HetConv: Heterogeneous Kernel-Based Convolutions for Deep CNNs

Pravendra Singh Vinay Kumar Verma Piyush Rai Vinay P. Namboodiri
Department of Computer Science and Engineering, IIT Kanpur, India
{psingh, vkverma, piyush, vinaypn}@cse.iitk.ac.in

Abstract

We present a novel deep learning architecture in which the convolution operation leverages heterogeneous kernels. The proposed HetConv (Heterogeneous Kernel-Based Convolution) reduces the computation (FLOPs) and the number of parameters as compared to standard convolution operation while still maintaining representational efficiency. To show the effectiveness of our proposed convolution, we present extensive experimental results on the standard convolutional neural network (CNN) architectures such as VGG [30] and ResNet [8]. We find that after replacing the standard convolutional filters in these architectures with our proposed HetConv filters, we achieve 3X to 8X FLOPs based improvement in speed while still maintaining (and sometimes improving) the accuracy. We also compare our proposed convolutions with group/depth wise convolutions and show that it achieves more FLOPs reduction with significantly higher accuracy.

1. Introduction

Convolutional neural networks [8, 19, 30] have shown remarkable performance in domains like Vision and NLP. The general trend to improve performance further has made models more complex and deeper. Increasing the accuracy by increasing model complexity with a deeper network is not for free; it comes with the cost of a tremendous increase in computation (FLOPs). Therefore, various types of convolution operations/convolutional filters have been proposed to reduce FLOPs to the model more efficient.

Existing convolutional filters can be roughly divided into three categories: 1- Depthwise Convolutional Filter to perform depthwise convolution (DWC) [38], 2- Pointwise Convolutional Filter to perform pointwise convolution (PWC) [36] and 3- Groupwise Convolutional Filter to perform groupwise convolution (GWC) [19]. Most of the recent architectures [12, 35, 2, 15, 37, 42] use a combination of these convolutional filters to make the model efficient. Using these convolutions (e.g., DWC, PWC, and GWC), many of the popular models [15, 12, 2] have explored new architectures to reduce FLOPs. However, designing a new architecture requires a lot of work to find out the best combination of filters that result in minimal FLOPs.

Another popular approach to increase the efficiency of a model is to use model compression [27, 1, 24, 11, 21, 10, 41]. Model compression can be broadly categorized into three categories: connection pruning [6], filter pruning [24, 11, 21, 10, 32, 31, 33] and quantization [6, 27].

In filter pruning, the idea is to prune a filter that has the minimal contribution in the model, and after removing this filter/connection, the model is usually finetuned to maintain its performance. While pruning the model, we require a pre-trained model (possibly requiring a computationally expensive training as a preprocessing step), and then later we discard the filter that has a minimal contribution. Hence it is a very costly and tricky process. Therefore, using an efficient convolutional filter or convolution operation to design an efficient architecture is a more popular approach than pruning. This does not require expensive training and then pruning since training is done from scratch efficiently.

Using efficient convolutional filters, there are two different objectives. One kind of work focuses on designing architectures that have minimal FLOPs while compromising on accuracy. These works focus on developing the model for the IoT/low-end device [12, 42]. These models suffer from the low accuracy hence they have to search the best possible model to create a balance between accuracy and FLOPs. So there is a tradeoff between FLOPs and model accuracy.

Another set of work focuses on increasing accuracy while keeping the model FLOPs the same as the original architecture. The recent architectures, such as Inception [35], RexNetXt [40] and Xception [2] are examples of this kind of work. Their objective is to design a more complex model using efficient convolutional filters while keeping the FLOPs the same as the base model. It is usually expected that a more complex model would learn better features, resulting in better accuracies. However, these methods are not focused on designing a new architecture, but primarily on using existing efficient filters in standard base architectures. Therefore these works keep the number of layers and

the architecture the same as the base model and increase the filters on each layer such that it does not increase the FLOPs.

In contrast to these two approaches, the primary focus of our work is to reduce the FLOPs of the given model/architecture by designing new kernels, without compromising on the loss of accuracy. Experimentally we find that the proposed approach has much lower FLOPs than the state-of-art pruning approaches while maintaining the accuracy of the base model/architecture. The pruning approaches are very costly and show a significant drop in accuracy to achieve FLOPs compression.

In the proposed approach, we are choosing a different strategy to increase the efficiency of the existing model without sacrificing the accuracy. An architecture search requires years of research to get an optimized architecture. Therefore, instead of designing a new efficient architecture, we design an efficient convolution operation (convolutional filter) that can be directly plugged into any existing standard architecture to reduce FLOPs. To achieve this, we propose a new type of convolution - *heterogeneous* convolution.

The convolution operation can be divided into two categories based on the types of the kernel:

- Homogeneous convolution using a traditional convolutional filter (for example standard convolution, groupwise convolution, depthwise convolution, pointwise convolution). Homogeneous convolution can be performed using a homogeneous filter. A filter is said to be homogeneous if it contains all kernels of the same size (for example, in a $3 \times 3 \times 256$ CONV2D filter, all 256 kernels will be of size 3×3).

- Heterogeneous convolution uses a heterogeneous convolutional filter (HetConv). A filter is said to be heterogeneous if it contains different sizes of kernels (for example, in a HetConv filter, out of 256 kernels some kernels are of size 3×3 and remaining kernels are of size 1×1).

Using a heterogeneous filter in deep CNN overcomes the limitation of the existing approaches that are based on efficient architecture search and model compression. One of the latest efficient architecture MobileNet [12] uses depthwise and pointwise convolution. The standard convolutional layer is replaced by two convolutional layers hence it has more latency (latency one). Please refer to Section-3.3 and Figure-4 for more details about latency. But our proposed HetConv has same latency as the original architecture (latency zero) unlike [12, 35, 36, 2] that have latency greater than zero.

Compared to model compression that suffers from high accuracy drop, our approach is very competitive to the state-of-art result of the standard model like ResNet [8] and VGGNet [30]. Using HetConv filters, we can train our model from scratch, unlike pruning approaches that need a pre-trained model, without sacrificing accuracy. The pruning approaches also suffer from sharp accuracy drop if we increase the degree of FLOP pruning. Using proposed HetConv filters, we have state-of-art result regarding FLOPs compare to the FLOP pruning methods. Also, the pruning process is inefficient as it takes a lot of time in training and fine tuning after pruning. Our approach is highly efficient and gives a similar result compared to the original model while training from scratch.

To the best of our knowledge, this is the first convolution/filter that is heterogeneous. This heterogeneous design helps to increase the efficiency (FLOPs reduction) of the existing architecture without sacrificing the accuracy. We did extensive experiment on different architectures like ResNet [8], VGG-16 [30] etc just by replacing their original filters to our proposed filters. We found that without sacrificing the accuracy of these models, we have a high degree of FLOPs reduction (3X to 8X). These FLOPs reductions are even significantly better as compared to existing pruning approach.

Our main contributions are as follows:

- We design an efficient heterogeneous convolutional filter, that can be plugged into any existing architecture to increase the efficiency (FLOPs reduction of order 3X to 8X) of the architecture without sacrificing the accuracy.

- The proposed HetConv filters are designed in such a way that it has zero latency. Therefore, there is negligible delay from input to output.

2. Related Work

The recent success of deep neural network [19, 28, 8, 14, 4, 5, 26, 39, 34] depends on the model design. To achieve a minimal error rate, the model becomes more and more complex. The complex and deeper architecture contain millions of parameters and requires billions of FLOPs (computations) [8, 30, 14]. These models require machines with high-end specifications, and these type of architecture are very inefficient on low computing resources. This raises interest in designing efficient models [7]. The work to increase the efficiency of the model can be divided into two parts.

2.1. Efficient Convolutional Filter

To design the efficient architecture recently few novel convolutional filters have been proposed. Among them Groupwise Convolution (GWC) [19], Depthwise convolution (DWC) [38] and Pointwise Convolution (PWC) [8] are the popular convolutional filters. These are widely used to design efficient architecture. GoogleNet [36] use the inception module and irregular stacking architecture. Inception

module uses GWC and PWC to reduce FLOPs. ResNet [8, 9] uses a bottleneck structure to design an efficient architecture with residual connection. They use PWC and the standard convolution that help to go deeper without increasing the model parameter and reduces the FLOPs explosion. Therefore they can design a much deeper architecture compare to VGG [30]. ResNetxt [40] use the ResNet architecture and they divide each layer with GWC and PWC. Therefore without increasing FLOPs, they can increase the cardinality [1]. They show that increasing cardinality is much more effective than a deeper or wider network. SENet [13] design a new connection that gives the weight to each output feature map with a minor increase in FLOPs but shows a boost in the performance.

MobileNet [12] is another popular architecture specially designed for the IoT devices contains DWC and PWC. This architecture is very light and highly efficient in term of FLOPs. This reduction in FLOPs are not for free and come with the cost of a drop in the accuracy compared to the state-of-art models. [17, 16] use different types of convolutional filers at the same layers, but each filter performs a *homogeneous* convolution due to the presence of same types of kernels in each filter. Using different types of convolutional filers at the same layers also helps in reducing parameters/FLOPs. In our proposed convolution, the convolution operation is heterogeneous due to the presence of *different types of kernels in each filter*.

2.2. Model Compression

Another popular approach to increase the efficiency of CNN is model compression. These can be categorised as: 1- Connection Pruning [6, 43], 2- Filter Pruning [23, 3, 32, 31, 33] and 3- Bit Compression [27]. Filter pruning approaches are more effective as compared to other approaches and give high compression rate in terms of FLOPs. Also, the filter pruning approaches do not need any special hardware/software support (sparse library).

Most of the works in filter pruning calculates the importance of the filter and prunes them based on some criteria followed by re-training to recover the accuracy drop. [20] used l_1 norm as a metric for ranking filters. But the pruning is done on the pre-trained model and involves iterative training and the pruning which is costly. Also, filter pruning shows a sharp accuracy drop in accuracy, if the degree of flop pruning increases [3, 41, 24].

3. Proposed Method

In this work, we propose a novel filter/convolution (HetConv) that contains a heterogeneous kernel (e.g., few kernels are of size 3×3, and others may be 1×1) to reduce the FLOPs of existing models with the same accuracy as the original model. This is very different from the standard convolutional filter that is made of homogeneous kernels (say all 3×3 or all 5×5). The heterogeneous filter is very efficient in terms of FLOPs. It can be approximated as a combined filter of a groupwise convolutional filter (GWC) and pointwise convolutional filter (PWC). To reduce the FLOPs of a convolutional layer, we generally replace it by two or more layers (GWC/DWC and PWC), but it increases the latency because next layer's input is the previous layer's output. Hence all computations have to be done sequentially to get the correct output. In contrast, our proposed HetConv has the same latency. Difference between the standard filter and HetConv filter is shown in the Figure- 1 and Figure- 2.

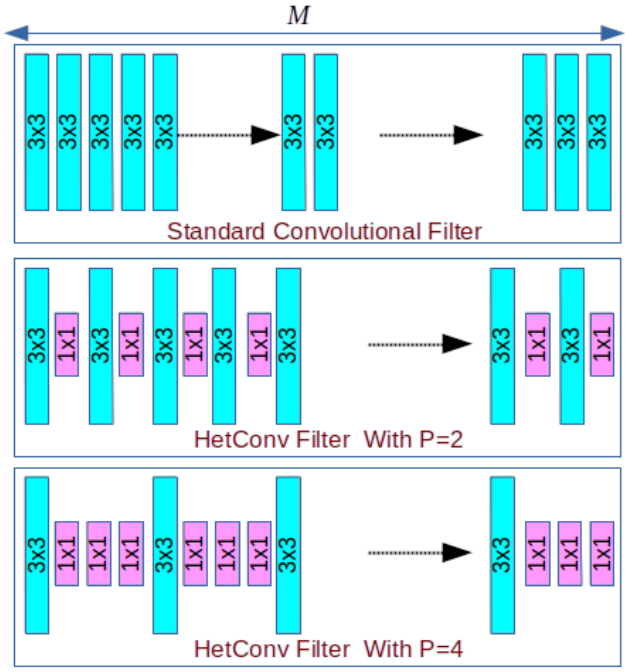

Figure 1. Difference between standard convolutional filter (homogeneous) and heterogeneous convolutional filter (HetConv). Here M is the input depth (number of input channels), and P is the part. Out of M kernels, M/P kernels will be of size 3×3 and remaining will be 1×1 kernels.

In the standard convolutional layer, let us assume input (input feature map) of size $D_i \times D_i \times M$. Here D_i is the input square feature map spatial width and height and M is the input depth (number of input channels). Also consider $D_o \times D_o \times N$ is the output feature map. Here D_o is the output square feature map spatial width and height and N is the output depth (number of output channels). An output feature map is obtained by applying the N filters of size $K \times K \times M$. Here K is the kernel size. Therefore the total computational cost at this layer L can be given as:

$$FL_S = D_o \times D_o \times M \times N \times K \times K \qquad (1)$$

[1] The size of the set of transformations

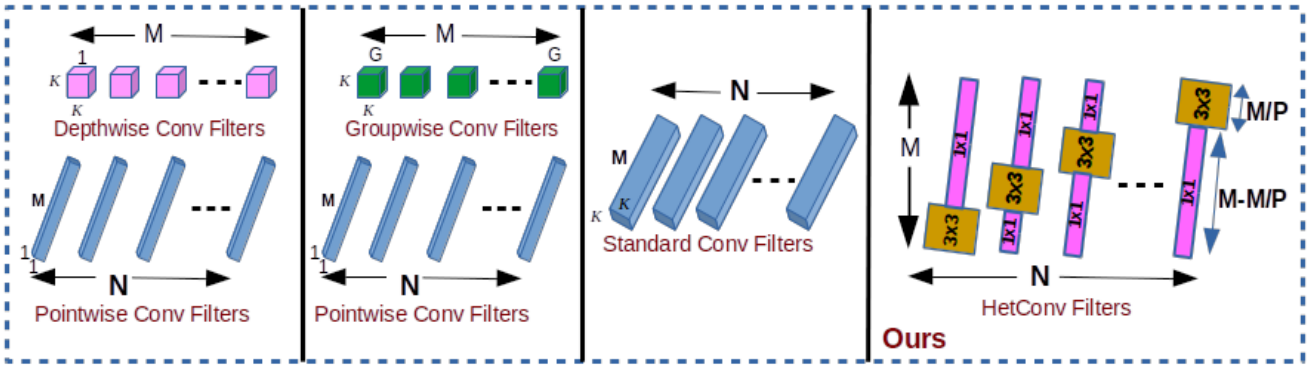

Figure 2. Comparison between the proposed Convolutional filters (HetConv) with other efficient convolutional filters. Our heterogeneous filters have zero latency while other (GWC+PWC or DWC+PWC) have a latency of one unit.

It is clear from the Equation-1 the computational cost depends on the kernel size (K), feature map size, input channels M and output channels N. This computational cost is very high which can be further reduced by carefully designing the new convolution operation. To reduce the high computation, various convolutions like DWC, PWC and GWC are proposed which is used in the many recent architecture [12, 42, 35] to reduce the FLOPs but all of them increase the latency.

The standard convolution operation and some recent convolution operations [12, 42, 35] use a homogeneous kernel (i.e., each kernel is of the same size for the whole filter). Here to increase the efficiency we are using the heterogeneous kernels. This contains different size kernels for the same filter. Please refer to Figure-3 to visualize all filters at a particular layer L. Let us define part P which controls the number of different types of kernels in a convolutional filter. For part P, a fraction $1/P$ out of total kernels will be for $K \times K$ kernels and remaining fraction $(1 - 1/P)$ will be for 1×1 kernels. For better understanding, Let's take an example, in a $3 \times 3 \times 256$ standard CONV2D filter if you replace $(1-1/P)*256$, 3×3 kernels with 1×1 (along with the central axis), you will get a HetConv filter with part P. Please refer to Figure-1 and 2.

The computational cost for $K \times K$ size kernels in the HetConv filters with part P on the layer L is given as:

$$FL_K = (D_o \times D_o \times M \times N \times K \times K)/P \qquad (2)$$

It reduces the cost P times since instead of M kernels of size $K \times K$, now we have only M/P kernels of size $K \times K$.

The remaining $(M - M/P)$ kernels are of size 1×1. The computational cost of the remaining 1×1 kernels can be given as:

$$FL_1 = (D_o \times D_o \times N) \times \left(M - \frac{M}{P}\right) \qquad (3)$$

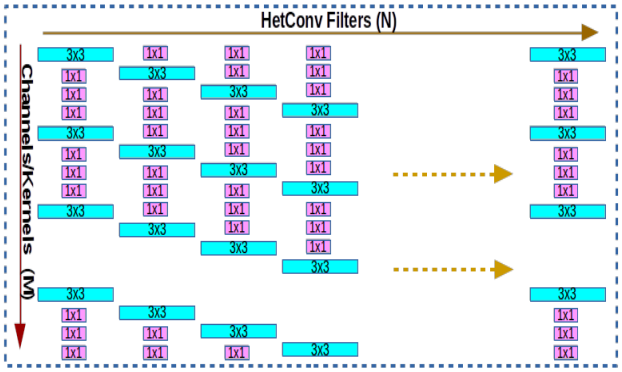

Figure 3. **Convolutional filters at a layer L:** Proposed Convolutional Filter (HetConv) using Heterogeneous Kernel. In this Figure, each channel is made of using the heterogeneous kernel of size 3×3 and 1×1. Replacing 3×3 kernels with 1×1 kernels in standard convolutional filter reduces the FLOPs dramatically while maintaining the accuracy. Filters of a particular layer are arranged in a shifted manner (i.e., if the first filter starts 3×3 kernel from the first position then the second filter starts the 3×3 kernel from the second position and so on).

Therefore the total computational cost at layer L is given as:

$$FL_{HC} = FL_K + FL_1 \qquad (4)$$

The total reduction (R) in the computation as compared to standard convolution can be given as:

$$\begin{aligned} R_{HetConv} &= \frac{FL_K + FL_1}{FL_S} \\ &= \frac{1}{P} + \frac{(1 - 1/P)}{K^2} \end{aligned} \qquad (5)$$

In the Equation-5 if we put $P = 1$ then it becomes standard convolutional filter.

By reducing the size of the filter on some channels from says 3×3 to 1×1, we are reducing the spatial extent of a

filter. However, by retaining the size to be 3×3 on some channel, we ensure that the filter does cover the spatial correlation on some channels and need not to have the same spatial correlation on all channels. We observe in the experimental section that by doing so, one can obtain similar accuracies as a homogeneous filter. On the other hand, if we avoid and retain a 1×1 filter size on all channels, then we would not have the necessary spatial correlation information covered, and the accuracy would suffer.

3.1. Comparision with DepthWise followed by PointWise Convolution

In extreme case when $P = M$ in HetConv, HetConv can be compared with DWC+PWC (DepthWise followed by PointWise Convolution). MobileNet [12] use this type of convolution. While MobileNet takes more FLOPs than our extreme case with the more delay since MobileNet [12] has latency one.

The total FLOPs for DWC+PWC (MobileNet) for layer L can be computed as:

$$FL_{MobNet} = D_o \times D_o \times M \times K \times K + M \times N \times D_o \times D_o \tag{6}$$

Therefore the total reduction in computation as compared to the standard convolution:

$$\begin{aligned} R_{MobNet} &= \frac{FL_{MobNet}}{FL_S} \\ &= \frac{1}{N} + \frac{1}{K^2} \end{aligned} \tag{7}$$

It is clear from the Equation-5 that we can change the part P value to trade off between the accuracy and FLOPs. If we decrease the P value, the resulting convolution will be closer to the standard convolution. To show the effectiveness of the proposed HetConv filter, we have shown results in the experimental section where HetConv achieves significantly better accuracy with similar FLOPs.

In the extreme case when $P = M$, from Eq.-5 and 7 (for MobileNet $N = M$),we can conclude:

$$\frac{1}{M} + \frac{(1 - 1/M)}{K^2} < \frac{1}{M} + \frac{1}{K^2} \tag{8}$$

Reduction = Total reduction in the computation as compared to standard convolution

$$Speedup = \frac{1}{Reduction} \tag{9}$$

Therefore from Eq.-8, it is clear that MobileNet takes more computation than our approach. In our HetConv, we have latency zero while MobileNet has latency one. In this extreme case, we have significantly better accuracy than MobileNet (refer to the experiment section).

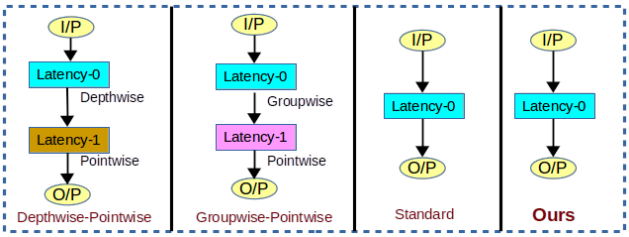

Figure 4. The figure shows the comparison with the different types of convolution in terms of latency.

3.2. Comparision with GroupWise followed by PointWise Convolution

In the case of groupwise convolution followed by pointwise convolution (GWC+PWC) with the group size G, the total FLOPs for GWC+PWC for layer L can be computed as:

$$FL_G = (D_o \times D_o \times M \times N \times K \times K)/G + M \times N \times D_o \times D_o \tag{10}$$

Therefore the total reduction in the computation as compare to the standard convolution:

$$\begin{aligned} R_{Group} &= \frac{FL_G}{FL_S} \\ &= \frac{1}{G} + \frac{1}{K^2} \end{aligned} \tag{11}$$

Similarly when $P = G$, from Eq.-5 and 11 we have:

$$\frac{1}{P} + \frac{(1 - 1/P)}{K^2} < \frac{1}{P} + \frac{1}{K^2} \tag{12}$$

Therefore from Eq.-12, it is clear that GWC+PWC takes more computation than our approach. In our HetConv, we have latency zero while GWC+PWC has latency one.

3.3. Running Latency

Most of the previous approaches [35, 36, 42, 12] designed efficient convolution to reduce the FLOPs, but they increase the latency[2] in the architecture. The latency in the different types of convolutions is shown in the Figure-4. In the Inception module [36, 35], one layer is broken down into two or more sequential layers. Therefore, the latency in the architecture is greater than zero. In the Xception [2] first GWC is applied, and on the output of GWC, PWC is applied. PWC waits for the completion of the GWC. Hence this approach reduces the FLOPs but increases latency in the system. Similarly in the MobileNet [12] first DWC and then PWC is applied therefore it has latency one. This latency includes a delay in parallel devices like GPU. In our

[2]One parallel step is converted to multiple sequential step hence reduction in parallelizability. Later stage of layers waits for the execution to be finished on the previous stage because all computations have to be done sequentially across layers

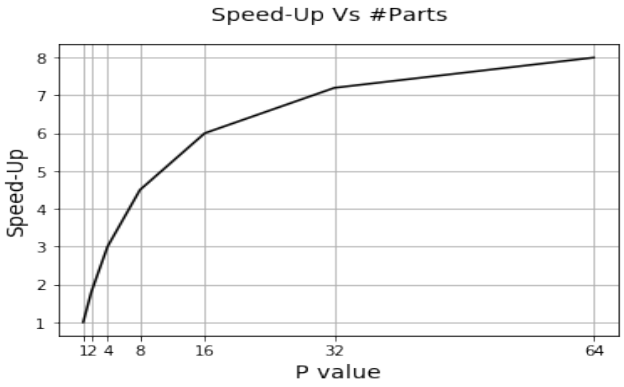

Figure 5. Speedup over standard convolution for different values of P for a HetConv Filter with 3×3 and 1×1 kernels.

proposed approach any layers are not replaced by sequential layers hence has the latency zero. We directly design the filter such that without increasing any latency we can reduce the FLOPs. Our proposed approach is very competitive in terms of FLOPs as compared to previous efficient convolutions while maintaining the latency zero.

3.4. Speedup over standard convolution for different values of P

As shown in Figure-5, Speedup increases with the P value. We can use P value to trade off between the accuracy and FLOPs. If we decrease the P value, the resulting convolution will be closer to the standard convolution. To show the effectiveness of the proposed HetConv filter, we have shown results in the experimental section where HetConv achieved significantly better accuracy with respect to the other types of convolution with similar FLOPs.

4. Experiments and Results

To show the effectiveness of the proposed HetConv filter we perform extensive experiments with the current state-of-art architectures. We replaced their standard convolutional filters from these architecture with the proposed one. We performed three large scale experiment on the ImageNet [29] with the ResNet-34, ResNet-50 [8] and VGG-16 [30] architectures. We have shown three small scale experiment on the CIFAR-10 [18] for the VGG-16, ResNet-56, and MobileNet [12] architectures. We set the value of reduction ratio to 8 for Squeeze-and-Excitation (SE) [13] in all our experiments.

4.1. Notations

XXX_Pα: XXX is the architecture, and part value is $P = \alpha$; XXX_Pα_SE: SE for Squeeze-and-Excitation with reduction-ratio = 8; XXX_GWCβ_PWC: GWCβ_PWC is the groupwise convolution with group size β followed by pointwise convolution; XXX_DWC_PWC: DWC_PWC is depthwise convolution followed by pointwise convolution; XXX_PC: PC is part value P = number of input channels (input depth).

4.2. VGG-16 on CIFAR-10

In this experiment, we use the VGG-16 architecture [30]. In the CIFAR-10 dataset, each image size is of 32×32 size on RGB scale. In VGG-16 architecture, there are 13 convolutional layers which use standard CONV2D convolution, and after each layer, we add batch normalization. We are using the same setting as described in [20]. The values of hyper-parameters are: weight decay = 5e-4, batch size = 128, initial learning rate = 0.1 which is decade by 0.1 after every 50 epochs.

Except for the initial convolution layer (i.e., CONV_1), All remaining 12 standard convolutional layers are replaced by our HetConv layers (same P value for all 12 layers) while keeping the number of filers same as earlier. As shown in Table-1, the value of P increase, the values of FLOPs (computation) decreases without any significant drop in accuracy. We also experimented for HetConv with SE technique and found that SE increases the accuracy initially but later due to over-fitting, it starts degrading the model performance (accuracy) as shown in Table-1.

4.2.1 Comparison with groupwise followed by pointwise convolution

We experimented with groupwise followed by pointwise convolution, where all standard convolutional layers (except the initial convolution layer, i.e., CONV_1) are replaced by two layers (groupwise convolutional layer with group size 4 and pointwise convolutional layer). Now the model has latency one. As shown in Table-1, VGG-16_GWC4_PWC has 92.76% accuracy whereas our model VGG-16_P4 has significantly higher 93.93% accuracy with lesser FLOPs.

4.2.2 Comparison with depthwise followed by pointwise convolution

We experimented with depthwise followed by pointwise convolution, where all standard convolutional layers (except the initial convolution layer, i.e., CONV_1) are replaced by two layers (depthwise convolutional layer and pointwise convolutional layer). Now the model has latency one. As shown in Table-1, VGG-16_DWC_PWC has only 91.27% accuracy on CIFAR-10 whereas our model VGG-16_P64 has significantly higher 93.42% accuracy with comparable FLOPs.

We also experimented with different P values for different layers. Except for the initial convolution layer (i.e., CONV_1), all remaining 12 standard convolutional layers are replaced by our HetConv layers with P = number of input channels. As shown in Table-1, our model

Model	Acc%	FLOPs	FLOPs Reduced (%)	Parameters	Parameters Reduced (%)
VGG-16_P1	94.06	313.74M	–	15.00M	–
VGG-16_P1_SE	94.13	314.19M	–	15.22M	–
VGG-16_P2	93.89	175.23M	44.15	8.45M	43.68
VGG-16_P2_SE	94.11	175.67M	44.00	8.68M	42.99
VGG-16_P4	93.93	105.98M	66.22	5.17M	65.45
VGG-16_P4_SE	94.29	106.42M	66.08	5.41M	64.48
VGG-16_GWC4_PWC	92.76	107.67M	65.68	5.42M	–
VGG-16_P8	93.92	71.35M	77.26	3.54M	76.40
VGG-16_P8_SE	93.97	71.79M	77.12	3.77M	75.22
VGG-16_P16	93.96	54.04M	82.78	2.72M	81.86
VGG-16_P16_SE	93.63	54.48M	82.64	2.95M	80.59
VGG-16_P32	93.73	45.38M	85.54	2.31M	84.58
VGG-16_P32_SE	93.41	45.82M	85.39	2.54M	83.28
VGG-16_P64	93.42	41.05M	86.92	2.11M	85.95
VGG-16_P64_SE	93.33	41.49M	86.77	2.34M	84.63
VGG-16_DWC_PWC	91.27	38.53M	87.72	1.97M	–
VGG-16_PC	92.53	38.18M	87.83	1.93M	–
VGG-16_PC_SE	93.08	38.62M	87.69	2.15M	–

Table 1. The table shows the detail results for VGG-16 on CIFAR-10 in different setups.

VGG-16_PC and VGG-16_PC_SE still performing better than VGG-16_DWC_PWC which shows the superior performance of our HetConv over DWC+PWC.

4.2.3 Comparison with FLOPs compression methods

Method	Error%	FLOPs Reduced(%)
Li-pruned [21]	6.60	34.20
SBP [25]	7.50	56.52
SBPa [25]	9.00	68.35
AFP-E [3]	7.06	79.69
AFP-F [3]	7.13	81.39
VGG-16_P32 (Ours)	**6.27**	**85.54**
VGG-16_P64 (Ours)	**6.58**	**86.92**

Table 2. The table shows the comparison of our models with state-of-art model compression methods for VGG-16 architecture on the CIFAR-10 dataset.

As shown in Table-2, our models VGG-16_P32, and VGG-16_P64 have significantly better accuracy as compare to state-of-art model compression methods. We reduced $\sim$ 85% FLOPs with no loss in accuracy whereas FLOPs compression methods suffer a significant loss in accuracy (more than 1%) as shown in Table-2.

4.3. ResNet-56 on CIFAR-10

We experimented with ResNet-56 architecture [8] on the CIFAR-10 dataset. ResNet-56 consists of three stages of the convolutional layer of size 16-32-64 where each convolution layer in each stage contains the same 2.36M FLOPs, and the total FLOP is 126.01M. We trained the model using the same parameters proposed by [8]. Except for the initial convolution layer, all remaining standard convolutional layers are replaced by our HetConv layers while keeping the number of filers same as earlier.

As shown in Table-3, our models ResNet-56_P2, and ResNet-56_P4 have significantly better accuracy as compare to state-of-art model compression methods with higher FLOPs reduction. We also experimented for HetConv with SE technique and found that SE performance is not as expected due to over-fitting.

4.4. MobileNet on CIFAR-10

We experimented with MobileNet architecture on the CIFAR-10 dataset. Except for the initial convolution layer, all remaining convolutional layers are replaced by our HetConv layers while keeping the number of filers same as

Method	Error%	FLOPs Reduced (%)
Li-B [21]	6.94	27.6
NISP [41]	6.99	43.6
CP [11]	8.20	50.0
SFP [10]	6.65	52.6
AFP-G [3]	7.06	60.9
ResNet-56_P1	6.41	–
ResNet-56_P2	**6.40**	**44.30**
ResNet-56_P4	**6.71**	**66.45**
ResNet-56_P1_SE	7.16	–
ResNet-56_P2_SE	6.75	44.27
ResNet-56_P4_SE	7.79	66.42

Table 3. The table shows the detail results and comparison with state-of-art model compression methods for ResNet-56 on CIFAR-10 in different setups.

Method	Accuracy (%)	FLOPs
MobileNet [12]	91.17	46.36M
MobileNet_P32	**92.06**	55.94M
MobileNet_P32_SE	**92.17**	56.91M

Table 4. The table shows the results for MobileNet [12] on CIFAR-10 in different setups.

earlier. In our model, two convolutional layers (depthwise convolutional layer and pointwise convolutional layer) is replaced by one HetConv convolutional layer which reduces the latency from one to zero.

As shown in Table-4, our models MobileNet_P32, and MobileNet_P32_SE have the significantly better accuracy (close to 1%) as compare to MobileNet model with almost similar FLOPs on MobileNet architecture which again shows the superior performance of our proposed HetConv convolution over depthwise+pointwise convolution.

4.5. VGG-16 on ImageNet

Method	Acc%(Top-1)	Acc%(Top-5)	FLOPs Reduced %
RNP (3X)[22]	–	87.57	66.67
ThiNet-70 [24]	69.8	89.53	69.04
CP 2X [11]	–	89.90	50.00
VGG-16_P1	71.3	90.2	–
VGG-16_P4	**71.2**	**90.2**	**65.8**

Table 5. Table shows the results for the VGG-16 on ImageNet [29]. Our model has no loss in accuracy as compare to state-of-art [11, 24] pruning approaches while significantly higher FLOPs reduction.

We experimented with VGG-16 [30] architecture on the large-scale ImageNet [29] dataset. Except for the initial convolution layer, all remaining convolutional layers are replaced by our HetConv layers while keeping the number of filers same as earlier. Our model VGG-16_P4 shows the state-of-art result over the recent approach proposed for flop compression. Channel-Pruning (CP) [11] has the 50.0% FLOP compression while we have 65.8% FLOP compression with no loss in accuracy. Please refer to Table-5 for the more detail results.

4.6. ResNet-34 on ImageNet

Method	Error (top-1)%	FLOPs	FLOPs Reduced(%)
Li-B [21]	27.83	2.7G	24.2
NISP [41]	27.69	–	43.76
ResNet-34_P1	26.80	3.6G	–
ResNet-34_P4	**27.00**	1.3G	**64.48**
ResNet-34_P4_SE	**26.50**	1.3G	**64.48**

Table 6. Table shows the results for ResNet-34 on ImageNet [29]. Our model has no loss in accuracy as compare to state-of-art [21, 41] pruning approaches while significantly higher FLOPs reduction in different setups.

We experimented with ResNet-34 [8] architecture on the large-scale ImageNet [29] dataset. Except for the initial convolution layer, all remaining convolutional layers are replaced by our HetConv layers. Our model ResNet-34_P4 shows the state-of-art result over the previously proposed methods. NISP [41] has the 43.76% FLOP compression while we have 64.48% FLOP compression with significantly better accuracy. For more details, please refer to Table-6.

4.7. ResNet-50 on ImageNet

Method	Error (top-1)%	FLOPs	FLOPs Reduced(%)
ThiNet-70 [24]	27.90	–	36.8
NISP [41]	27.33	–	27.31
ResNet-50_P1	23.86	4.09G	–
ResNet-50_P4	**23.84**	2.85G	**30.32**

Table 7. Table shows the results for ResNet-50 on ImageNet [29]. Our model has no loss in accuracy as compare to state-of-art [24, 41] flop pruning approaches.

ResNet-50 [8] is a deep convolutional neural network having 50 layers with the skip/residual connection. In this architecture, we replace standard convolutions with our proposed HetConv convolution. The values of hyper-parameters are: weight decay = 1e-4, batch size = 256, initial learning rate = 0.1 which is decade by 0.1 after every 30 epochs and model is trained in 90 epochs.

It is clear from Table-7 that our model ResNet-50_P4 has no loss in accuracy while flop pruning approaches [24, 41] suffers from the heavy accuracy drop in top-1 accuracy. Our model is trained from scratch, but pruning approaches [24, 41] requires a pre-trained model and involve iterative pruning and fine-tuning which is a very time-consuming process.

5. Conclusion

In this work, we proposed a new type of convolution using heterogeneous kernels. We have compared our proposed convolution with the popular convolutions (depthwise convolution, groupwise convolution, pointwise convolution, and standard convolution) on various existing architectures (VGG-16, ResNet and MobileNet). Experimental results show that our HetConv convolution is more efficient (lesser FLOPs with better accuracy) as compared to existing convolutions. Since our proposed convolution does not increase the layer (replacing a layer with a number of layers, for example, MobileNet) to get FLOPs reduction, hence has latency zero. We also compared HetConv convolution based model with the FLOPs compression methods and shown that it produces far better results as compared to compression methods. In the future, using this type of convolution, we can design more efficient architectures.

References

[1] Wenlin Chen, James Wilson, Stephen Tyree, Kilian Weinberger, and Yixin Chen. Compressing neural networks with the hashing trick. In *ICML*, pages 2285–2294, 2015. 1

[2] Francois Chollet. Xception: Deep learning with depthwise separable convolutions. *CVPR*, 2017. 1, 2, 5

[3] Xiaohan Ding, Guiguang Ding, Jungong Han, and Sheng Tang. Auto-balanced filter pruning for efficient convolutional neural networks. *AAAI*, 2018. 3, 7

[4] Chelsea Finn, Pieter Abbeel, and Sergey Levine. Model-agnostic meta-learning for fast adaptation of deep networks. *ICML*, 2017. 2

[5] Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial nets. In *Advances in neural information processing systems*, pages 2672–2680, 2014. 2

[6] Song Han, Huizi Mao, and William J Dally. Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding. *ICLR*, 2016. 1, 3

[7] Kaiming He and Jian Sun. Convolutional neural networks at constrained time cost. In *2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 5353–5360. IEEE, 2015. 2

[8] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *CVPR*, pages 770–778, 2016. 1, 2, 6, 7, 8

[9] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Identity mappings in deep residual networks. In *European conference on computer vision*, pages 630–645. Springer, 2016. 2

[10] Yang He, Guoliang Kang, Xuanyi Dong, Yanwei Fu, and Yi Yang. Soft filter pruning for accelerating deep convolutional neural networks. *IJCAI*, 2018. 1, 7

[11] Yihui He, Xiangyu Zhang, and Jian Sun. Channel pruning for accelerating very deep neural networks. In *ICCV*, page 6, 2017. 1, 7, 8

[12] Andrew G Howard, Menglong Zhu, Bo Chen, Dmitry Kalenichenko, Weijun Wang, Tobias Weyand, Marco Andreetto, and Hartwig Adam. Mobilenets: Efficient convolutional neural networks for mobile vision applications. *arXiv preprint arXiv:1704.04861*, 2017. 1, 2, 3, 4, 5, 6, 8

[13] Jie Hu, Li Shen, and Gang Sun. Squeeze-and-excitation networks. *CVPR*, 2018. 3, 6

[14] Gao Huang, Zhuang Liu, Laurens Van Der Maaten, and Kilian Q Weinberger. Densely connected convolutional networks. In *CVPR*, 2017. 2

[15] Forrest N Iandola, Song Han, Matthew W Moskewicz, Khalid Ashraf, William J Dally, and Kurt Keutzer. Squeezenet: Alexnet-level accuracy with 50x fewer parameters and¡ 0.5 mb model size. *arXiv preprint arXiv:1602.07360*, 2016. 1

[16] Yani Ioannou, Duncan Robertson, Roberto Cipolla, Antonio Criminisi, et al. Deep roots: Improving cnn efficiency with hierarchical filter groups. 2017. 3

[17] Yani Ioannou, Duncan Robertson, Jamie Shotton, Roberto Cipolla, and Antonio Criminisi. Training cnns with low-rank filters for efficient image classification. *arXiv preprint arXiv:1511.06744*, 2015. 3

[18] Alex Krizhevsky and Geoffrey Hinton. Learning multiple layers of features from tiny images. 2009. 6

[19] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In *NIPS*, pages 1097–1105, 2012. 1, 2

[20] Hao Li, Asim Kadav, Igor Durdanovic, Hanan Samet, and Hans Peter Graf. Pruning filters for efficient convnets. *arXiv preprint arXiv:1608.08710*, 2016. 3, 6

[21] Hao Li, Asim Kadav, Igor Durdanovic, Hanan Samet, and Hans Peter Graf. Pruning filters for efficient convnets. *ICLR*, 2017. 1, 7, 8

[22] Ji Lin, Yongming Rao, Jiwen Lu, and Jie Zhou. Runtime neural pruning. In *Advances in Neural Information Processing Systems*, pages 2181–2191, 2017. 8

[23] Christos Louizos, Karen Ullrich, and Max Welling. Bayesian compression for deep learning. In *NIPS*, pages 3288–3298, 2017. 3

[24] Jian-Hao Luo, Jianxin Wu, and Weiyao Lin. Thinet: A filter level pruning method for deep neural network compression. In *CVPR*, pages 5058–5066, 2017. 1, 3, 8

[25] Kirill Neklyudov, Dmitry Molchanov, Arsenii Ashukha, and Dmitry P Vetrov. Structured bayesian pruning via log-normal multiplicative noise. In *NIPS*, pages 6775–6784, 2017. 7

[26] Mehdi Noroozi and Paolo Favaro. Unsupervised learning of visual representations by solving jigsaw puzzles. In *European Conference on Computer Vision*, pages 69–84. Springer, 2016. 2

[27] Mohammad Rastegari, Vicente Ordonez, Joseph Redmon, and Ali Farhadi. Xnor-net: Imagenet classification using binary convolutional neural networks. In *ECCV*, pages 525–542. Springer, 2016. 1, 3

[28] Shaoqing Ren, Kaiming He, Ross Girshick, and Jian Sun. Faster r-cnn: Towards real-time object detection with region proposal networks. In *NIPS*, pages 91–99, 2015. 2

[29] Olga Russakovsky, Jia Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, Michael Bernstein, et al. Imagenet large scale visual recognition challenge. *IJCV*, 115(3):211–252, 2015. 6, 8

[30] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. *ICLR*, 2015. 1, 2, 3, 6, 8

[31] Pravendra Singh, Vinay Sameer Raja Kadi, Nikhil Verma, and Vinay P Namboodiri. Stability based filter pruning for accelerating deep cnns. *WACV*, 2019. 1, 3

[32] Pravendra Singh, Neeraj Matiyali, Vinay P Namboodiri, et al. Multi-layer pruning framework for compressing single shot multibox detector. *WACV*, 2019. 1, 3

[33] Pravendra Singh, Vinay Kumar Verma, Piyush Rai, and Vinay P Namboodiri. Leveraging filter correlations for deep model compression. *arXiv preprint arXiv:1811.10559*, 2018. 1, 3

[34] Jake Snell, Kevin Swersky, and Richard Zemel. Prototypical networks for few-shot learning. In *Advances in Neural Information Processing Systems*, pages 4077–4087, 2017. 2

[35] Christian Szegedy, Sergey Ioffe, Vincent Vanhoucke, and Alexander A Alemi. Inception-v4, inception-resnet and the impact of residual connections on learning. In *AAAI*, volume 4, page 12, 2017. 1, 2, 4, 5

[36] Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, and Andrew Rabinovich. Going deeper with convolutions. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 1–9, 2015. 1, 2, 5

[37] Christian Szegedy, Vincent Vanhoucke, Sergey Ioffe, Jon Shlens, and Zbigniew Wojna. Rethinking the inception architecture for computer vision. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 2818–2826, 2016. 1

[38] Vincent Vanhoucke. Learning visual representations at scale. *ICLR invited talk*, 2014. 1, 2

[39] V Kumar Verma, Gundeep Arora, Ashish Mishra, and Piyush Rai. Generalized zero-shot learning via synthesized examples. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2018. 2

[40] Saining Xie, Ross Girshick, Piotr Dollár, Zhuowen Tu, and Kaiming He. Aggregated residual transformations for deep neural networks. In *Computer Vision and Pattern Recognition (CVPR), 2017 IEEE Conference on*, pages 5987–5995. IEEE, 2017. 1, 3

[41] Ruichi Yu, Ang Li, Chun-Fu Chen, Jui-Hsin Lai, Vlad I Morariu, Xintong Han, Mingfei Gao, Ching-Yung Lin, and Larry S Davis. Nisp: Pruning networks using neuron importance score propagation. *CVPR*, 2018. 1, 3, 7, 8

[42] Xiangyu Zhang, Xinyu Zhou, Mengxiao Lin, and Jian Sun. Shufflenet: An extremely efficient convolutional neural network for mobile devices. 2018. 1, 4, 5

[43] Xiangyu Zhang, Jianhua Zou, Xiang Ming, Kaiming He, and Jian Sun. Efficient and accurate approximations of nonlinear convolutional networks. In *NIPS*, pages 1984–1992, 2015. 3