Not All Network Weights Need to Be Free

David Marwood
Google Research
Google, Inc
Mountain View, CA.
marwood@google.com

Michele Covell
Google Research
Google, Inc
Mountain View, CA.
covell@google.com

Shumeet Baluja
Google Research
Google, Inc
San Diego, CA.
shumeet@google.com

***Abstract*—As state of the art network models routinely grow to architectures with billions and even trillions of learnable parameters, the need to efficiently store and retrieve these models into working memory becomes a more pronounced bottleneck. This is felt most severely in efforts to port models to personal devices, such as consumer cell phones, which now commonly include GPU and TPU processors designed to handle the enormous computational burdens associated with deep networks. In this paper, we present novel techniques for dramatically reducing the number of free parameters in deep network models with the explicit goals of (1) model compression with little or no model decompression overhead at inference time and (2) reducing the number of free parameters in arbitrary models without requiring any modifications to the architecture. We examine four techniques that build on each other and provide insight into when and how each technique operates. Accuracy as a function of free parameters is measured on two very different deep networks: ResNet and Vision Transformer. On the latter, we find that we can reduce the number of parameters by 20% with no loss in accuracy.**

***Index Terms*—Free parameters, Model compression, Deep networks, Deep network implementation**

I. Introduction

Recent research in Deep Neural Networks (DNNs) has focused on improving accuracy by increasing the size and complexity of the neural network architectures. State of the art results across a variety of tasks in domains including natural language processing, visual scene analysis, and text-to-image generation are now achieved with networks with more than 10 billion free parameters [1] [2]. Concurrently, as personal devices become more powerful and privacy and security come more into focus, there is an effort to move processing out of data centers and onto personal devices. Not only is this beneficial for data privacy, it also has the potential for lower-latency off-line results. By having DNNs operate on phones directly, all interactions, including pictures and audio recordings, never leave the device; see [3] [4] [5] [6] for a discussion of on-device computation models. The computational requirements of deep network models has become feasible on consumer-level personal devices as both GPUs and TPUs are becoming standard [7] [8]. This shifts the obstacles from on-device computation to storage of the enormous DNN models, and on the speed at which the stored model can be loaded into memory for computation.

Structured Multi-Hashing [9] provides a technique to reduce the model storage requirement for general DNN models by using a large matrix-matrix multiply to decompress the stored model for actual classification use. While [9] reported impressive model compression rates, their approach could require a significant amount of computation when the model is loaded and decompressed from persistent storage. For example, in a 10^8-parameter model, to achieve a five-fold reduction in storage size, the matrix sizes would be $10K \times 1K$, requiring 10^{11} multiplies. Since many phone applications (especially camera apps) aggressively reclaim working memory, in practice, this decoding overhead would be incurred often — as much as each time the model is used on a new image or recording.

Our work starts with [9] as inspiration. Similar to their work, our primary goal is to tackle DNN compression. We add the very pragmatic constraint of reducing, and even eliminating, the model decompression costs. Like [9], our primary goal is to achieve this improvement *without any changes to the network architecture*. Our goal is not to recommend new architectures for specific tasks, rather it is to demonstrate that it is possible to efficiently store and uncompress any given network architecture using our techniques. Throughout the remainder of this paper, we will experiment with two DNNs to quantitatively measure the effect of free parameter reduction and accuracy.

Section II briefly looks at background and related work, focusing on [9]. Section III details a sequence of four techniques, starting with the one that is closest to the matrix-matrix multiply of [9] but with significantly less computation required for the decompression stage. For each technique, we examine the advantages and disadvantages of that approach using ResNet-50 on ImageNet as our target network and application. Section IV summarizes our ResNet experimental results and then examines performance on a Vision Transformer network. Finally, Section V presents our conclusions and provides directions for future work.

II. Related Work

Though our techniques are novel, the goal of shrinking networks is not new. For example, ResNet [10], EfficientNet [11] and Vision Transformer [12] (ViT) all have architecture hyperparameters that modify the width of layers or depth of the network in an effort to tune the number of free parameters and computational cost at the expense of accuracy. Our techniques focus on the free parameters while eliminating the need to change the architecture.

We briefly provide a high-level overview of a few alternate approaches to model compression. This review is not meant to be comprehensive, rather it should provide a starting point for discovery into related approaches.

Network Pruning. Network pruning is often used as a post-training procedure in which weights, output units, channels or even entire layers are removed from the network [13] [14] [15] [16] [17] [18]. The underlying assumption for the pruning process, which mirrors our underlying assumptions as well, is that DNNs are hugely over-parameterized, for example, see [19]. The process often involves finding sets of either redundant features and weights or features that are weighted too low to have an impact on the overall performance of the network. The network is commonly fine-tuned multiple times during successive iterations of the pruning process.

Quantization. Extensive research has been done to quantize the full floating point weights of networks, as well as their activations, to lower precision. This quantization alleviates the need to store large number of bytes for each weight [20] [21] [22] [23] [24] [25]. Quantization to binary weights has been extensively studied; this can have direct beneficial consequences to power consumption. Quantization techniques can be used in combination with the network pruning described above, as well as the techniques that will be described in this paper. The quantization process has been used as both a post-training and during-training process. Many formulations of quantization rely on either an implicit or explicit clustering step to find reasonable settings for shared weight values [26] [27]. Our work will also take advantage of the ability to share weights, however, we will *first* select which weights will be clustered, and will then allow the value of that set of weights to be assigned.

Rethinking Network Architectures. Rather than viewing the model compression step as a post-training step, research has also been conducted towards starting with compact models and training them from scratch [19] [28] [29], as well as introducing additional error terms to encourage smaller network through regularization terms, see [30] for a good discussion of variants. Many automatic techniques for the creation of small and/or efficient networks have been proposed, ranging from heuristic search to parameterizing the space of networks for differentiability, and are being used in practice [31] [32] [33]. Other approaches to revising network architectures include the commonly employed broad set of network knowledge-distillation techniques [34] [35] which, viewed in this context, provides an alternate route to reduce network size.

The closest related work to ours is Structured Multi-Hashing (SMH), a framework for model compression introduced by [9]. While their framework is quite general, most of their reported results effectively treat the DNN's weights as reshaped slices from a single, large, square matrix. The square matrix is size $N_{\text{SMH}} \times N_{\text{SMH}}$ where N_{SMH}^2 is as large or (more typically) larger than the number of DNN weights. The $N_{\text{SMH}} \times N_{\text{SMH}}$ matrix is formed as part of the decompression process, as a matrix product of two rectangular matrices of dimensions $N_{\text{SMH}} \times M_{\text{SMH}}$ and $M_{\text{SMH}} \times N_{\text{SMH}}$, requiring $M_{\text{SMH}} N_{\text{SMH}}^2$ multiplies where $M_{\text{SMH}} < \frac{1}{2} N_{\text{SMH}}$. The ratio of $\frac{N_{\text{SMH}}}{2M_{\text{SMH}}}$ gives the approximate compression rate for this approach. Their results show an impressive compression rate of $10\times$ with approximately a 5% drop in accuracy using Resnet-101 [10] on ImageNet [36].

Building on the ideas and goals of SMH, we offer techniques that avoid the matrix multiplication entirely, thereby reducing decompression costs, while maintaining little or no loss in accuracy.

III. Techniques

Inspired by Structured Multi-Hashing [9], we examine a series of techniques that not only reduce the model size, they also minimize decompression times. Throughout this presentation, for clarity in reproduction, we always refer to the learned free parameters that are stored to disk as "backing weights" and the larger set of weights that are required by the DNN for computation as "working weights". Working weights are derived from backing weights.

The four techniques we examine are:

1) ***Outer Product*** - Working weights within each layer are an outer product of rank-1 matrices of the backing weights.
2) ***All Pairs*** - Working weights within each layer are products of pairs drawn from a pool of backing weights.
3) ***Layered Weight Pool*** - Working weights within each layer are randomly assigned, many-to-one, to backing weights.
4) ***Global Weight Pool*** - Working weights across the full network are drawn from a circular queue of backing weights.

Within this section, we preview a selection of results obtained on ResNet-50 using these four techniques. The specifics of the methodology and detailed results are presented in Section IV.

A. Outer Product

The strongest results achieved by [9] use the product of two rectangular matrices of backing weights to give a reduced-rank matrix containing working weights. We begin with this. In our first investigation, *Outer Product,* we use a similar approach on a per-layer basis, but simplify the matrices to rank-1, using the outer product of two vectors of length N_1 and N_2. This gives a solution that, in decompression, uses the fewest multiplies possible for a matrix product approach — approximately 1 multiply per working weight. As with [9], the shape of the working-weight matrix, $N_1 \times N_2$, does not have to conform to the shape of the layer kernel, it simply needs to be as large or larger than the total size of the layer kernel. This allows us to pick the outer dimensions of our working-weight matrix according to our desired compression ratio. For example, if K, the kernel of a dense layer,[1] is $N_{\text{in}} \times N_{\text{out}}$ and if W, our working-weight matrix is $W = W_1 W_2^T$, we can reshape

[1]The derivation is very similar for convolutional kernels with two additional dimensions for the spatial support on the layer kernel.

the working-weight matrix into a single N_1N_2-length vector, truncate that vector to length $N_{\text{in}}N_{\text{out}}$, and reshape again into the $N_{\text{in}} \times N_{\text{out}}$.

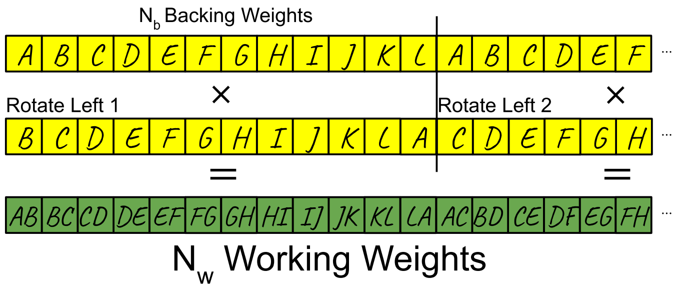
Upon first glance, it is natural to assume that we should set $N_{\text{in}} = SN_1$ and $N_2 = SN_{\text{out}}$. This special case does offer computational savings (as well as on-disk compression). We can explicitly exploit the structure of the working-weight matrix by thinking of it as a set of S separate $N_1 \times N_{\text{out}}$ weight matrices and thinking of the input channels as S sets of inputs, each of length N_1. In this case, we can multiply our S dense-layer inputs by W_1 to give S scalar values and then use those scalars to combine our S N_{out}-long pieces of W_2. This approach uses $SN_1 + N_2$ multiplies instead of the N_1N_2 that would be needed without using this structured-matrix exploitation. Unfortunately, the kernel is low rank and, experimentally, this sometimes (though not always) produces low accuracy results. In general, we need the least-common multiple of N_2 and N_{out} to be greater than $N_{\text{in}}N_{\text{out}}$ to avoid this low-rank or "striding problem".

One way to avoid the low-rank kernel while still using a rank-1 outer product to generate the working weights is to simply permute the weights that we get from W_1W_2. Permuting the entries destroys the scaling structure that originally existed between the column/row vectors of the layer's kernel. However, this weight permutation prevents us from using the structured-matrix exploit for computational savings. Without this savings, there is no advantage to having $N_2 = SN_{\text{out}}$. Further, the permutation approach is extremely slow to decode due to the very high cost of the random memory accesses needed for the permute operation.

A more efficient solution is avoid any small-integer relationship between N_2 and N_{out} and instead simply select N_2 to be co-prime with N_{out} such that $N_1N_2 \geq N_{\text{in}}N_{\text{out}}$. Using this co-prime–sized outer product approach gives an accuracy which, experimentally, is as good as permuting but decodes much faster than permutation in TensorFlow [37].

For convenience, we assume $N_1 \leq \lceil\sqrt{N_{\text{in}}N_{\text{out}}}\rceil$ and $N_2 = \lceil\frac{N_{\text{in}}N_{\text{out}}}{N_1}\rceil$. Choosing $N_1 = 1$ is degenerate: it does not reduce the number of backing weights below the number of working weights. Choosing $N_1 = 2$ can result in the striding problem since often N_{in} is a multiple of 2: in this case, N_2 will itself be a multiple of N_{out} and will violate the least-common-multiple constraint needed to avoid striding. The next larger value of $N_1 = 3$ produces a large compression ratio and results in a relatively high loss in accuracy. As shown in Figure 4, there is a drop in accuracy of 3.3% (from 76% to 72.7%) when using $N_1 = 3$.

In summary, while *Outer Product* is conceptually closest to [9] and has much lower decompression costs than that earlier work, its lowest compression ratio is quite large and, at that compression rate, there is a strong negative impact on accuracy. Next, we will remove the structural requirements imposed by a matrix-matrix multiplication.



N_b Backing Weights
A B C D E F G H I J K L A B C D E F ...
Rotate Left 1
Rotate Left 2
×
B C D E F G H I J K L A C D E F G H ...
=
AB BC CD DE EF FG GH HI IJ JK KL LA AC BD CE DF EG FH ...
N_w Working Weights

Fig. 1. All Pairs technique. Backing weights are duplicated and arranged in memory so every weight is paired with every other weight, allowing a vectorized multiply to produce N_w weights.

B. *All Pairs*

The next approach, *All Pairs*, continues to use the idea from Subsection III-A of minimizing multiplications by forming working weights from pairwise products of backing weights. In *All Pairs*, we do not impose the separation and structure that matrix-matrix multiplication imposes. Instead, we treat the backing weights as a single pool of N_b values from which we draw N_w pairs of samples, which we multiply together to form the N_w working weights.

Drawing pairs randomly suffers from the same random-memory-access slowdowns that permutation did in Subsection III-A. Instead we implement the *All Pairs* approach by first pairing weights that are 1 step apart in the backing-weights buffer, then 2 steps apart, etc. When arranged as separate buffers for the first multiplicands and the second multiplicands, backing weights are contiguous in these two buffers and the buffers can be constructed using Tensorflow concat and slice operations. These operations avoid high-cost random memory accesses. The arrangement is shown in Figure 1. The pairs of backing weights are multiplied to create working weights. There are $\frac{N_b(N_b-1)}{2}$ possible pairs.

All Pairs allows for a wider range of compression ratios than is possible with *Outer Product*, from $N_h = \frac{N_w}{N_b} = 1$ to $\sqrt{N_w/2}$. At less extreme compression ratios ($N_h << \sqrt{N_w/2}$), *All Pairs* more uniformly re-uses backing weights compared to *Outer Product*, thereby reducing the maximum number of working weights that are tied to any single backing weight. As shown in Figure 4, *All Pairs* improves the accuracy over that of *Outer Product* everywhere below 10M free parameters in ResNet-50. Next we look at a generalization of *All Pairs*.

C. *Layered Weight Pool*

Both *Outer Product* and *All Pairs* multiply pairs of backing weights to create the working weights. Our next approach is based on generalizing that concept to multiplying any m-tuples to create the working weights. We experimented with various settings of m. One might initially expect, as we did, that having a larger number of multiplies to create the working weights would help to disentangle weights and therefore improve performance. However, we were surprised to see the opposite held true.

Our finding was that setting $m = 1$, *i.e.* 1-tuples where we directly tied multiple working weights to the same single backing weight, was not only the simplest implementation but also the most effective. This is similar to Random Weight Sharing described in [38]. First, the first N_b working weights are each assigned to a distinct backing weight. Then the next N_b working weights are assigned to a random permutation of the backing weights, and this is repeated until N_w working weights are obtained (see Figure 2). By construction, this ensures uniform use of the backing weights, a property shared with *All Pairs*. Additionally, the random sampling approach (probabilistically) avoids the striding problem seen in *Outer Product*.

This effectively "ties" sets of working weights in the network. These working weights must always have the same value. This constraint seems much more stringent than that imposed by *All Pairs* or *Outer Product*, which merely ties together single multiplicands of different working weights. As can be seen in Figure 4, *Layered Weight Pool* does quite well at small compression ratios: it outperforms *All Pairs* for compression ratios less than 2.5 with no multiplications needed. Why doesn't the tying of weights hurt performance? Our hypothesis is that the DNN can mitigate the tied weights by effectively "re-organizing" dense units or convolutional channels as needed. For example, if we consider a layer in a trained network in which two output units compute two independent values that would not be possible had some weights been tied together, the network can simply move that computation to an output unit in which the weights are not tied. In a sequence of dense layers, there are many combinations of weights that produce exactly the same output and even more combinations of weights that produce different outputs with similar overall accuracy. The computation is under-constrained by the shape of the architecture. The same is true of CNNs. DNNs are able to exploit this flexibility to find a layout for the required computation in which the tied weights are least harmful.

Note that in the procedure described to this point, we have entirely eliminated the need for multiplications to construct the working weights. Nonetheless, the decompression time for the model remains slow due to the random memory access needed to map backing weights to working weights. Second, though this method performs well at lower compression rates, it underperforms *Outer Product* and *All Pairs* at compression ratios higher than 2.5. One hypothesis for this underperformance is that, when larger sets of working weights that are directly tied together (without even the freedom of the additional multiplicand of *All Pairs*), the DNN loses too much flexibility and can not find a good compromise value for the backing weight. This loss of freedom is further amplified by the tied-together sets all residing within a single layer of the DNN. We follow that line of investigation with our next technique, *Global Weight Pool*.

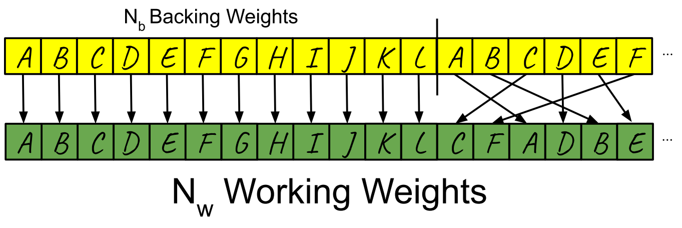

N_b Backing Weights
A B C D E F G H I J K L A B C D E F ...
A B C D E F G H I J K L C F A D B E ...
N_w Working Weights

Fig. 2. An example of the Layered Weight Pool technique using backing weights A to L.

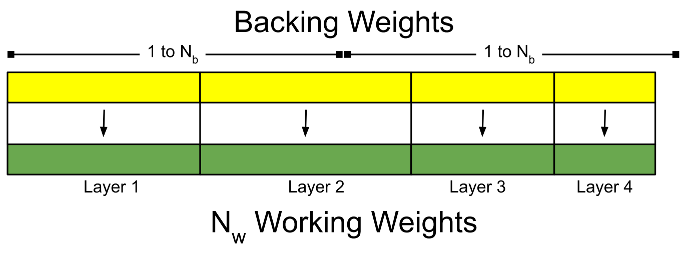


Fig. 3. In Global Weight Pool, layers draw working weights from a circular queue of ordered backing weights.

D. Global Weight Pool

In the previous approach, *Layered Weight Pool* showed promising results at low compression ratios but suffered at higher compression ratios. *Global Weight Pool* takes a similar approach to *Layered Weight Pool* but, to mitigate the accuracy loss from tying weights in the same layer, it ties weights across layers. Fortunately, this also simplifies the approach tremendously.

Global Weight Pool draws working weights in ordered slices from a global circular queue of backing weights. See Figure 3. In experiments, permuting these weights has no effect on accuracy, perhaps because their greater separation in the network makes tying weights together less problematic or perhaps because the DNN has more flexibility to mitigate tied weights in different layers. By keeping the backing weights ordered, we avoid the random memory accesses of *Layered Weight Pool*.

As can be seen in Figure 4, *Global Weight Pool* provided the best accuracy across nearly all of the compression ratios that we considered.

Our next section reviews our ResNet results in more detail, and examines the performance of *Global Weight Pool* on a transformer architecture. Finally, we will introduce *Split Pools*, a refinement of *Global Weight Pool*, which exploits the functional structure of Transformer Networks.

IV. Experiments

Results for all four approaches on ResNet-50 [10] are next. The best performing approach, *Global Weight Pool*, is then tested on the DeiT [39] variant of the Vision Transformer [12] in Section IV-B.

TABLE I
TRAINING TIME AND NUMBER OF DECOMPRESSION OPERATIONS.

Rand Mem is the number of random memory accesses during decompression.

Technique	Training (hours)	Decompression Multiplies	Decompression Rand Mem
SMH [9]		$O(N_w^{3/2})$	0
Unmodified ResNet	1.6	0	0
Outer Product	1.6	$O(N_w)$	0
Outer Product with Permute	7.2	$O(N_w)$	$O(N_w)$
All Pairs	1.6	$O(N_w)$	0
Layered Weight Pool	7.4	0	$O(N_w)$
Global Weight Pool	1.6	0	0

A. ResNet

The first set of experiments, using ResNet-50, apply the four techniques to all the convolutional layers inside bottleneck blocks and in projection shortcuts. The "stem" convolutional layer is never modified and the final dense layer is modified only in select experiments. Only convolutional and dense kernels are modified, leaving biases, batch normalization, and other learned variables unmodified. In each layer, N_b is chosen to keep $N_h = N_w/N_b$ as constant as possible across layers. For *Global Weight Pool*, the layers share a single pool with a single N_b. One of the details that must be addressed in implementing *Global Weight Pool* is the kernel initializer. In many standard implementations, the ResNet kernel initializer depends on the shape of the convolutional kernel. In *Global Weight Pool*, multiple kernels can use the same backing weight, thereby making initialization more challenging. For consistency, we use *TruncatedNormal* to initialize the backing weights for all techniques. To verify the suitability of this choice, we compared an unmodified ResNet-50 to a *Layered Weight Pool* using *TruncatedNormal* with $N_h = 1$ so that both networks had the same number of free parameters. Both networks achieved the same accuracy (76.5%).

Figure 4 shows the top-1 accuracy of each technique when varying the number of free parameters (by controlling N_b). The rightmost points use $N_h = 1$ (no compression). As expected, the rightmost points for *Layered Weight Pool* and *Global Weight Pool* are equal in both free parameters and accuracy to the unmodified ResNet (76.5%).

For insight into our experiments, Table I shows the wall time to train each technique. All of our runs use 32 TPU v3 [40] cores with a batch size of 128 on each core and train for 28,080 steps. We measure the training time and, while there can be considerable variance between runs, it is dominated by whether weights are randomly accessed in memory (over 7 hours) or can be used in place (1.6 hours).

Table I also looks at the cost of decompression, as distinct from initially loading backing weights into memory, split into the number of multiplies and the number of random memory accesses required. The techniques that access contiguous blocks of memory could be implemented as reference to existing memory with no additional accesses to the backing weights.

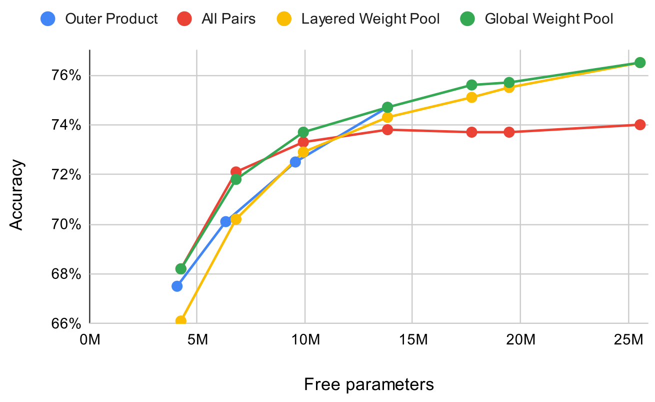


Fig. 4. Accuracy of the four techniques applied to ResNet-50. N_b, and consequently the number of free parameters, is varied.

In summary, we find that we are able to reduce the number of free parameters by a factor of $N_h = \frac{N_w}{N_b} = 1.44\times$ with an accuracy loss of only 0.9% or by $2.57\times$ with an accuracy loss of 2.8%. We review the results for each technique:

- ***Outer Product***. This is limited to compression rates of $N_h = 2$ or higher. Even in this operating region, the performance was surpassed by other techniques.
- ***All Pairs***. Though this method performs well with high compression rates, it does poorly for larger numbers of free parameters.
- ***Layered Weight Pool***. This simple free-parameter replication worked better than the potentially larger representational freedom provided by m-tuple combinations. However, this method suffers from long decompression times as random access across the backing weights can be computationally expensive.
- ***Global Weight Pool***. This method performed the best across all operating regions. The improvement over *Layered Weight Pool* suggests that tying weights in the same layer is more harmful than tying weights in different layers. In addition, it runs faster than *Layered Weight Pool* because it avoids permuting the weights.

Finally, we return to Structured Multi-Hashing [9] which provided the foundations for our study. To make our work more similar, we extended *Global Weight Pool* by modifying the final dense layer to also use the weight pool. Figure 5 compares this technique to the results presented in [9]. Not only do we achieve higher accuracy across the measured points, we incur *no extra* decompression time over an unmodified ResNet-50 since we are using *Global Weight Pool*.

B. Vision Transformers and DeiT

For more than a decade, convolutional neural networks, such as the ResNet architectures described in this paper, have been the dominant architecture for image understanding and classification tasks. The recent rapid advances in Natural Language Modeling tasks using attention-based transformer models [41]

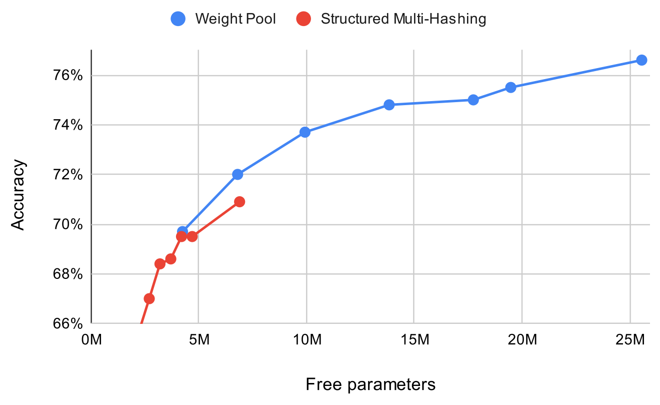


Fig. 5. ResNet-50 modified to use Global Weight Pool in both convolutional and dense layers compared to Structured Multi-Hashing [9]. A Global Weight Pool with $N_b = N_w$ gets the same accuracy (76.5%) as an unmodified ResNet-50. Both have 25.6M free parameters.

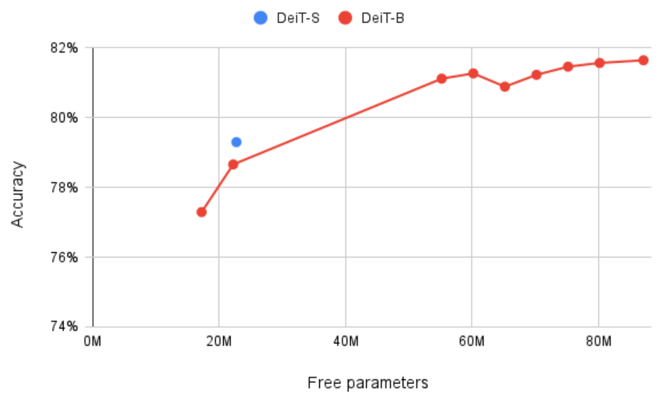


Fig. 6. DeiT-B accuracy at various numbers of free parameters. The DeiT-S point uses no compression.

[42] has led to numerous investigations into using similar attention-based models for visual tasks [43] [44] [45] [46].

Many transformer networks require immense numbers of training examples and large training times. Even when the goal is ImageNet classification, it is common to use much larger repositories of external data in training the transformer. Touvron et al. [39] presented an attention-based network that contains no convolutional layers and that achieves competitive results against the state of the art on ImageNet without the use of any external data. In their study, their networks, termed Data-Efficient Image Transformers (DeiT), were trained on a single node with 4 GPUs in three days. DeiT [39] uses the same architecture as ViT [45]: namely, a sequence of 12 transformer-encoding modules, each of which contains a multi-head attention block (with an enclosing skip connection) followed by a 2-layer dense multi-layer perceptron (MLP) (also with an enclosing skip connection). The multi-head attention blocks use 3 dense layers to project down from D dimensions to h separate subspaces (for h heads), each of dimension d: one layer each for the query (Q), key (K), and value (V) for each of the heads separately. The resulting h separate d-dimensional outputs are re-projected using another dense layer to return to the single, original D-dimension space before then feeding into the MLP.

The architectures from [39] that we use are DeiT-S (embedding dimension 384, 6 heads, 12 layers, 22.7M params) and DeiT-B (embedding dimension 768, 12 heads, 12 layers, 87.2M params).

For these experiments, we begin with *Global Weight Pool* since it produced the overall best results for ResNet. The results are shown in Figure 6; we measure the accuracy as we reduce the number of free parameters in the larger of the two models, DeiT-B. We see small reductions in accuracy for small reductions in size and we see a range of numbers of free parameters that have higher accuracy than DeiT-S.

Due to the structure within the transformer-encoding modules, we can create 6 separate weight pools, one for each of the distinct functions within the module: (1) query projection, (2) key projection, (3) value projection, (4) across-head–output combination, (5) first MLP layer, and (6) second MLP layer. Note that, unlike *Layered Weight Pool*, these pools are shared across modules (which is across layers). The pool separation is by *function* not by depth within the network. We refer to this splitting of backing weights by function as *Split Pools*. Table II shows the performance of this experiment.

TABLE II
RESULTS USING GLOBAL WEIGHT POOL AND SPLIT POOLS

Compressing DeiT-S and -B using *Split Pools* gives better performance than using a single *Global Weight Pool*.

Arch	Free Parameters	Weight Pool Accuracy	Split Pools Accuracy
DeiT-S	22,731,113	uncompressed accuracy = 79.3%	
	21,497,449	79.2%	79.2%
	17,497,449	78.1%	78.5%
	13,497,449	76.4%	76.6%
DeiT-B	87,159,017	uncompressed accuracy = 81.6%	
	80,224,361	81.6%	81.6%
	70,224,361	81.2%	81.7%
	22,224,361	78.7%	78.5%
	17,224,361	77.3%	77.0%

In summary, note that in the larger model, DeiT-B, we can reduce the number of free parameters to 70M (approximately a 20% reduction in free parameters), simply by tying their values together, *with no loss in accuracy*. Moreover, when we reduce the number of free parameters in DeiT-B to the size of DeiT-S (a compression ratio of 3.8×), accuracy is only 0.6% lower than DeiT-S even though DeiT-S was tuned to work with an appropriately sized embedding dimension. Whereas using a different pool for each layer in ResNet produced worse results, splitting the pool by function in DeiT in many cases produced better results.

V. Conclusions and Future Work

This paper has presented novel techniques to reduce the number of free parameters in large DNNs. Of the techniques, Weight Pool worked best, or Split Pools where there is a clear functional split between free parameters. Both simply tie the value of some weights together during training and thereby reduce the number of learned weights while achieving little or no loss in accuracy. This was demonstrated on two diverse architectures: ResNet and Data-Efficient Image Transformers. Absolutely no architecture changes were required to benefit from our techniques.

One important benefit of our work over the closest related successful approach presented in [9] is in the decompression process from the reduced number of backing weights to the working weights required for DNN inference. [9] and the earlier techniques mentioned in this paper require a potentially enormous matrix-matrix multiply to obtain working weights from the backing weights. For both *Global Weight Pool* and *Split Pools*, no multiplies are required for decompression. In both of these methods, by ensuring that we are careful in our use of weight-reuse so that random memory accesses are minimized – thereby incurring no extra overhead over standard networks for weight decompression.

There are numerous and varied avenues for future research.

1) All of these methods can be used in conjunction with other techniques for reducing network size, such as model distillation and model quantization. To minimize network size, all three approaches can be combined.
2) The dichotomy in performance of global-vs-non-global pools between ResNet and Transformers was pronounced. With transformers, we split the pool according to the network layer's function, whereas with ResNet it was done simply by layer. In networks in which different sections of the network have clearly defined roles, using *Split Pools* is warranted. Alternatively, analyzing the statistical characteristics of portions of trained networks may yield insight into which groups of weights will be least impacted if they are tied together. Whether such statistical characteristics occur in ResNet is an open question.
3) It will be interesting to more thoroughly understand the ramifications of these findings to network architectures and sizing. If we can reduce the number of learnable free parameters in a model, does that indicate that we should use smaller models? Will a deeper analysis of these results hint at the size of the network that will be sufficient, and even which portions of the network require more parameters by seeing where the weight-tying is most detrimental? These questions are open for future study.

References

[1] A. Ramesh, M. Pavlov, G. Goh, S. Gray, C. Voss, A. Radford, M. Chen, and I. Sutskever, "Zero-shot text-to-image generation," in *Proceedings of the 38th International Conference on Machine Learning*, ser. Proceedings of Machine Learning Research, M. Meila and T. Zhang, Eds., vol. 139. PMLR, 18–24 Jul 2021, pp. 8821–8831. [Online]. Available: https://proceedings.mlr.press/v139/ramesh21a.html

[2] A. Chowdhery, S. Narang, and e. a. Devlin, Jacob, "Palm: Scaling language modeling with pathways," 2022. [Online]. Available: https://arxiv.org/abs/2204.02311

[3] M. R. Rahimi, J. Ren, C. H. Liu, A. V. Vasilakos, and N. Venkatasubramanian, "Mobile cloud computing: A survey, state of art and future directions," *Mobile Networks and Applications*, vol. 19, no. 2, pp. 133–143, 2014.

[4] S. Malik and M. Chaturvedi, "Privacy and security in mobile cloud computing," *International Journal of Computer Applications*, vol. 80, no. 11, 2013.

[5] P. Corcoran, "Privacy in the age of the smartphone," *IEEE Potentials*, vol. 35, no. 5, pp. 30–35, 2016.

[6] M. Othman, S. A. Madani, S. U. Khan *et al.*, "A survey of mobile cloud computing application models," *IEEE communications surveys & tutorials*, vol. 16, no. 1, pp. 393–413, 2013.

[7] Apple, "Apple introduces iphone 13 and iphone 13 mini," https://www.apple.com/newsroom/2021/09/apple-introduces-iphone-13-and-iphone-13-mini/, 2021, accessed: 2022-6-26.

[8] M. Gupta, "Google tensor is a milestone for machine learning," https://blog.google/products/pixel/introducing-google-tensor/, 2021, accessed: 2022-6-26.

[9] E. Eban, Y. Movshovitz-Attias, H. Wu, M. Sandler, A. Poon, Y. Idelbayev, and M. Carreira-Perpiñán, "Structured multi-hashing for model compression," in *Computer Vision and Pattern Recognition, 2020. CVPR 2020. IEEE Conference on*. IEEE, 2020.

[10] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2016, pp. 770–778.

[11] M. Tan and Q. Le, "Efficientnet: Rethinking model scaling for convolutional neural networks," in *International Conference on Machine Learning*. PMLR, 2019, pp. 6105–6114.

[12] A. Dosovitskiy, L. Beyer, A. Kolesnikov, D. Weissenborn, X. Zhai, T. Unterthiner, M. Dehghani, M. Minderer, G. Heigold, S. Gelly, J. Uszkoreit, and N. Houlsby, "An image is worth 16x16 words: Transformers for image recognition at scale," in *ICLR: International Conference on Learning Representations*, 2021.

[13] Y. LeCun, J. Denker, and S. Solla, "Optimal brain damage," *Advances in neural information processing systems*, vol. 2, 1989.

[14] Z. Liu, M. Sun, T. Zhou, G. Huang, and T. Darrell, "Rethinking the value of network pruning," *arXiv preprint arXiv:1810.05270*, 2018.

[15] H. Peng, J. Wu, S. Chen, and J. Huang, "Collaborative channel pruning for deep networks," in *International Conference on Machine Learning*. PMLR, 2019, pp. 5113–5122.

[16] X. Dong, S. Chen, and S. Pan, "Learning to prune deep neural networks via layer-wise optimal brain surgeon," *Advances in Neural Information Processing Systems*, vol. 30, 2017.

[17] R. Yu, A. Li, C.-F. Chen, J.-H. Lai, V. I. Morariu, X. Han, M. Gao, C.-Y. Lin, and L. S. Davis, "Nisp: Pruning networks using neuron importance score propagation," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2018, pp. 9194–9203.

[18] A. Renda, J. Frankle, and M. Carbin, "Comparing rewinding and fine-tuning in neural network pruning," in *International Conference on Learning Representations*, 2020. [Online]. Available: https://openreview.net/forum?id=S1gSj0NKvB

[19] J. Hoffmann, S. Borgeaud, A. Mensch, E. Buchatskaya, T. Cai, E. Rutherford, D. d. L. Casas, L. A. Hendricks, J. Welbl, A. Clark *et al.*, "Training compute-optimal large language models," *arXiv preprint arXiv:2203.15556*, 2022.

[20] M. Nagel, M. v. Baalen, T. Blankevoort, and M. Welling, "Data-free quantization through weight equalization and bias correction," in *Proceedings of the IEEE/CVF International Conference on Computer Vision*, 2019, pp. 1325–1334.

[21] M. Courbariaux, Y. Bengio, and J.-P. David, "Binaryconnect: Training deep neural networks with binary weights during propagations," *Advances in neural information processing systems*, vol. 28, 2015.

[22] R. Andri, L. Cavigelli, D. Rossi, and L. Benini, "Yodann: An ultra-low power convolutional neural network accelerator based on binary weights," in *2016 IEEE Computer Society Annual Symposium on VLSI (ISVLSI)*. IEEE, 2016, pp. 236–241.

[23] M. Covell, D. Marwood, S. Baluja, and N. Johnston, "Table-based neural units: Fully quantizing networks for multiply-free inference," *arXiv preprint arXiv:1906.04798*, 2019.

[24] X. Lin, C. Zhao, and W. Pan, "Towards accurate binary convolutional neural network," *Advances in neural information processing systems*, vol. 30, 2017.

[25] Z. Yang, Y. Wang, K. Han, C. Xu, C. Xu, D. Tao, and C. Xu, "Searching for low-bit weights in quantized neural networks," *Advances in neural information processing systems*, vol. 33, pp. 4091–4102, 2020.

[26] J. H. Lee, J. Yun, S. J. Hwang, and E. Yang, "Cluster-promoting quantization with bit-drop for minimizing network quantization loss," in *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)*, October 2021, pp. 5370–5379.

[27] Y. Guo, "A survey on methods and theories of quantized neural networks," *arXiv preprint arXiv:1808.04752*, 2018.

[28] F. N. Iandola, S. Han, M. W. Moskewicz, K. Ashraf, W. J. Dally, and K. Keutzer, "Squeezenet: Alexnet-level accuracy with 50x fewer parameters and¡ 0.5 mb model size," *arXiv preprint arXiv:1602.07360*, 2016.

[29] A. G. Howard, M. Zhu, B. Chen, D. Kalenichenko, W. Wang, T. Weyand, M. Andreetto, and H. Adam, "Mobilenets: Efficient convolutional neural networks for mobile vision applications," *arXiv preprint arXiv:1704.04861*, 2017.

[30] H. Wang, C. Qin, Y. Zhang, and Y. Fu, "Neural pruning via growing regularization," *arXiv preprint arXiv:2012.09243*, 2020.

[31] X. He, K. Zhao, and X. Chu, "Automl: A survey of the state-of-the-art," *Knowledge-Based Systems*, vol. 212, p. 106622, 2021.

[32] J. Liang, E. Meyerson, B. Hodjat, D. Fink, K. Mutch, and R. Miikkulainen, "Evolutionary neural automl for deep learning," in *Proceedings of the Genetic and Evolutionary Computation Conference*, 2019, pp. 401–409.

[33] H. Liu, K. Simonyan, and Y. Yang, "Darts: Differentiable architecture search," *arXiv preprint arXiv:1806.09055*, 2018.

[34] G. Hinton, O. Vinyals, J. Dean *et al.*, "Distilling the knowledge in a neural network," *arXiv preprint arXiv:1503.02531*, vol. 2, no. 7, 2015.

[35] J. Gou, B. Yu, S. J. Maybank, and D. Tao, "Knowledge distillation: A survey," *International Journal of Computer Vision*, vol. 129, no. 6, pp. 1789–1819, 2021.

[36] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," *Advances in neural information processing systems*, vol. 25, pp. 1097–1105, 2012.

[37] M. Abadi, P. Barham, J. Chen, Z. Chen, A. Davis, J. Dean, M. Devin, S. Ghemawat, G. Irving, M. Isard *et al.*, "Tensorflow: A system for large-scale machine learning," in *12th USENIX Symposium*, 2016, pp. 265–283.

[38] W. Chen, J. T. Wilson, S. Tyree, K. Q. Weinberger, and Y. Chen, "Compressing neural networks with the hashing trick." in *ICML*, ser. JMLR Workshop and Conference Proceedings, F. R. Bach and D. M. Blei, Eds., vol. 37. JMLR.org, 2015, pp. 2285–2294. [Online]. Available: http://dblp.uni-trier.de/db/conf/icml/icml2015.html#ChenWTWC15

[39] H. Touvron, M. Cord, M. Douze, F. Massa, A. Sablayrolles, and H. Jegou, "Training data-efficient image transformers & distillation through attention," in *Proceedings of the 38th International Conference on Machine Learning*, ser. Proceedings of Machine Learning Research, M. Meila and T. Zhang, Eds., vol. 139. PMLR, 18–24 Jul 2021, pp. 10 347–10 357. [Online]. Available: https://proceedings.mlr.press/v139/touvron21a.html

[40] "System architecture," https://web.archive.org/web/20220606044125/https://cloud.google.com/tpu/docs/system-architecture-tpu-vm, accessed: 2022-06-16.

[41] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova, "Bert: Pre-training of deep bidirectional transformers for language understanding," *arXiv preprint arXiv:1810.04805*, 2018.

[42] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, and I. Polosukhin, "Attention is all you need," *Advances in neural information processing systems*, vol. 30, 2017.

[43] N. Carion, F. Massa, G. Synnaeve, N. Usunier, A. Kirillov, and S. Zagoruyko, "End-to-end object detection with transformers," in *European conference on computer vision*. Springer, 2020, pp. 213–229.

[44] X. Li, W. Wang, X. Hu, and J. Yang, "Selective kernel networks," in *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, 2019, pp. 510–519.

[45] A. Dosovitskiy, L. Beyer, A. Kolesnikov, D. Weissenborn, X. Zhai, T. Unterthiner, M. Dehghani, M. Minderer, G. Heigold, S. Gelly *et al.*, "An image is worth 16x16 words: Transformers for image recognition at scale," *arXiv preprint arXiv:2010.11929*, 2020.

[46] B. Wu, C. Xu, X. Dai, A. Wan, P. Zhang, Z. Yan, M. Tomizuka, J. Gonzalez, K. Keutzer, and P. Vajda, "Visual transformers: Token-based image representation and processing for computer vision," *arXiv preprint arXiv:2006.03677*, 2020.