FAST SPARSE CONVNETS

Anonymous authors
Paper under double-blind review

ABSTRACT

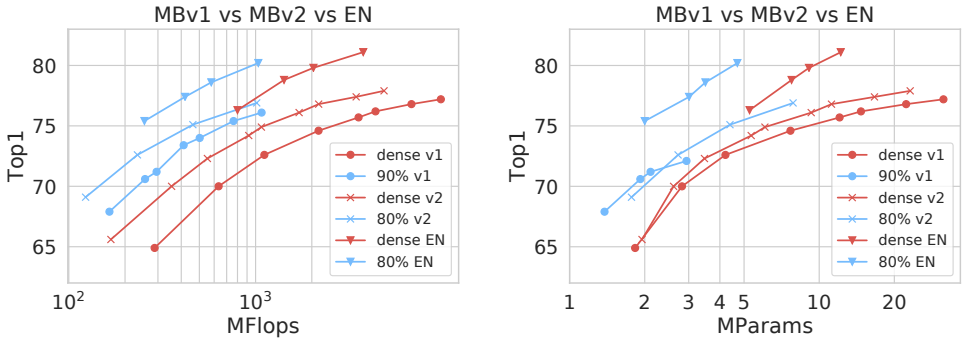
Historically, the pursuit of efficient inference has been one of the driving forces behind the research into new deep learning architectures and building blocks. Some of the recent examples include: the squeeze-and-excitation module of (Hu et al., 2018), depthwise separable convolutions in Xception (Chollet, 2017), and the inverted bottleneck in MobileNet v2 (Sandler et al., 2018). Notably, in all of these cases, the resulting building blocks enabled not only higher efficiency, but also higher accuracy, and found wide adoption in the field. In this work, we further expand the arsenal of efficient building blocks for neural network architectures; but instead of combining standard primitives (such as convolution), we advocate for the replacement of these dense primitives with their sparse counterparts. While the idea of using sparsity to decrease the parameter count is not new (Mozer & Smolensky, 1989), the conventional wisdom is that this reduction in theoretical FLOPs does not translate into real-world efficiency gains. We aim to correct this misconception by introducing a family of efficient sparse kernels for several hardware platforms, which we plan to open-source for the benefit of the community. Equipped with our efficient implementation of sparse primitives, we show that sparse versions of MobileNet v1 and MobileNet v2 architectures substantially outperform strong dense baselines on the efficiency-accuracy curve. On Snapdragon 835 our sparse networks outperform their dense equivalents by $1.1 - 2.2\times$ – equivalent to approximately one entire generation of improvement. We hope that our findings will facilitate wider adoption of sparsity as a tool for creating efficient and accurate deep learning architectures.

1 INTRODUCTION

Convolutional neural networks (CNNs) have proven to be excellent at solving a diverse range of tasks (Bhandare et al., 2016). Standard network architectures are used in classification, segmentation, object detection and generation tasks (Pan et al., 2019; Long et al., 2015; Zhao et al., 2019). Given their wide utility, there has been significant effort to design efficient architectures that are capable of being run on mobile and other low power devices while still achieving high classification accuracy on benchmarks such as ImageNet (Russakovsky et al., 2015). For example, MobileNets (Howard et al., 2017; Sandler et al., 2018) employ the depthwise separable convolutions introduced in (Sifre & Mallat, 2014) to significantly reduce resource requirements over previous architectures. Inference time and computational complexity in these architectures are dominated by the 1×1 convolutions, which directly map to matrix-matrix multiplications.

Weight sparsity is generally known to lead (Cheng et al., 2017) to theoretically smaller and more computationally efficient (in terms of number of floating-point operations) models, but it is often disregarded as a practical means of accelerating models because of the misconception that sparse operations cannot be fast enough to achieve actual speedups during inference. In this work we introduce fast kernels for Sparse Matrix-Dense Matrix Multiplication (SpMM) specifically targeted at the accceleration of sparse neural networks. The main distinction of our SpMM kernel from prior art (Nagasaka et al., 2018; Yang et al., 2018) is that we focus on a different point in the design space. While prior work focused on extremely sparse problems (typically $>$99%, found in scientific and graph problems), we target the sparsity range of 70-95%, more common when inducing weight sparsity in neural networks. As a result our kernels outperform both the Intel MKL (Intel, 2009) and the TACO compiler (Kjolstad et al., 2017).

Using these kernels, we demonstrate the effectiveness of weight sparsity across three generations of MobileNet (Howard et al., 2017; Sandler et al., 2018; Tan et al., 2018; Tan & Le, 2019) architectures.



(a) Top-1 Accuracy vs. FLOPs

(b) Top-1 Accuracy vs. Parameter Count

Figure 1: MobileNet v1 and v2 and EfficientNet models. Sparse models: blue, dense models: red. Sparse models include the cost of storing the location of non-zeros for sparse tensors as a bitmask converted back into parameter count. That is every 32 values in the bitmask contributes one "parameter".

Sparsity leads to an approximately one generation improvement in each architecture, with a sparse EfficientNet significantly more efficient than all previous models. These models represent a new generation of efficient CNNs, which reduces inference times by $1.1 - 2.2\times$, parameter counts by over $2\times$ and number of floating-point operations (FLOPs) by up to $3\times$ relative to the previous generations.

2 Related Work

Improvements in convolutional network architectures (Krizhevsky et al., 2012; Simonyan & Zisserman, 2015; He et al., 2016; Huang et al., 2017), as measured by increased classification accuracy on benchmark tasks such as ImageNet (Russakovsky et al., 2015), have generally been concomitant with increases in model parameter counts, FLOPs and memory requirements. Recently this evolution has led to networks found through neural architecture search (Zoph et al., 2017; Real et al., 2019) which can achieve over 82% top-1 accuracy, but require nearly 25 GFLOPs for one inference.

Given these prohibitive inference costs, there have been many lines of work attempting to improve CNN efficiency, which is often defined as one of three metrics:

1. Inference speedup on real hardware
2. Theoretical speedup through FLOPs reduction
3. Model size reduction

These axes are neither parallel nor orthogonal. The effect of (3) and (2) on (1) in particular can be quite complicated and highly varied depending on the hardware in question.

The MobileNet family of architectures (Howard et al., 2017; Sandler et al., 2018) has focused on improving efficiency by taking advantage of the depthwise separable convolutions introduced in (Sifre & Mallat, 2014), which can be thought of as a hand-crafted sparsification of full convolutions with a predefined sparse topology, and which are responsible for the parameter efficiency of these architectures. MobileNet v1 (MBv1) used layers of 1×1 convolutions followed by depthwise convolutions. MobileNet v2 (MBv2) introduced the inverted residual block which consists of a 1×1 convolution expanding the channel count, a depthwise convolution on the expanded channel count, and then a 1×1 convolution reducing the parameter count. Across MobileNet architectures, the depthwise convolutions account for only a small fraction of the total FLOPs, parameters, and inference time of these models. In MBv1, they account for less than 2% of the total FLOPs and in MBv2 less than 3%.

A different line of work attempted to make more efficient CNNs by directly pruning the weights of full convolutional filters accompanied by the necessary inference kernels (Park et al., 2016; Liu

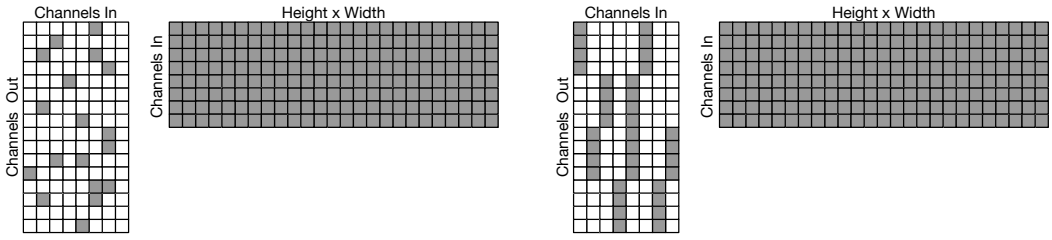


Figure 2: Sparse 1x1 Convolution as SpMM. Left: Unstructured sparsity (or block size 1). Right: Output channel block size of 4

et al., 2015). Park et al. (2016) was not able to accelerate 1×1 convolutions, Liu et al. (2015) did not attempt it. The latter also required generating a new set of kernels for each instance of a model, which is often impractical for deployment. Due to the difficultly of accelerating sparse computation, channel pruning approaches have been preferred (Gordon et al., 2018; Dai et al., 2018; Luo et al., 2017; Louizos et al., 2018; Theis et al., 2018; He et al., 2018). These approaches prune away entire filters leaving the final model dense, and function more as an architecture search over channel counts.

Full Neural Architecture Search has also been applied directly to architectures resembling MBv2 resulting in MobileNet v3 (Tan et al., 2018), FBNet (Wu et al., 2019), and EfficientNet (Tan & Le, 2019).

Alternatively, factorizations of the 1×1 convolutions have been considered in ShuffleNet (Zhang et al., 2017) and Learnable Butterfly Factorizations (Dao et al., 2019). ShuffleNet factorizes the weight matrix into a product of a permutation matrix and block diagonal matrix. Butterfly Factorizations factorize the weight matrix into a sequence of permutation matrices and weight matrices with special structure that can represent many common $O(NlogN)$ transforms such as Fast Fourier Transforms.

Work in Text-to-Speech (TTS) (Kalchbrenner et al., 2018) demonstrated that increasing sparsity and concomitant increase in state size in RNN models lead to increased model quality for a given non-zero parameter count. They additionally demonstrated fast block-sparse matrix-vector (SpMV) multiplication routines necessary for RNN inference.

3 METHODS

To understand how to design the most efficient convolutional models, we investigate both how to construct and train sparse MBv1, MBv2 and EfficientNet models and also the performance of our SpMM kernels.

3.1 SPARSIFYING NETWORKS

We train on the ImageNet (Russakovsky et al., 2015) dataset with standard augmentation and report top-1 accuracies on the provided 50k example validation set. To make the networks sparse we use the gradual magnitude pruning technique of (Zhu & Gupta, 2018).

We do not prune the first dense convolution at the beginning of all three networks. Its overall contribution to the parameter count, FLOP count, and runtime is small and does not warrant introducing a new sparse operator. Instead, we implement a dense convolutional kernel which takes as input the image in the standard HWC layout and outputs the CHW layout consumed by the sparse operators in the rest of the network. In HWC layout, the values for different channels corresponding to one spatial location are adjacent in memory. In CHW layout, the values of all the spatial locations for one channel are adjacent in memory.

We also do not prune the squeeze-excitation (Hu et al., 2018) blocks in EfficientNet as they contribute $<1\%$ of the total FLOPs to the dense model. The last fully-connected layer in all models also contributes insignificantly ($<1\%$) to the total FLOP count, but does contribute a significant fraction (20-50%) of total parameters, especially after the rest of the model is pruned. As we are concerned

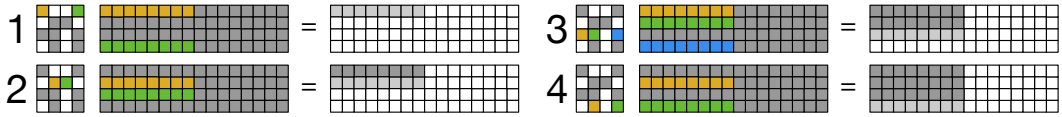


Figure 3: Visualization of the memory reads and writes of our algorithm. In step 1, we load 8 spatial locations simultaneously for each of the non-zero weights in the first row of the weight matrix. We multiply each scalar weight by its corresponding row, accumulate the results, and in the end write them out. Step 2 performs the same calculation for the next output channel. After steps 1 and 2, all values for these spatial locations are in the cache, so future loads in steps 3 and 4 will be fast, despite being random access.

with maximizing top-1 accuracy for a given runtime, we do not prune the final layer in MobileNet v1 and v2 as doing so leads to a small decrease in top-1 accuracy. Standard EfficientNets do not scale the number of filters in the last convolution by the width of the model, however we find that when introducing sparsity it is beneficial to do this; in all sparse EfficientNet models we double the units from 1280 to 2560. We also find that it is possible to make the fully-connected layer sparse without loss of accuracy in EfficientNet, so we do so.

3.2 Kernel Implementation

A diagram of the 1×1 convolution as a SpMM is seen in figure 2. Our scheme requires activation tensors be stored in CHW format, in contrast to dense mobile inference libraries (Jacob, 2017; Dukhan et al., 2019; Jacob, 2019) which favor HWC.

There are two key insights enabling the high performance of our kernels:

1. While the weight matrix is sparse, the activation matrix is dense. This means that we can perform vector loads from the activation matrix and process multiple spatial locations simultaneously.
2. By processing the matrix in the right order we can keep values that will be randomly accessed in the L1 cache, from which random access is fast and constant time.

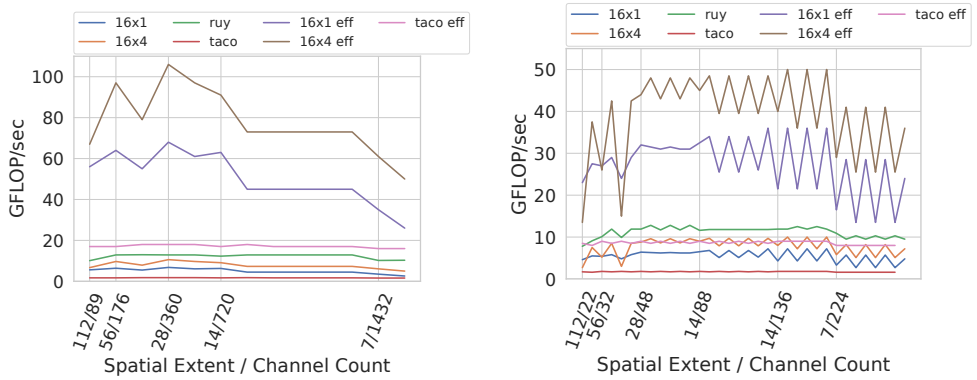
Figure 3 shows the memory read and write patterns of a few steps of the kernel. The figure shows 8 elements being processed together but other values are possible and we also implement 4 and 16. The outer loop is over columns and the inner loop is over rows; this allows each strip of 4, 8 or 16 spatial locations in the activations to remain in the L1 cache until it is no longer needed. In figure 3 steps 1 and 2 prime the cache, while subsequent steps 3 and 4 load all right hand side values from the L1 cache.

In addition to the vectorization in the HW dimension, taking advantage of small amounts of structure in the weight matrix can offer significant performance boosts by increasing data reuse after values are loaded into registers. Constraining the sparsity pattern so that multiple output or input channels all share the same zero/non-zero pattern creates 'blocks' in the weight matrix (see figure 3 right). Blocks in the output channel dimension allow for more data reuse than blocks in the input channel dimension. Experiments (see figure 6) show that either choice has the same effect on accuracy, so we implement output channel blocking with sizes of 2 and 4. Our nomenclature for kernels is to give their spatial vectorization width followed by the output channel block size – `16x2` means 16 pixels and 2 output channels are processed in the inner loop.

We implement the ARM kernels in C with NEON intrinsics unlike current production libraries (Jacob, 2017; Dukhan et al., 2019; Jacob, 2019) which rely on expert-optimized assembly. As reference, the code for the `4x1` inner loop is available in appendix A.

3.3 Library

We provide a library that can run sparse models trained with the model pruning library in TensorFlow (Abadi et al., 2015). This includes conversion from a dense representation to a Block Compressed Sparse Row (BCSR)-like representation suitable for inference. In addition to the high performance 1×1 convolutions, we also provide all supporting CHW kernels – depthwise convo-



(a) MB v1 ARM NEON

(b) MB v2 ARM NEON

Figure 4: FLOPs with increasing layer depth. All measurements taken on a Snapdragon (SD) 835.

lutions, global average pooling and a 3×3 stride-2 dense convolution – necessary for running all three generations of models. While we provide high performance versions of these kernels, we do not detail them here. They are included in end-to-end measurements.

4 RESULTS

In the main text we mainly include results for MBv1 and MBv2 due to space limitations. EfficientNets generally follow the same trends as MBv2 models, plots for EfficientNet can be found in appendix C.

First we reveal performance results for our SpMM kernels, then we show how the networks respond to sparsity and then finally we combine this information to find the models with the lowest inference time.

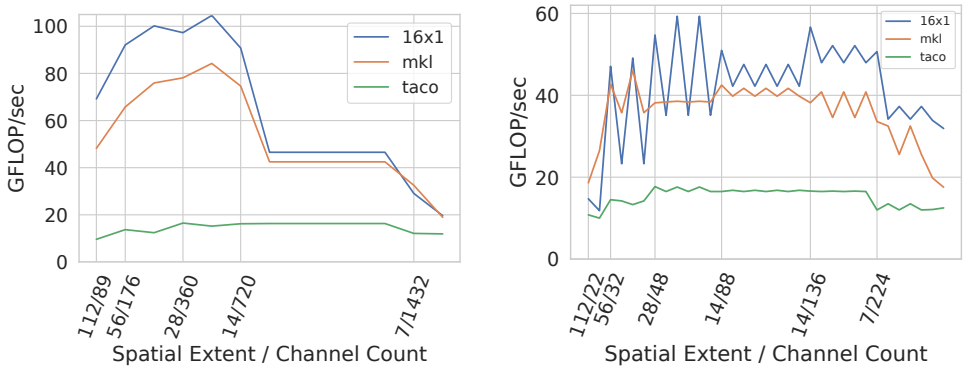
4.1 ARM KERNEL PERFORMANCE

We use Ruy (Jacob, 2019), the current TensorFlow Lite ARM64 backend written largely in hand-coded assembly, as the dense baseline. For a sparse baseline we use the kernel generated by the TACO compiler (Kjolstad et al., 2017). We present results by plotting the FLOPs achieved at each layer in the model, with increasing depth to the right in figure 4. For MBv1 we use a width multiplier of 1.4 and 90% sparse and for MBV2 we use a width multiplier of 1.4 and 80% sparse as these configurations approximately match the top-1 accuracy of the width 1 dense models. The kernel variants that process 16 spatial locations at a time (e.g. `16x1`, etc.) are the highest performing and all reported numbers are from these kernel variants. TACO only supports unstructured sparsity and should be compared with the `16x1` kernels.

The raw performance of the sparse kernels falls in the range of 40–90% of the dense kernels. And as they must do much less work, when taking the sparsity of the layer into account, the effective FLOPs are in the 2–7$\times$ range. In MBv1 performance falls significantly in the last two layers of the model when the number of channels (1024) causes the size of one "strip" of spatial locations to exceed the size of the L1 cache. In MBv2 the sawtooth pattern is caused by the alternating expand and contract operations. The performance is higher for the expand kernels due to greater data reuse of each "strip" that is brought into the L1 cache.

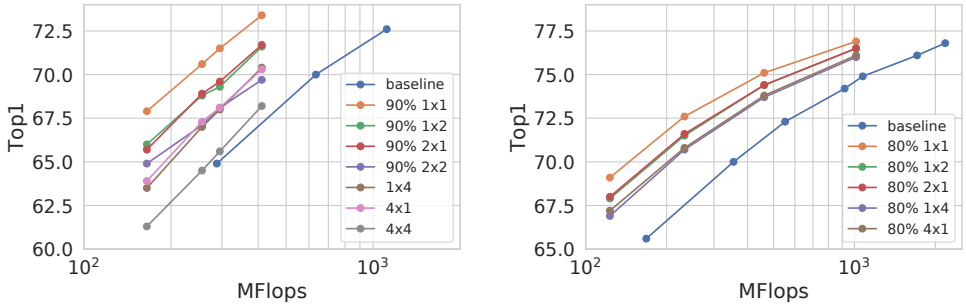
4.2 X86-64 KERNEL PERFORMANCE

We implement an AVX-512 version of our scheme with intrinsics to compare with the Intel MKL (Intel, 2009) SpMM. Results are in figure 5. In the majority of layers our scheme outperforms the MKL. The geometric mean speedup over all layers is 1.20 in both MBv1 and MBv2.



(a) MBv1 (b) MBv2

Figure 5: FLOPs with increasing layer depth. Measurements taken on an Intel Xeon W-2135.



(a) MBv1 90% Sparse (b) MBv2 80% Sparse

Figure 6: Effect of block size on top-1 accuracy. It only matters how many elements are in a block, the configuration is unimportant.

4.3 Model Performance

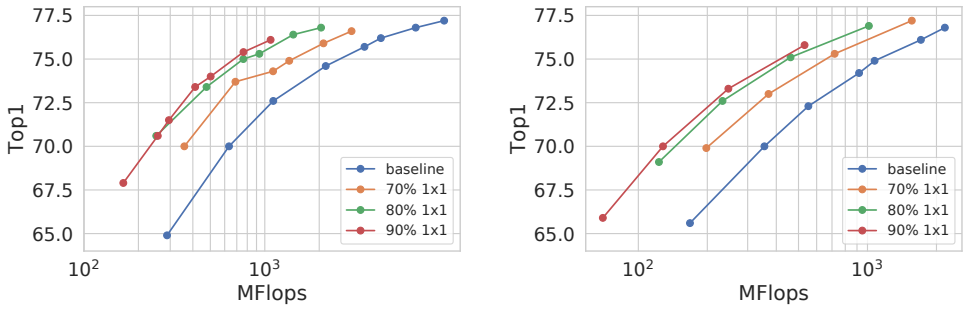
The hyper-parameters used to train MBv1 and MBv2 are listed in table 2, they were found with a grid search on dense models with a width multiplier of 1.0 to reproduce the original results, which used RMSProp, with SGD with momentum. The same hyper-parameters are used to train sparse models. This change allows us to match or exceed the reported accuracies with only 45k iterations of training.

The hyper-parameters used to train EfficientNet are largely unmodified from their code release, with the exception of extending training from 350 to 550 epochs and increasing the learning rate decay exponent to .985 from .97 so that the learning rate decays more slowly. These changes do not improve the dense baseline.

We induce sparsity in MBv1 and MBv2 by starting the sparsification process at iteration 7,000 and stopping at 28,000 with a pruning frequency of 2,000. For EfficientNet we start at iteration 23,000 and end at iteration 105,000, also with a pruning frequency of 2,000.

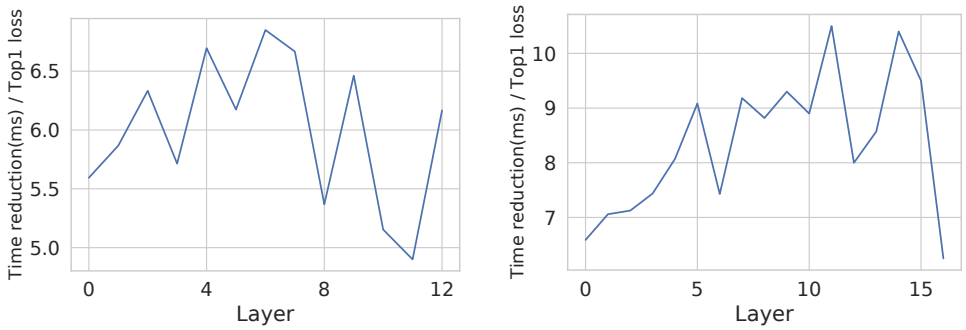
We train on the ImageNet (Russakovsky et al., 2015) dataset with standard data augmentation. Top-1 accuracies are reported on the validation set with center single-crops.

To understand the effect of block size, we plot in figure 6 accuracy against flops for different block sizes. In these plots, every sparse tensor in the network uses the same output channel block size. The tradeoff for block sparsity only appears to involve how many elements are in each block, and not their configuration. For example, in MBv1, the 1×4, 4×1 and 2×2 curves all lie on top of one another. The loss in accuracy due to blocking seems to decrease slightly for larger width models.



(a) MBv1 (b) MBv2

Figure 7: Effect of sparsity on top-1 accuracy. The sparser a model is, the fewer flops it requires to achieve a given Top-1 accuracy.



(a) MBv1 (b) MBv2

Figure 8: The x-axis corresponds to turning that layer and all following layers to block size 4, the prior layers are unstructured. The y-axis is the efficiency of making this change over an unstructured model given as a ratio where the numerator is the speedup of changing the block(s) from unstructured to block size 4 and the denominator is the decrease in top-1 accuracy that occurs by making this change.

To understand how the sparsity level affects the efficiency of the models, we train models at 70%, 80% and 90% unstructured sparsity which is constant throughout the model. The results are plotted in figure 7. MBv1 and MBv2 are more efficient the more sparse they become, confirming that the results of Kalchbrenner et al. (2018) hold not just for RNNs, but also for convolutional models as well.

In figure 1 we plot Top-1 accuracy vs. FLOPs for all three generations of sparse and dense models. MobileNet v1 is 90% sparse, the other models are 80% sparse. A sparse MBv1 exceeds MBv2 in terms of FLOP and parameter efficiency; a sparse MBv2 matches EfficientNet in terms of FLOP and parameter efficiency; and a sparse EfficientNet exceeds all other models in both categories.

4.4 Model Design

To design the models with the best top-1 accuracy vs. inference time frontiers we make the following assumptions to reduce the search space:

1. We leave the models themselves unchanged.
2. We consider only block size 1 and block size 4 variants.
3. We induce the same level of sparsity in all 1×1 convolutions.

	Model	Width	Top-1	Mega-Params	Time (ms) SD835	Time (ms) SD670
MBv1	Dense	1.0	70.9	4.24	125	106
	Sparse	1.4	**72.0**	**2.31**	**63**	**63**
MBv1	Dense	.75	68.4	2.59	73	64
	Sparse	1.0	68.4	**1.48**	**33**	**34**
MBv1	Dense	.5	63.3	1.34	36	33
	Sparse	.75	**64.4**	**1.29**	**21**	**20**
MBv2	Dense	1.4	75.0	6.06	150	129
	Sparse	2.0	74.9	**4.63**	**127**	**118**
MBv2	Dense	1.0	71.8	3.47	83	74
	Sparse	1.4	72.0	**2.86**	**63**	**56**
MBv2	Dense	.75	**69.8**	2.61	64	57
	Sparse	1.0	68.6	**1.85**	**35**	**33**
MBv2	Dense	.5	65.4	2.61	33	30
	Sparse	.75	65.2	**1.65**	**29**	**25**

Table 1: All input image sizes are 224x224. Sparse MBv1 models are 90% sparse, Sparse MBv2 models are 80% sparse. In sparse MBv1 models, layer 12 uses a block size of 4. This is almost as efficient as the models in 4.4 and matches the top-1 scores of the dense models more closely. In sparse MBv2 width multiplier 2.0 model, layers 14-16 use block size of 4. In all other MBv2 models, layers 11-16 use a block size of 4.

Then we do a search at width multiplier 1.4 over N models when there are N residual blocks in a model. An x-axis location of n corresponds to a model in which the first n residual blocks are unstructured and the last $N - n$ residual blocks have an output channel block size of 4. We train each model, note its top-1 accuracy and then measure its inference time. From this we can calculate the ratio of inference time reduction relative to a fully unstructured model and top-1 lost, which are plotted in figure 9. We choose the model with the highest ratio and train models at all widths with this choice. This amounts to making layers 6 and deeper block size 4 in MBv1 models and layers 11 and deeper block size 4 in MBv2.

A full Neural Architecture Search (Zoph & Le, 2017; Liu et al., 2019) will likely lead to even more efficient models, but we leave this to future work.

Table 1 contains the timings for running our sparse models on a single big core of two different processors, a Snapdragon 835 and a Snapdragon 670. We compare them with MBv1 and MBv2 models from their official repositories (Google, 2018a;b) run on the dense-inference TF Lite framework with the standard Ruy backend.

5 Conclusion

We demonstrate that for a constant computational budget, sparse convolutional networks are more accurate than dense ones; this corroborates the findings of Kalchbrenner et al. (2018), which demonstrated that for a set number of floating-point operations, sparse RNNs are more accurate than dense RNNs. We enable the use of weight sparsity to accelerate state-of-the-art convolutional networks by providing fast SpMM kernels along with all necessary supporting kernels for ARM processors. On Snapdragon 835 the sparse networks we present in this paper outperform their dense equivalents by $1.1 - 2.2\times$ – equivalent to approximately one entire generation of improvement. By overturning the misconception that "sparsity is slow", we hope to open new avenues of research that would previously not be considered.

A Code Listing

We present the code for the `4x1` kernel here for reference. ARM intrinsics have been renamed for clarity and casts have been removed for brevity.

Listing 1: 4x1 SpMM Kernel ARM Neon

```
size_t n = HW;
while (n != 0) { // Loop over spatial positions
 float* w;   // Weights, non-zeros are stored consecutively
 int32_t* widx_dmap; // Deltas between columns in bytes
 uint32_t* nnzmap;   // Non-zeros per row
 size_t k = OutputChannels;
 do {  // Loop over output channels
  uint32_t nnz = *nnzmap++;
  // This next line loads the bias
  float32x4_t vacc = load_1_f32_value_and_broadcast(w++);
  while (nnz-- != 0) { // Loop over non-zero input channels
   intptr_t diff = *dmap++; // get delta in bytes and advance
   float32x4_t vx = load_4_f32_values(x); // Load activations
   x += diff; // advance the activations for the next non-zero
   float32x4_t vw = load_1_f32_value_and_broadcast(w++);
   vacc = multiply_add_4_f32(vacc, vx, vw); // vacc += vx * vw
  }
  store_4_f32_values(y, vacc);
  y += HW; // advance down the output strip
 } while (--k != 0);
 // Reset pointers for the next strip of spatial locations
 y -= OutputChannels * HW; y += 4; x += 4; n -= 4;
}
```

B HYPER PARAMETERS

	MBv1	MBv2
learning rate	$.35 * 16 = 5.6$	$.24 * 16 = 3.84$
momentum	0.9	0.92
l2 coefficient	5e-5	4e-5

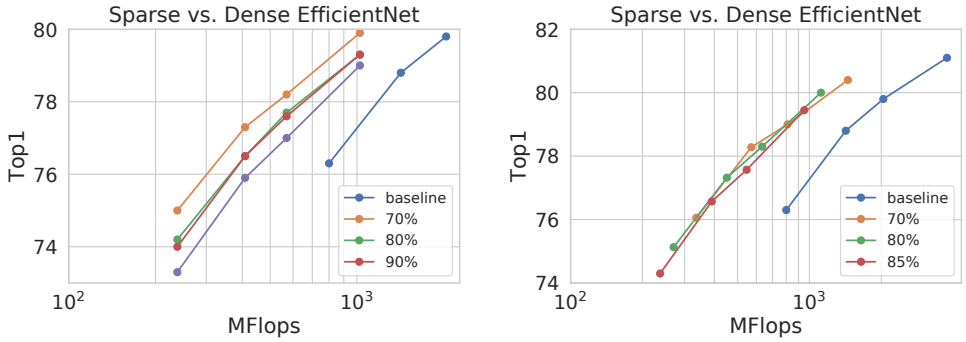
Table 2: Hyper-parameters for MBv1 and MBv2 training. Learning rates are specified in a reduced space and then multiplied by a factor of 16 due to the batch size.

C EFFICIENTNET PLOTS

Here we present the plots for EfficientNet corresponding to those in the main text for scaling with sparsity and block size. The same trend for block size is observed - the configuration of the blocks isn't important, only the total size of the block. EfficientNet exhibits less improvement as sparsity increases.

REFERENCES

Martín Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S. Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Ian Goodfellow, Andrew Harp, Geoffrey Irving, Michael Isard, Yangqing Jia, Rafal Jozefowicz, Lukasz Kaiser, Manjunath Kudlur, Josh Levenberg, Dan Mané, Rajat Monga, Sherry Moore, Derek Murray, Chris Olah, Mike Schuster, Jonathon Shlens, Benoit Steiner, Ilya Sutskever, Kunal Talwar, Paul Tucker, Vincent Vanhoucke, Vijay Vasudevan, Fernanda Viégas, Oriol Vinyals, Pete Warden, Martin Wattenberg, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng. TensorFlow: Large-scale machine learning on heterogeneous systems, 2015. URL `http://tensorflow.org/`. Software available from tensorflow.org.



(a) Block Scaling

(b) Sparsity Scaling

Figure 9: **(left)** EfficientNet scaling with block size. **(right)** EfficientNet scaling with sparsity.

Ashwin Bhandare, Maithili Bhide, Pranav Gokhale, and Rohan Chandavarkar. Applications of convolutional neural networks. *International Journal of Computer Science and Information Technologies*, 7(5):2206–2215, 2016.

Yu Cheng, Duo Wang, Pan Zhou, and Tao Zhang. A survey of model compression and acceleration for deep neural networks. *arXiv preprint arXiv:1710.09282*, 2017.

F. Chollet. Xception: Deep learning with depthwise separable convolutions. In *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 1800–1807, July 2017. doi: 10.1109/CVPR.2017.195.

Bin Dai, Chen Zhu, Baining Guo, and David Wipf. Compressing neural networks using the variational information bottleneck. In Jennifer Dy and Andreas Krause (eds.), *Proceedings of the 35th International Conference on Machine Learning*, volume 80 of *Proceedings of Machine Learning Research*, pp. 1135–1144, Stockholmsmssan, Stockholm Sweden, 10–15 Jul 2018. PMLR. URL http://proceedings.mlr.press/v80/dai18d.html.

Tri Dao, Albert Gu, Matthew Eichhorn, Atri Rudra, and Christopher Ré. Learning fast algorithms for linear transforms using butterfly factorizations. In *Proceedings of the 36th International Conference on Machine Learning, ICML 2019, 9-15 June 2019, Long Beach, California, USA*, pp. 1517–1527, 2019. URL http://proceedings.mlr.press/v97/dao19a.html.

Marat Dukhan, Yiming Wu, and Hao Lu. Qnnpack: Open source library for optimized mobile deep learning, 2019. URL https://engineering.fb.com/ml-applications/qnnpack/.

Google, 2018a. URL https://github.com/tensorflow/models/blob/master/research/slim/nets/mobilenet_v1.md.

Google, 2018b. URL https://github.com/tensorflow/models/tree/master/research/slim/nets/mobilenet.

A. Gordon, E. Eban, O. Nachum, B. Chen, H. Wu, T. Yang, and E. Choi. Morphnet: Fast simple resource-constrained structure learning of deep networks. In *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 1586–1595, June 2018. doi: 10.1109/CVPR.2018.00171.

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep Residual Learning for Image Recognition. In *2016 IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2016, Las Vegas, NV, USA, June 27-30, 2016*, pp. 770–778, 2016.

Yihui He, Ji Lin, Zhijian Liu, Hanrui Wang, Li-Jia Li, and Song Han. AMC: automl for model compression and acceleration on mobile devices. In *Computer Vision - ECCV 2018 - 15th European Conference, Munich, Germany, September 8-14, 2018, Proceedings, Part VII*, pp. 815–832, 2018.

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. *CoRR*, abs/1704.04861, 2017. URL http://arxiv.org/abs/1704.04861.

J. Hu, L. Shen, and G. Sun. Squeeze-and-excitation networks. In *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 7132–7141, June 2018. doi: 10.1109/CVPR.2018.00745.

G. Huang, Z. Liu, L. v. d. Maaten, and K. Q. Weinberger. Densely connected convolutional networks. In *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 2261–2269, July 2017. doi: 10.1109/CVPR.2017.243.

Intel. *Intel Math Kernel Library. Reference Manual*. Intel Corporation, Santa Clara, USA, 2009. ISBN 630813-054US.

Benoit Jacob. gemmlowp: a small self-contained low-precision gemm library, 2017. URL https://github.com/google/gemmlowp.

Benoit Jacob. Ruy, 2019. URL https://github.com/tensorflow/tensorflow/tree/master/tensorflow/lite/experimental/ruy.

Nal Kalchbrenner, Erich Elsen, Karen Simonyan, Seb Noury, Norman Casagrande, Edward Lockhart, Florian Stimberg, Aäron van den Oord, Sander Dieleman, and Koray Kavukcuoglu. Efficient Neural Audio Synthesis. In *Proceedings of the 35th International Conference on Machine Learning, ICML 2018, Stockholmsmässan, Stockholm, Sweden, July 10-15, 2018*, pp. 2415–2424, 2018.

Fredrik Kjolstad, Shoaib Kamil, Stephen Chou, David Lugato, and Saman Amarasinghe. The tensor algebra compiler. *Proceedings of the ACM on Programming Languages*, 1(OOPSLA):77, 2017.

Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In F. Pereira, C. J. C. Burges, L. Bottou, and K. Q. Weinberger (eds.), *Advances in Neural Information Processing Systems 25*, pp. 1097–1105. Curran Associates, Inc., 2012. URL http://papers.nips.cc/paper/4824-imagenet-classification-with-deep-convolutional-neural-networks.pdf.

Baoyuan Liu, Min Wang, Hassan Foroosh, Marshall F. Tappen, and Marianna Pensky. Sparse convolutional neural networks. *2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 806–814, 2015.

Hanxiao Liu, Karen Simonyan, and Yiming Yang. DARTS: Differentiable architecture search. In *International Conference on Learning Representations*, 2019. URL https://openreview.net/forum?id=S1eYHoC5FX.

Jonathan Long, Evan Shelhamer, and Trevor Darrell. Fully convolutional networks for semantic segmentation. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 3431–3440, 2015.

Christos Louizos, Max Welling, and Diederik P. Kingma. Learning sparse neural networks through l0 regularization. In *International Conference on Learning Representations*, 2018. URL https://openreview.net/forum?id=H1Y8hhg0b.

Jian-Hao Luo, Jianxin Wu, and Weiyao Lin. Thinet: A Filter Level Pruning Method for Deep Neural Network Compression. In *IEEE International Conference on Computer Vision, ICCV 2017, Venice, Italy, October 22-29, 2017*, pp. 5068–5076, 2017.

Michael C. Mozer and Paul Smolensky. Using relevance to reduce network size automatically. *Connection Science*, 1(1):3–16, 1989. doi: 10.1080/09540098908915626. URL https://doi.org/10.1080/09540098908915626.

Yusuke Nagasaka, Satoshi Matsuoka, Ariful Azad, and Aydın Buluç. High-performance sparse matrix-matrix products on intel knl and multicore architectures. In *Proceedings of the 47th International Conference on Parallel Processing Companion*, ICPP '18, pp. 34:1–34:10, New York, NY, USA, 2018. ACM. ISBN 978-1-4503-6523-9. doi: 10.1145/3229710.3229720. URL http://doi.acm.org/10.1145/3229710.3229720.

Z. Pan, W. Yu, X. Yi, A. Khan, F. Yuan, and Y. Zheng. Recent progress on generative adversarial networks (gans): A survey. *IEEE Access*, 7:36322–36333, 2019. doi: 10.1109/ACCESS.2019.2905015.

Jongsoo Park, Sheng R. Li, Wei Wen, Hai Li, Yiran Chen, and Pradeep Dubey. Holistic sparsecnn: Forging the trident of accuracy, speed, and size. *CoRR*, abs/1608.01409, 2016. URL http://arxiv.org/abs/1608.01409.

Esteban Real, Alok Aggarwal, Yanping Huang, and Quoc V Le. Regularized evolution for image classifier architecture search. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 33, pp. 4780–4789, 2019.

Olga Russakovsky, Jia Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, Michael Bernstein, Alexander C. Berg, and Li Fei-Fei. Imagenet large scale visual recognition challenge. *Int. J. Comput. Vision*, 115(3):211–252, December 2015. ISSN 0920-5691. doi: 10.1007/s11263-015-0816-y. URL http://dx.doi.org/10.1007/s11263-015-0816-y.

M. Sandler, A. Howard, M. Zhu, A. Zhmoginov, and L. Chen. Mobilenetv2: Inverted residuals and linear bottlenecks. In *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 4510–4520, June 2018. doi: 10.1109/CVPR.2018.00474.

Laurent Sifre and Prof Stephane Mallat. Ecole polytechnique, CMAP phd thesis rigid-motion scattering for image classification, 2014.

Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. In *International Conference on Learning Representations*, 2015.

Mingxing Tan and Quoc V. Le. Efficientnet: Rethinking model scaling for convolutional neural networks. *CoRR*, abs/1905.11946, 2019. URL http://arxiv.org/abs/1905.11946.

Mingxing Tan, Bo Chen, Ruoming Pang, Vijay Vasudevan, and Quoc V. Le. Mnasnet: Platform-aware neural architecture search for mobile. *CoRR*, abs/1807.11626, 2018. URL http://arxiv.org/abs/1807.11626.

Lucas Theis, Iryna Korshunova, Alykhan Tejani, and Ferenc Huszár. Faster gaze prediction with dense networks and Fisher pruning. *CoRR*, abs/1801.05787, 2018. URL http://arxiv.org/abs/1801.05787.

Bichen Wu, Xiaoliang Dai, Peizhao Zhang, Yanghan Wang, Fei Sun, Yiming Wu, Yuandong Tian, Peter Vajda, Yangqing Jia, and Kurt Keutzer. Fbnet: Hardware-aware efficient convnet design via differentiable neural architecture search. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2019.

Carl Yang, Aydm Buluc, and John D. Owens. Design principles for sparse matrix multiplication on the gpu. *International European Conference on Parallel and Distributed Computing - EURO-PAR*, 8 2018.

Xiangyu Zhang, Xinyu Zhou, Mengxiao Lin, and Jian Sun. Shufflenet: An extremely efficient convolutional neural network for mobile devices. *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 6848–6856, 2017.

Zhong-Qiu Zhao, Peng Zheng, Shou-tao Xu, and Xindong Wu. Object detection with deep learning: A review. *IEEE transactions on neural networks and learning systems*, 2019.

Michael Zhu and Suyog Gupta. To Prune, or Not to Prune: Exploring the Efficacy of Pruning for Model Compression. In *International Conference on Learning Representations*, 2018.

Barret Zoph and Quoc V. Le. Neural architecture search with reinforcement learning. In *5th International Conference on Learning Representations, ICLR 2017, Toulon, France, April 24-26, 2017, Conference Track Proceedings*, 2017. URL https://openreview.net/forum?id=r1Ue8Hcxg.

Barret Zoph, Vijay Vasudevan, Jonathon Shlens, and Quoc V. Le. Learning transferable architectures for scalable image recognition. *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 8697–8710, 2017.