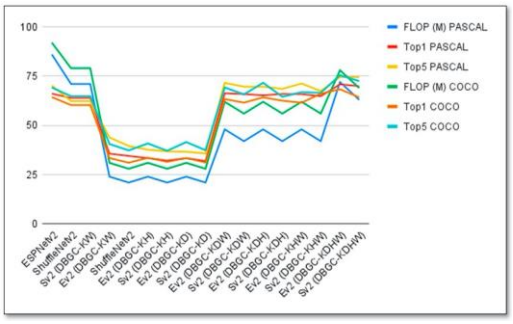


Fig. 28. Compare the performance of ESPNetv2 and ShuffleNetv2 with their DBGC-enhanced versions on PASCAL and COCO datasets.

## V. Conclusion

In conclusion, the proposed DBGC unit demonstrates its versatility by being applicable to various CNN-based network models. By integrating DBGC into ESPNetv2 and ShuffleNetv2 architectures, extensive evaluations based on FLOPs, Top1, and Top5 accuracies were conducted using the PASCAL VOC dataset. The findings indicate that unoptimized kernel based DBGC substantially reduces FLOPs by about one third, leading to significantly improved speed. However, this reduction in FLOPs comes at the cost of a notable decrease in accuracy. On the other hand, semi-optimized dimension-based kernels offer a balance between reduced FLOPs (around half) and maintained or slightly improved accuracy in ShuffleNetV2 with DBGC. Lastly, optimized dimension-based kernels achieve the highest accuracy while still reducing FLOPs by approximately five million. These results emphasize the potential of DBGC for enhancing the efficiency and accuracy of various CNN architectures.

## Future Work

For future research, the proposed architecture could be extended to evaluate its performance across different datasets, enabling a broader understanding of its capabilities. Additionally, investigating whether unoptimized dimension-based kernels can yield improved accuracy when applied to single dimensional data could be a valuable exploration. Further enhancements could involve automating the dimension selection process in the dimension selector modules to dynamically choose dimensions based on the characteristics of the datasets provided, potentially optimizing the efficiency and effectiveness of the DBGC approach.

## References

[1] Bhatt, D.; Patel, C.; Talsania, H.; Patel, J.; Vaghela, R.; Pandya, S.; Modi, K.; Ghayvat, H. CNN Variants for Computer Vision: History,

[2] Architecture, Application, Challenges and Future Scope. Electronics 2021, 10, 2470. [CrossRef]

[3] Bhatt, D.; Bhensadadiya, N.P. Survey On Various Intelligent Traffic Management Schemes For Emergency Vehicles. Int. J. Recent Innov. 2013, 1, 11–16.

[4] Garg, S.; Patel, C.; Tank, H.; Ukani, V. Efficient Vehicle Detection and Classification for Traffic Surveillance System. In Proceedings of the International Conference on Advances in Computing and Data Sciences, Ghaziabad, India, 11–12 November 2016; pp. 495–503.

[5] Garg, S.; Zaveri, T.; Banerjee, J.; Patel, R.; Patel, C.I. Human action recognition using fusion of features for unconstrained video sequences. Comput. Electric. Eng. 2018, 70, 284–301.

[6] Zhang, X.; Zheng, H.-T.; Sun, J.; Ma, N. ShuffleNet V2: Practical Guidelines for Efficient CNN Architecture Design. In Proceedings of the Computer Vision and Pattern Recognition, Salt Lake City, UT, USA, 18–22 June 2018; Available online: https://arxiv.org/abs/1807.11164v1 (accessed on 20 January 2022).

[7] Rastegari, M.; Shapiro, L.; Hajishirzi, H.; Mehta, S. ESPNetv2: A LightWeight, Power Efficient, and General Purpose Convolutional Neural Network. arXiv 2019, arXiv:1811.11431.

[8] Mehta, S.; Hajishirzi, H.; Rastegari, M. DiCENet: Dimension-wise Convolutions for Efficient Networks. IEEE Trans. Pattern Anal. Mach. Intell. 2020. [CrossRef]

[9] Howard, A.; Zhu, M.; Zhmoginov, A.; Chen, L.-C.; Sandler, M. MobileNetV2: Inverted Residuals and Linear Bottlenecks. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR), Salt Lake City, UT, USA, 18–23 June 2018; pp. 4510–4520.

[10] Feng, C.; Zhuo, S.; Zhang, X.; Shen, L.; Aleksic, M.; Sheng, T. A Quantization-Friendly Separable Convolution for MobileNets. In Proceedings of the 1stWorkshop on Energy Efficient Machine Learning and Cognitive Computing for Embedded Applications (EMC2),Williamsburg, VA, USA, 25 March 2018; pp. 14–18.

[11] Zhu, F.; Liu, J.; Liu, G.; Zhang, R. Depth-Wise Separable Convolutions and Multi-Level Pooling for an Efficient Spatial CNN-Based Steganalysis. IEEE Trans. Inf. Forensics Secur. 2020, 15, 1138–1150.

[12] Yin, Z.; Wu, M.; Wu, Z.; Kamal, K.C. Depthwise separable convolution architectures for plant disease classification. Comput.Electron. Agric. 2019, 165, 104948.

[13] Choi, Y.; Choi, H.; Yoo, B. Fast Depthwise Separable Convolution for Embedded Systems. In Proceedings of the International Conference on Neural Information Processing (ICONIP), Siem Reap, Cambodia, 13–16 December 2018.

[14] Kaiser, L.; Gomez, A.N.; Chollet, F. Depthwise Separable Convolutions for Neural Machine Translation. arXiv 2017, arXiv:1706.03059.

[15] Tran, M.-K.; Yeung, S.-K.; Hua, B.-S. Pointwise Convolutional Neural Networks. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR), Salt Lake City, UT, USA, 18–23 June 2018; pp. 984–993.

[16] Bracewell, R. Two-Dimensional Convolution. In Fourier Analysis and Imaging; Springer: Boston, MA, USA, 2003.

[17] Wang, H.; Zhang, Q.; Yoon, S.W.; Won, D.; Lu, H. A 3D Convolutional Neural Network for Volumetric Image Semantic Segmentation. Procedia Manuf. 2019, 39, 422–428.

[18] Gosling, J.B. Floating Point Operation. In Design of Arithmetic Units for Digital Computers; Springer: New York, NY, USA, 1980.

[19] Zhu, M.; Chen, B.; Kalenichenko, D.;Wang,W.; Howard, A.G. Mobilenets:Efficient convolutional neural networks for mobile vision applications. arXiv 2017, arXiv:1704.04861.

[20] MS COCO Dataset. Available online: https://cocodataset.org/#download (accessed on 20 January 2022).

[21] Bosamiya, D.; Kamariya, N.; Miyatra, A. A Survey on Disease and Nutrient Deficiency Detection in Cotton Plant. Int. J. Recent Innov. Trends Comput. Commun. 2013, 1, 812–815.