The Ramifications of Making Deep Neural Networks Compact

Nandan Kumar Jha, Sparsh Mittal, Govardhan Mattela
Department of Computer Science and Engineering, IIT Hyderabad, India
Email: {cs17mtech11010, sparsh, cs17resch01004}@iith.ac.in

***Abstract*—The recent trend in deep neural networks (DNNs) research is to make the networks more compact. The motivation behind designing compact DNNs is to improve energy efficiency since by virtue of having lower memory footprint, compact DNNs have lower number of off-chip accesses which improves energy efficiency. However, we show that making DNNs compact has indirect and subtle implications which are not well-understood. Reducing the number of parameters in DNNs increases the number of activations which, in turn, increases the memory footprint. We evaluate several recently-proposed compact DNNs on Tesla P100 GPU and show that their "activations to parameters ratio" ranges between 1.4 to 32.8. Further, the " memory-footprint to model size ratio" ranges between 15 to 443. This shows that a higher number of activations causes large memory footprint which increases on-chip/off-chip data movements. Furthermore, these parameter-reducing techniques reduce the arithmetic intensity which increases on-chip/off-chip memory bandwidth requirement. Due to these factors, *the energy efficiency of compact DNNs may be significantly reduced which is against the original motivation for designing compact DNNs*.**

***Index Terms*—Deep neural networks (DNNs), compact DNNs, activations, parameters, energy-efficiency, embedded systems.**

I. Introduction

Deep neural networks (DNNs) have shown phenomenal results in various domains such as image classification and object detection, etc. [1]–[3]. After the success of AlexNet [1], to improve accuracy, researchers have proposed even deeper [2], [4] and wider [5]–[7] networks which are deemed as over-parameterized. These networks have huge compute, memory and power demands which hinders their deployment on resource-constrained embedded and mobile devices [8]. To enable the deployment of DNNs on resource-constrained platforms, researchers have proposed two types of heuristics: (1) compressing the existing over-parameterized deeper and wider networks and (2) designing new algorithms which have very few parameters i.e. compact models. For example, Han et al. [9] propose magnitude-based pruning of filter weights and Yang et al. [10] propose energy-aware pruning to improve energy efficiency. These pruning methods, however, require exhaustive retraining to achieve the accuracy of the original pre-trained model.

Compact DNNs have the advantage that they avoid retraining overheads and after training, they can be directly deployed on resource constrained devices. The current trends in designing compact DNNs is to reduce the number of parameters and computations by leveraging error-tolerance of DNN application domains. Reducing the number of parameters helps in fitting the network in limited on-chip memory and avoids expensive off-chip access. This makes the DNNs energy efficient. The computational cost of a DNN is measured in terms of number of MAC (multiply-accumulate) operations performed in conv and FC layers. Since measuring the memory footprint and total energy consumption is not straightforward, researchers generally use number of parameters and number of MACs (respectively) as their proxies [11]. However, this approach has crucial limitations. The total memory footprint is sum of the (1) size of weights (2) size of activations and (3) gradients corresponding to activations and parameters. Hence it depends on both the number of parameters and activations as well. However, since the number of activations cannot be estimated from the number of parameters, the number of parameters is not a good indicator of memory footprint. Further, as shown in Figure 1, one MAC operation requires three read and one write operations. Hence, the energy consumed in each MAC operation depends on (1) the location of the operands in the memory hierarchy, such as register file, cache or main memory which decides the operand fetch energy [12] and (2) the type of convolution such as 3×3, 1×1 or depth-wise separable convolution which decides the degree of reuse [13]. Hence, the number of MAC is not an accurate indicator of the energy consumption of a DNN.

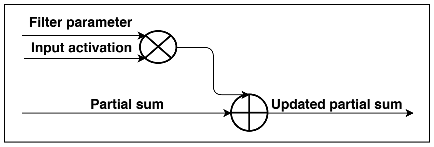

Fig. 1: Illustration of a MAC operation.

In this paper, we study state-of-the-art compact DNNs and analyze the unforeseen implications of reducing the number of parameters. For example, 1.0-G-SqNxt-23, a variant of SqueezeNext [14] has 112× fewer parameters than AlexNet (Table II) but higher memory footprint than AlexNet (Table III). With larger batch size (B), this becomes even worse and for $B = 100$, 1.0-G-SqNxt-23 consumes 5.7× higher memory compared to AlexNet. Further, 1.0-G-SqNxt-23 has 3.27× fewer MACs than AlexNet (Table II) but the energy efficiency of 1.0-G-SqNxt-23 is 45× lower than that of AlexNet (Table III). On digging deeper to understand the sources of inefficiency, we found that 1.0-G-SqNxt-23 has

Support for this work was provided by Science and Engineering Research Board (SERB), India, award number ECR/2017/000622.

2380-6923/19/$31.00 ©2019 IEEE
DOI 10.1109/VLSID.2019.00056

8.7× higher activation and 28.5× lower MACs/activation ratio compared to AlexNet. Lower MACs/activation ratio leads to lower arithmetic intensity and makes the DNN bandwidth bound. This increases the on-chip/off-chip memory access and results into higher energy consumption. We summarize our contributions as follows.

- We analyze DNNs which are representative of the state-of-the-art compact DNNs. We perform kernel-level analysis of compact DNNs to get insights into utilization of compute resources by the MACs and the performance bottlenecks of each DNN.
- We find implications of making DNNs compact on memory footprint, energy efficiency and throughput. We find that memory footprint depends not only on the number of parameters but activations also. In fact, the contribution of activations in memory footprint is very high.
- Since measuring the arithmetic intensity is relatively difficult, we propose using MACs/parameter ratio and MACs/activation ratio as the proxies for them. Low arithmetic intensity indicates lower degree of reuse of parameters and activations which increases energy required for processing inputs.

II. Ramifications of Making DNNs Compact

Notations: Table I shows the terms used for measuring the dimension of DNN layers. Equations 1, 2 and 3 show the formula for computing the number of parameters, activations and MACs (respectively), assuming batch size of one [15].

TABLE I: Dimension of layers in DNN

DNN shape parameters	Description
N	# input (input feature map) channels
M	# output (output feature map) channels
S_N	input (input feature map) width/height
S_M	output (output feature map) width/height
S_F	filter (2-D) width/height

$$\#Parameters = N \times M \times S_F \times S_F \tag{1}$$

$$\#Activations = M \times S_M \times S_M \tag{2}$$

$$\#MACs = N \times M \times S_M \times S_M \times S_F \times S_F \tag{3}$$

Filter factorization and its impact: To reduce the number of parameters, larger size filter is factorized into equivalent smaller size filters. As shown in Figure 2, factorizing one 5×5 filter into two 3×3 filter produces equal-sized output feature map (here in all convolution padding is zero and stride is one). We study the impact of this filter-factorization on the number of parameters, activations and MACs. We found that this factorization reduces the number of parameters and MACs by 52% and 51.3% respectively, but increases the number of output activations by 102%. This may lead to an increase in memory footprint in DNNs with high degree of nonlinearity (refer section III-C) even though the number of parameters is reduced by half.

However, previous works have ignored the impact of this increase on memory footprint and energy efficiency. Although this factorization increases the MACs/parameter ratio only by 1.34% but decreases the MACs/activation ratio by 61%. This leads to lower arithmetic intensity which, in turn, increases the memory accesses. *This reduces the overall energy efficiency*. Hence, contrary to the popular belief, this parameter-reducing techniques degrade energy efficiency.

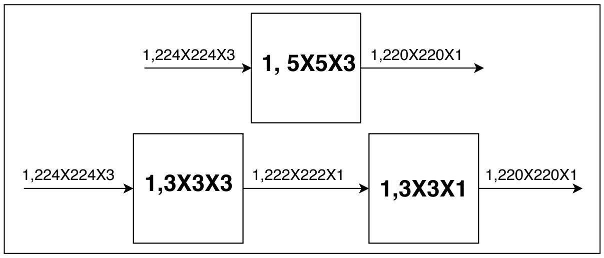

Fig. 2: Factorizing one 5×5 filter into two 3×3 filters produces equal sized output feature map.

We now discuss the architecture and building blocks of several compact DNNs. Table II summarizes their characteristics. We also discuss the implications of the parameter-reducing techniques and highlight the sources of inefficiency.

A. SqueezeNet

To reduce the number of parameters, SqueezeNet [16] uses fire module as a building block. Both SqueezeNet versions, V1.0 and V1.1, have 8 fire modules (fire2 - fire9) which consist of squeeze and expand layers. The squeeze layer has only 1×1 filters which serves as the bottleneck layer similar to that in the inception module [3] and reduces the dimensionality of input feature map before 3×3 convolutions performed in the expand module. It reduces the number of computation (MACs) and the parameters. The fire module has majority of 1×1 filters. Replacing a 3×3 filter with a 1×1 filter reduces the number of parameters by 9×. SqueezeNetV1.1 has 2.2× fewer MACs than SqueezeNetV1.0.

Limitations: In squeeze layer of fire module, 1×1 filter reduces the number of activations whereas in expand layer, it increases the number of activations. Compared to AlexNet, SqueezeNet has 50× fewer parameters and at $B = 1$, it has lower memory footprint, however with increasing value of batch size, its memory footprint increases at a faster rate. For example, at $B = 128$, AlexNet and SqueezeNetV1.0 consume 2287MB and 8225MB of global memory on GPU, respectively. Also, using small filters lowers data reuse as well as arithmetic intensity which makes the convolution operation bandwidth bound. By comparison, in large filters, convolution operation is computation-bound. Reducing the degree of data reuse lowers energy efficiency since the MAC operands are fetched from lower levels of memories such as DRAM and not from upper levels such as L1 cache.

B. SqueezeNext

In SqueezeNext family [14], 1.0-SqNxt-23 is the baseline model which has 23 building blocks, where each building block is an aggressive version of the fire block. Each fire block has (1) two stage bottleneck layers (2) decomposed 3×3 (into 1×3 and 3×1) filters and (3) skip connection across the block. Each bottleneck stage reduces the number of output

TABLE II: Characteristics of compact DNNs (Params and Acts refer to parameters and activations of DNNs, respectively).

Model Name	Image size	MACs (M)	#Params (M)	#Acts (M)	Acts/Params	MACs/Params	MACs/Acts
AlexNet	224 × 224	723	60.97	2.05	0.03	11.86	352.65
SqueezeNet-V1.0	224 × 224	848	1.25	12.3	9.84	678.08	68.91
SqueezeNet-V1.1	224 × 224	349	1.24	7.2	5.81	281.57	48.49
1.0-G-SqNxt-23	224 × 224	221	0.54	17.81	32.80	406.35	12.39
1.0-SqNxt-23	224 × 224	273	0.72	17.81	24.84	380.50	15.32
1.0-SqNxt-23v5	224 × 224	225	0.93	14.06	15.12	242.04	16.01
2.0-SqNxt-23	224 × 224	726	2.36	32.21	13.65	307.62	22.54
2.0-SqNxt-23v5	224 × 224	703	3.22	24.66	7.66	218.41	28.52
1.0-MobileNet-224	224 × 224	574	4.23	20.32	4.80	135.65	28.24
DenseNet-121	224 × 224	3080	7.98	69.99	8.77	385.96	44.01
GoogLeNet	224 × 224	1590	7.00	10.06	1.44	227.14	158.05
Inception-V2	231 × 231	2200	11.19	18.03	1.61	196.60	122.02

feature map by a factor of two and this reduces the number of input feature maps to 3×3 convolution in the expand layer. Decomposing a 3×3 filter into a 1×3 and a 3×1 filter leads to an additional reduction of $1.5\times$ in the number of parameters. It further reduces the number of parameters by using a bottleneck layer before the fully connected layer [14].

Among these 23 blocks, four groups of size 6 blocks, 6 blocks, 8 blocks and 1 blocks (from input to output) have input feature map resolution of 55×55 (64 channels), 55×55 (32 channels), 28×28 (64 channels) and 14×14 (128 channels), respectively. Blocks with higher input resolution lead to lower arithmetic intensity because they have higher number of activations. The distribution of blocks in 1.0-G-SqNxt-23, 1.0-SqNxt-23 and 2.0-SqNxt-23 [14] are [6,6,8,1] whereas in the v5 versions (1.0-SqNxt-23v5 and 2.0-SqNxt-23v5) this is [2,4,14,1] [14].

Limitations: SqueezeNext uses a two-stage bottleneck layer in each fire module and the number of such fire blocks is nearly $3\times$ higher compared to that in SqueezeNet. This aggravates the problem of 1×1 convolution since it leads to low data reuse and low arithmetic intensity. The skip connection across a fire block further aggravates the problem of higher number of activations because it requires more feature maps in global memory of GPU. Although the 1.0-G-SqNxt-23 has $112\times$ fewer parameters than AlexNet but its memory footprint is higher than that of the AlexNet. As shown in Table III, at batch size one, the memory consumption of 1.0-G-SqNxt-23 and AlexNet are 1019MB and 1015MB, respectively. With increasing batch size, the memory footprint of SqueezeNext increases rapidly.

C. MobileNet

MobileNet [15] was designed to reduce the inference time. MobileNet reduces the computational complexity, i.e. number of MACs, by allowing only a single filter to convolve with each input feature map. MobileNet uses depthwise separable convolution which factorizes a standard convolution into depthwise convolutions and pointwise convolutions. In depthwise convolution, single filter (3×3) is used for each input feature map whereas pointwise convolution uses 1×1 filters to linearly combine the output feature maps of depthwise convolutions. Factorizing the standard convolution into two stages enables separating the filtering and combining operations. This reduces both the number of parameters and MACs.

Limitations: In MobileNet, 1×1 convolution operation is responsible for 94.9% of total MACs [15]. *Although the 1×1 convolution does not require reordering in memory, thus no im2col overhead, it has low data reuse (compare to 3×3 filter). Low data reuse implies low arithmetic intensity and high bandwidth requirement. This makes MobileNets energy inefficient* (Table III).

As shown in Table III, for MobileNet, `Gemv2T` and `Gemv2N` kernels account for 55.5% and 30.5% of inference time, respectively. Hence, MobileNet has stalls due to both memory dependencies and instruction dependencies. By using larger batch size, memory dependencies can be mitigated but instruction dependencies cannot be mitigated. This is evident from the fact that the inference time with $B = 1$ and $B = 90$ is 29.4ms and 23.6ms, respectively. Higher fraction of `Gemv2T` and `Gemv2N` kernels in total inference time leads to poor utilization of streaming multiprocessors (SMs). This further aggravates the low data re-usability problem of 1×1 convolution. These MACs with low data re-usability and poor SMs utilization result into a throughput of 42 FPS (frames-per-second), which is lowest among all the compact DNNs studied in this paper.

D. DenseNet

DenseNet [17] comprises of dense blocks and transition layers. In dense blocks, each layer is connected to all the previous layers and produce same sized output feature maps. In transition layers, the size of output feature maps reduces due to use of convolution and pooling. Dense blocks have dense connectivity between layers such that each layer can access the output feature maps of all the previous layers. DenseNet reduces the number of parameters in following two ways. (1) As each layer in the dense block can access the collective knowledge of the network, it does not require larger number of filters. Specifically, number of filters in dense blocks of DenseNet-121 are 32 and 128 whereas between 96 and 384 in AlexNet. (2) DenseNet introduces the bottleneck layer to reduce the number of feature maps in transition layers.

Limitations: Because of the dense connectivity in dense blocks, it requires multiple copies of the feature maps in the memory. Due to this, among the compact DNNs studied here, DenseNet has the highest memory footprint of 1405MB, as shown in Table III. *With a batch size of only 20, it consumes nearly entire global memory of GPU P100.*

E. GoogLeNet and Inception-V2

GoogLeNet [3] has 12× fewer parameters than AlexNet but is significantly more accurate than AlexNet. It uses inception module as the building block and 1×1 filters as the bottleneck layer inside the inception module. This bottleneck layer reduces the dimension of input feature map and hence, the number of MACs, before 3×3 and 5×5 convolutions. This technique allows increasing the depth and width of network, which makes the network more expressive and discriminative, without increasing the computational budget. Similarly, Inception-V2 [6] uses inception module in more aggressive manner and factorizes the 5×5 convolution into 3×3 and further into 1×3 and 3×1 convolutions. This reduces the number of parameters and MACs.

Limitations: Although factorizing the larger filter into smaller filters reduces computation and parameters but it increases the number of activations, as shown in Figure 2. This increase in activations and decrease in MACs lowers MACs/activation ratio significantly and hence the arithmetic intensity which in turn increases bandwidth demand and reduces energy efficiency. This also makes Inception-V2 unsuitable for larger batch size. For example, on P100 GPU, GoogLeNet can work with a batch size of 128 whereas Inception-V2 cannot work due to exceeding the global memory size limitation.

III. Experimental Results

A. Methodology

We run the DNNs using Caffe framework [18] on Tesla P100 GPU. P100 GPU has a global memory capacity of 12193 MB. Since P100 is a high-end GPU with ample amount of compute and memory resources, this by itself does not present any resource bottlenecks, and hence, we can focus on the characteristics of DNN itself, for example, we can find the performance bottlenecks of a DNN with large batch size. We use CUDA version 8.0 and cuDNN version 7.0.1. For measuring the memory footprint and average power measurement of GPU, we use *nvidia-smi* utility. We record 50 samples per second. For measuring the performance of individual kernels, we use Nvidia visual profiler.

Metrics: We measure the correlation between different values using the Pearson product-moment correlation coefficient (PPMCC). PPMCC varies between +1 and -1 where values close to +1 or -1 indicate high degree of positive or negative (respectively) linear correlation and a value of 0 indicates no linear correlation. For getting insights into characteristics of GPU kernels, we use the following metrics [19], [20]. *Compute utilization* is combined utilization of arithmetic units, control-flow units, load-store units and functional units. *Bandwidth utilization* shows the percentage utilization of the global memory bandwidth. *Occupancy* shows the ratio of total number of active warps and the maximum warps supported in a streaming multiprocessor (SM). *Warp execution efficiency* is the average percentage of active threads in each executed warp.

B. Results

Table III summarizes the results on memory footprint, latency, throughput and energy efficiency. Also it illustrates the percentage contribution of kernels in total inference time. We analyze these results to get deeper insights. Table II shows the number of parameters and activations of various DNNs. Notice that *except AlexNet, all the DNNs have more activations than parameters*. AlexNet has 61M parameters but only 2.05M activations whereas 1.0-G-SqNxt-23, which is a variant of SqueezeNext, has 0.54M parameters but 17.81M activations. Also, DensenNet-121 has 8M parameters and 70M activations. Evidently, *compact DNNs have large disparity in number of parameters and activations*.

As shown in Table IV, there is a high linear correlation between number of activations and memory footprint. This substantiates our claims that (1) the memory footprint depends on both the number of parameters and activations (2) the contribution of activations in memory footprint is significantly higher than that of parameters. Hence, we recommend that *future techniques for designing compact DNNs should also focus on reducing the number of activations*.

Figure 3 shows activations to parameters ratio and memory footprint to model size ratio. Evidently, as the ratio of activations to parameters (left vertical axis) increases, the ratio of total memory footprint to model size (right vertical axis) also increases. Compared to AlexNet, which has activations to parameters ratio 0.03 and total memory-footprint to model size ratio 4.16, 1.0-G-SqNxt-23 has activations to parameters ratio of 32.8 and total memory-footprint to model size ratio of 443. The PPMCC between these two ratios is 0.96 and thus, these two ratios are strongly correlated.

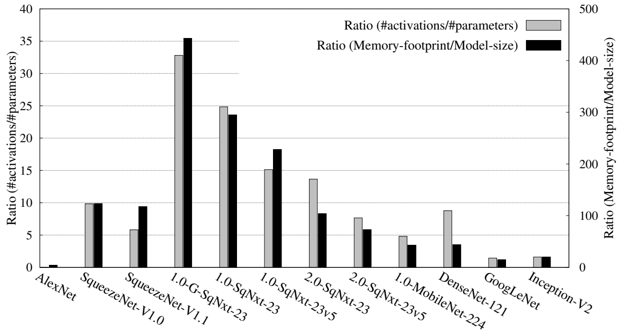

Fig. 3: Activations/Parameters and memory-footprint/model-size ratio. Lower value of both ratios is better.

We have used batch size four, to utilize the GPU compute resources, in calculation of energy efficiency. Figure 4 shows the energy efficiency (left vertical axis), in terms of $MACs \times 10^9/Joule$. Clearly, energy efficiency depends not only on MACs/parameters ratio (right vertical axis) but also on the MACs/activations ratio (right vertical axis). *The PPMCC between energy efficiency and MACs/parameter ratio is -0.18 and that between energy efficiency and MACs/activation ratio is 0.88.* This shows that there is high linear correlation between MACs/activation ratio and energy efficiency. Therefore networks with comparable MACs/parameter ratio, but

TABLE III: Inference time (ms), memory footprint (MB), energy efficiency ($10^9 \times MACs/Joule$) and throughput (FPS). The last three columns show the percentage contribution of kernels `Gemv2T`, `Gemv2N` and `Gemmk1` in the total inference time.

Model Name	Inference time	Memory footprint	Energy efficiency	Throughput	Gemv2T(%)	Gemv2N(%)	Gemmk1(%)
AlexNet	2.1	1015	75.94	4000	7.54	0.18	11.43
SqueezeNet-V1.0	3.8	615	40.95	1479	0	0	0
SqueezeNet-V1.1	3.5	587	27.07	2778	0	0	0
1.0-G-SqNxt-23	24.4	1019	1.69	763	40.92	5.41	8.64
1.0-SqNxt-23	24.5	885	2.16	770	42.13	5.7	9.35
1.0-SqNxt-23v5	23.8	867	1.92	1004	31.68	6.82	10.73
2.0-SqNxt-23	28.2	995	3.75	412	35.10	5.06	7.99
2.0-SqNxt-23v5	27.7	957	3.86	541	24.71	6.53	8.61
1.0-MobileNet-224	29.4	733	1.65	42	**55.47**	**30.5**	0.68
DenseNet-121	33.0	1405	8.41	182	10.04	0	5.34
GoogLeNet	11.2	801	21.95	1333	0.22	0.16	0.05
Inception-V2	19.0	987	14.05	658	6.38	0.03	3.98

TABLE IV: Correlation between values

X	Y	PPMCC(X,Y)
# Parameters	Memory footprint	0.24
# Activations	Memory footprint	0.75
# Parameters + # activations	Memory footprint	0.82

higher MACs/activation ratio have higher energy efficiency, for example 1.0-G-SqNxt-23 and 1.0-SqNxt-23. MACs/activation ratio of AlexNet is higher than that of all the compact DNNs studied here and this makes AlexNet the most energy efficient. Here, we recommend that *future techniques for designing energy efficient DNNs should focus on increasing the arithmetic intensity, primarily by increasing the the MACs/activation ratio.*

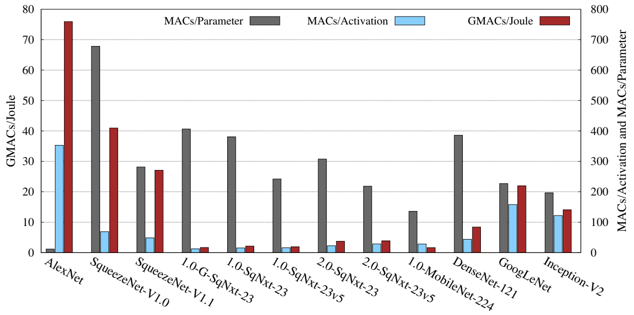

Fig. 4: Energy efficiency (left vertical axis) and MACs/parameter and MACs/activation (right vertical axis). For all three metrics, higher is better. *(better viewed in color).*

Kernel-level analysis: To gain more insights, we perform kernel level analysis by profiling the DNNs on Tesla P100 GPU. We focus on three cuBLAS kernels, viz. `Gemv2T`, `Gemv2N`, `Gemmk1` because these kernels present in all the DNNs, except SqueezeNet, we analyzed in this paper. Table V shows the resource utilization of these kernels along with stall reasons. Note that Gemm (general matrix-matrix multiplication) is a dense matrix style of computation and because of better data reuse [13], it has high arithmetic intensity. Gemv (general matrix-vector multiplication) is a sparse matrix style of computation and because of poor data reuse, it has lower arithmetic intensity. The letters 'T' in `Gemv2T` and 'N' in `Gemv2N` stand for transpose and not-transpose respectively. Gemm-style computation is faster than Gemv style [13] and we found that the `Gemmk1` kernel, which is based on Gemm style computation, has excellent SM utilization (Table V).

TABLE V: Analysis of kernel properties

Attributes/stall reasons	Gemv2T	Gemv2N	Gemmk1
Compute utilization (%)	15	8	5
Bandwidth utilization (%)	0.92	0.094	11
Occupancy (%)	6.2	6.2	11.8
Warp execution efficiency(%)	80.7	99.3	98.4
Branch divergence (%)	25	0	4
SM utilization	**Poor**	**Very poor**	**Excellent**
Memory dependency (%)	**44.3**	**3.6**	**54.4**
Instruction dependency (%)	**32**	**56.9**	**19**
Synchronization (%)	8.8	25	9.1
Others (%)	15	14.5	17.5

Both the compute utilization and bandwidth utilization of these three kernels is well below 60% and hence, the performance of these kernels is bounded by either instruction latency or memory latency. In `Gemv2T` and `Gemmk1`, memory dependency causes 44% and 54% of total stalls and hence, these kernels are bottlenecked by memory latency. Also, high branch divergence (25%) in `Gemv2T` reduces its warp execution efficiency which further reduces the utilization of SMs. For `Gemv2N`, instruction dependency causes 57% of total stalls and thus, it is bottlenecked by instruction latency.

It is well-known that the memory latency bottleneck can be mitigated by using larger batch size, whereas the instruction latency bottleneck can be mitigated only by higher instruction-level parallelism. A high value of instruction dependency leads to poor utilization of SMs. Hence, the SMs utilization of `Gemv2N` kernel is very poor. We conclude that a network, which spends high fraction of time in `Gemmk1` and low fraction of time in `Gemv2T` and `Gemv2N`, will show higher utilization of SMs. For example, MobileNet has 5.4× fewer MACs than DenseNet (Table II) but 4.3× lower throughput (Table III). This is because `Gemv2T` and `Gemv2N` together contribute 86% of inference time in MobileNet but only 10% of time in DenseNet (Table III).

C. Discussion

In linear DNNs, such as AlexNet and VGGNet, neighboring layers are connected in sequential order and producer-consumer relationships exist only between a layer and its immediate neighboring layers [21]. By comparison, in non-linear DNNs, such as SqueezeNet, SqueezeNext, DenseNet etc., producer-consumer relationships exist between a layer and more than one of its neighboring layers. This nonlinearity aggravates the problem of memory over-provisioning [21]. We found that activations/parameters ratio for linear DNNs is low, e.g., for AlexNet, it is only 0.03. By comparison, for all compact DNNs studied in this paper, the ratio is higher than one. *We propose using "activation to parameter ratio" as the measure of nonlinearity of the DNN.*

In what follows, we classify the compact DNNs in groups and discuss the implications of architectural characteristics on memory footprint, energy efficiency and throughput.

SqueezeNet family: Compared to the baseline V1.0, V1.1 has two changes [16]. First, it has low dimensional filters in conv1 layer (64, 3×3 filters) compared to the baseline (96,7×7 filters). This reduces the number of MACs and parameters by 87% while the number of activations are reduced by only 31% in the conv1 layer. Second, it introduces pooling layers earlier [16] which reduces the size of output feature map and hence, the number of activations. As shown in Table II, these changes (1) decrease the number of MACs leading to higher throughput, (2) decrease the number of activations leading to lower memory footprint, and (3) have negligible impact on the number of parameters. However, the value of metrics viz., MACs/activation and MACs/parameter are decreased and hence, V1.1 has lower energy efficiency than V1.0.

SqueezeNext family: The v5 versions of SqueezeNext have lower number of blocks in groups with higher input resolution and higher number of block in groups with lower input resolution. Although this change increases the number of parameters but reduces the number of activations hence memory footprint. Also, this reduces the number of MACs and percentage contribution of `Gemv2T` kernel in total inference time, which improves SMs utilization. Both these factors increase the throughput (Table III). Increasing the number of blocks in a group with lower input resolution and decreasing the number of blocks in groups with higher input resolution reduces MACs/parameter but increases MACs/activation. In 1.0-SqNxt-23v5, the increase in MACs/activation is very small which does not make it more energy efficient than 1.0-SqNxt-23. By contrast, 2.0-SqNxt-23v5 has higher MACs/activation than 2.0-SqNxt-23 which increases its energy efficiency.

Inception module, DenseNet and MobileNet: We compare inception module family (GoogLeNet and Inception-V2) with DenseNet and MobileNet. Among these, MobileNet and Inception-V2 have lowest and highest number of parameters, respectively. Also, MobileNet and DenseNet have the lowest and highest memory footprint, respectively. This is because DenseNet has the highest number of activations. DenseNet has highest number of activations due to its densely connected structure which leads to multiple copies of the same activation maps during the processing of inputs and it exponentially increases with larger batch size.

Further, among these DNNs, MobileNet has lowest number of MACs, so one may expect it to have the highest energy efficiency. But, *MobileNet is in fact the least energy efficient* due to (1) poor data reuse and (2) very high fraction of `Gemv2T` and `Gemv2N` kernels in total inference time (Table III). So MACs in MobileNet utilizing SMs very poorly. For the same reason, MobileNet also has the lowest throughput.

IV. Conclusion and Future work

In this paper, we have analyzed the state-of-the art compact DNNs and the implications of parameter-reducing techniques on memory footprint, energy efficiency and throughput. We show that there is very high correlation between number of activations and memory footprint. Naively reducing number of parameters can lead to very high number of activations which increases the memory footprint. Also, the activations/parameters ratio is representative of memory-footprint/model-size ratio. Lower number of MACs does not necessarily improve energy efficiency and throughput. Energy efficiency also depends on arithmetic intensity which we estimate in terms of MACs/parameter and MACs/activation. Throughput of these compact DNNs depends on the number of MACs as well as how these MACs are utilizing the SMs. We also provide important guidelines for designing compact and energy efficient DNNs: while reducing number of parameters and MACs, researchers should also ensure high arithmetic and operational intensity and low activations to parameters ratio. Our future work will focus on optimizing DNNs on different hardware platforms [22] and developing tools to estimate the energy efficiency of the DNNs at design time.

References

[1] A. Krizhevsky *et al.*, "Imagenet classification with deep convolutional neural networks," *NIPS*, pp. 1097–1105, 2012.

[2] K. Simonyan *et al.*, "Very deep convolutional networks for large-scale image recognition," *CoRR*, vol. abs/1409.1556, 2014.

[3] C. Szegedy *et al.*, "Going deeper with convolutions," in *CVPR*, 2015.

[4] H. Kaiming *et al.*, "Identity mappings in deep residual networks," in *European conference on computer vision*, 2016, pp. 630–645.

[5] S. Xie *et al.*, "Aggregated residual transformations for deep neural networks," *CoRR*, vol. abs/1611.05431, 2016.

[6] C. Szegedy *et al.*, "Rethinking the inception architecture for computer vision," in *CVPR*, June 2016.

[7] C. Szegedy *et al.*, "Inception-v4, inception-resnet and the impact of residual connections on learning," in *AAAI*, 2017.

[8] S. Mittal, "A Survey of FPGA-based Accelerators for Convolutional Neural Networks," *Neural computing and applications*, 2018.

[9] S. Han *et al.*, "Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding," *ICLR*, 2016.

[10] T.-J. Yang *et al.*, "Designing energy-efficient convolutional neural networks using energy-aware pruning," *CVPR*, pp. 6071–6079, 2017.

[11] T. Yang *et al.*, "Netadapt: Platform-aware neural network adaptation for mobile applications," *ECCV*, 2018.

[12] S. Mittal *et al.*, "DESTINY: A Comprehensive Tool with 3D and Multi-level Cell Memory Modeling Capability," *JLPEA*, 2017.

[13] L. Lai *et al.*, "Not all ops are created equal!" *SysML*, 2018.

[14] A. Gholami *et al.*, "Squeezenext: Hardware-aware neural network design," in *CVPR Workshops*, 2018.

[15] A. G. Howard *et al.*, "Mobilenets: Efficient convolutional neural networks for mobile vision applications," *CoRR*, vol. abs/1704.04861, 2017.

[16] F. N. Iandola *et al.*, "Squeezenet: Alexnet-level accuracy with 50x fewer parameters and <1mb model size," *arXiv:1602.07360*, 2016.

[17] G. Huang *et al.*, "Densely connected convolutional networks," *CVPR*, 2017.

[18] Y. Jia *et al.*, "Caffe: Convolutional architecture for fast feature embedding," in *ACM International Conference on Multimedia*, 2014.

[19] http://www.nvidia.in/object/tesla-p100-in.html, 2012.

[20] S. Mittal, "A Survey of Techniques for Architecting and Managing GPU Register File," *TPDS*, 2017.

[21] L. Wang *et al.*, "Superneurons: Dynamic GPU memory management for training deep neural networks," *SIGPLAN Not. abs/1801.04380*, 2018.

[22] S. Mittal, "A Survey of ReRAM-based Architectures for Processing-in-memory and Neural Networks," *MAKE*, 2018.