BlockSwap: Fisher-guided Block Substitution for Network Compression

Jack Turner **Elliot J. Crowley** **Gavin Gray** **Amos Storkey** **Michael O'Boyle**
School of Informatics
University of Edinburgh
{jack.turner, elliot.j.crowley, g.d.b.gray, a.storkey}@ed.ac.uk, mob@inf.ed.ac.uk

Abstract

The desire to run neural networks on low-capacity edge devices has led to the development of a wealth of compression techniques. Moonshine (Crowley et al., 2018a) is a simple and powerful example of this: one takes a large pre-trained network and substitutes each of its convolutional blocks with a selected cheap alternative block, then distills the resultant network with the original. However, not all blocks are created equally; for a required parameter budget there may exist a potent combination of many different cheap blocks. In this work, we find these by developing *BlockSwap*: an algorithm for choosing networks with interleaved block types by passing a single minibatch of training data through randomly initialised networks and gauging their *Fisher potential*. We show that block-wise cheapening yields more accurate networks than single block-type networks across a spectrum of parameter budgets. Code is available at `https://github.com/BayesWatch/pytorch-blockswap`.

1 Introduction

In Crowley et al. (2018a), the authors propose *Moonshine*, a simple and effective strategy to compress a pre-trained deep neural network. It consists of two steps:

1. Substitute each of the original network's convolutional blocks with a **single** chosen cheap alternative block (e.g. a block introducing a bottleneck, or a block where the convolutions are grouped) to form a smaller network.
2. Treat this smaller network as a student, and train it through a distillation process—such as knowledge distillation (Ba & Caruana, 2014; Hinton et al., 2015) or attention transfer (Zagoruyko & Komodakis, 2017)—with the original network as a teacher.

The resulting networks are compact, and retain most of the performance of the teacher. Moreover, they prove more potent than networks that are simply reduced width/depth versions of the teacher. However, the range of networks considered is limited; each student network is created by first selecting a **single** cheap block, which is substituted in for every block in the original network. If we consider C different substitute blocks then this straightforwardly produces C possible student networks over a range of parameter budgets. A pictorial example of this is given in Figure 1 where each of the original blue blocks is replaced by a yellow block.

Extending this, if we allow each of the B blocks in the original network to be uniquely substituted for one of these C cheap blocks then there are C^B possible networks with mixed-and-matched blocks, some of which could prove to be better students. For the CIFAR experiments in Crowley et al. (2018a) the authors use a network consisting of 18 blocks and propose 20 substitutions; giving a total of 2.6×10^{23} potential mixed-and-matched networks. How can we tell which of these are suitable without training each one over the course of an eternity?

Preprint. denotes equal contribution.

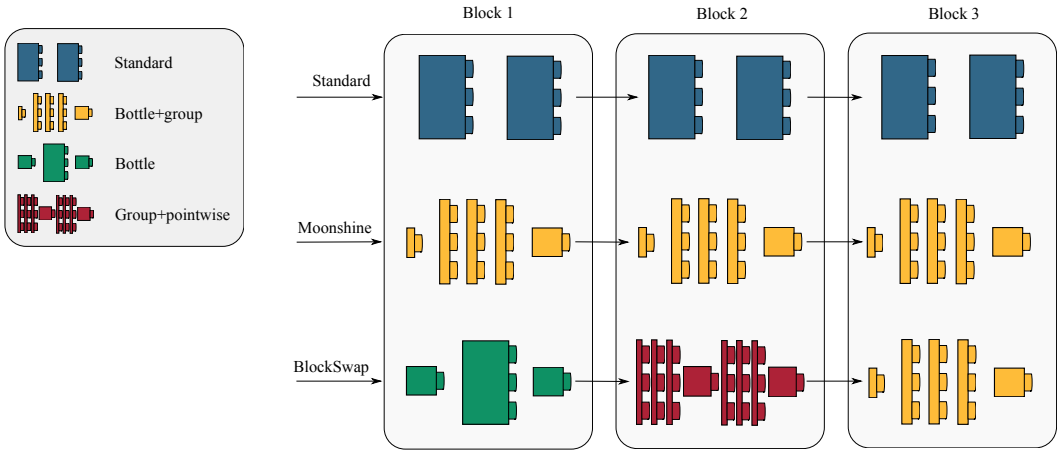

Figure 1: Neural networks typically consist of repeating connected blocks. For example in a ResNet (He et al., 2016) each block consists of two convolutional layers, and there is a skip connection between the input and output of each block. In Moonshine (Crowley et al., 2018a) a smaller network is created by replacing each of these blocks with a single cheap alternative block. In the example above, each blue standard block has been replaced with a yellow block. In this work, we posit that for a given parameter budget, there exists a network that consists of mixed-and-matched blocks that proves superior and design a method—*BlockSwap*—to find it.

Our goal in this paper is, given a desired parameter budget, to quickly identify a suitable mixed-and-matched variant of the original network that makes for a powerful student; effectively improving upon Moonshine with minimal overhead. Inspired by the one-shot search technique of Lee et al. (2019) we present a simple method—which we coin *BlockSwap*—to achieve this. First, we generate a collection of candidate mixed-and-matched architectures that satisfy a desired parameter budget. A single minibatch is then pushed through each candidate network to calculate its *Fisher potential*: the sum of the total Fisher information (Theis et al., 2018) for each of its blocks. Finally, the network with the highest potential is selected as a student and trained through distillation—specifically, attention transfer (Zagoruyko & Komodakis, 2017)—with the original teacher.

In Section 3 we review the block substitutions used in Moonshine (which we use in BlockSwap) as well as briefly describing distillation via attention transfer, and Fisher information in this context. We elucidate on our method in Section 4. Finally, we experimentally verify the potency of BlockSwap on both CIFAR-10 (Section 5) and ImageNet (Section 6).

Our contributions are as follows:

1. We introduce BlockSwap, an algorithm for reducing large neural networks by performing block-wise substitution. We show that block-wise substitution is able to produce more accurate models than both reduced depth/width models and models with a single block type.
2. We outline a simple method for quickly comparing candidate models via Fisher information. We confirm that our metric is highly correlated to final training error using an ablation study, and show that it can be used to choose performant architectures.
3. We provide thorough empirical evidence to validate that our methodology outperforms Moonshine as well as random mixed-and-matched networks.

2 Related Work

It has been established that neural networks tend to be overparameterised: in Denil et al. (2013) the authors are able to accurately predict most of the weights in a network starting with a small subset of them; Frankle & Carbin (2019) hypothesise that within a large network, there exists a fortuitously initialised subnetwork that drives its performance. In spite of this, it remains difficult to exploit this overparameterisation without taking a severe hit in performance.

One means to combat this is to use a large *teacher* network to regularise the training of a small *student* network; a process known as distillation. The small network is trained from scratch, but is also forced to match the outputs (Ba & Caruana, 2014; Hinton et al., 2015) or activation statistics (Romero et al., 2015; Zagoruyko & Komodakis, 2017; Kim et al., 2018) of the teacher using an additional loss term. When utilising distillation one must decide how to create a student network. A simple approach would be to reduce the depth of the original large network, although this can prove detrimental (Urban et al., 2017). An effective strategy is to create a student by replacing all the teacher's convolutions with grouped alternatives (Crowley et al., 2018a).

Grouped convolutions are a popular replacement for standard convolutions as they drastically cut the number of parameters used by splitting the input along the channel dimension and applying a much cheaper convolution on each split. They were originally used in AlexNet (Krizhevsky et al., 2012) due to GPU memory limitations, and have appeared in several subsequent architectures (Ioffe & Szegedy, 2015; Chollet, 2017; Xie et al., 2017; Ioannou et al., 2017; Huang et al., 2018). However, grouped convolutions are not without their disadvantages; as the number of groups increases, fewer channels are mixed, which hinders representational capacity. MobileNet (Howard et al., 2017) compensates for this by following its heavily-grouped depthwise convolutions[1] by a pointwise (1×1) convolution to allow for channel mixing. The parameter burden introduced by these pointwise convolutions is non-neglible; ShuffleNets (Zhang et al., 2018) get around this by splitting their pointwise convolutions into groups and then performing a riffle shuffle (Gilbert, 1955) along the channel dimensions.

All of the above networks required significant engineering effort. The increasing complexity of neural network designs has encouraged the development of methods for automating *neural architecture search* (NAS). Zoph & Le (2017) use an RNN to generate network descriptions and filter the options using reinforcement learning. Storing such a large quantity of possible networks is expensive, and their evaluation strategy utilised 450 GPUs over the course of 3 days. To address this, Pham et al. (2018) propose giving all models access to a shared set of weights, achieving similar performance to Zoph & Le (2017) with a single GPU in less than 24 hours. Subsequent works have made extensive use of this technique (Liu et al., 2019a; Luo et al., 2018; Chen et al., 2018). However, it has been observed that under the constrained architecture search space of the above methods, random architecture search provides a competitive baseline (Li & Talwalkar, 2019; Sciuto et al., 2019). In particular, Sciuto et al. (2019) show that weight sharing hampers the ability of candidate networks to learn and causes many NAS techniques to find suboptimal architectures.

The above NAS techniques predominantly take a bottom-up approach; they find a powerful building block, and form neural networks using stacks of these blocks. Other works have taken a top-down approach by searching for architectures using pruning (Lee et al., 2019; Liu et al., 2019b; Frankle & Carbin, 2019; Crowley et al., 2018b). SNIP (Lee et al., 2019) is a fast and effective means of doing this: one randomly initialises a large network and quantifies the sensitivity of each connection using a single minibatch. The lowest sensitivity connections are removed to produce a sparse architecture which is then trained as normal. The sensitivity metric used is similar to that used in Fisher pruning (Theis et al., 2018).

3 Preliminaries

3.1 Substitute Blocks

Here, we will briefly elaborate on the block substitutions used in Moonshine, which are by no means exhaustive. In our experiments we restrict ourselves to using the same set of blocks. In doing so, we demonstrate that it is the *combination* of blocks that is important rather than the representational capacity of a specific highly-engineered block e.g. one from the NAS literature.

The blocks considered are variations of the standard block used in residual networks. In the majority of these blocks, the input has the same number of channels as the output, so we describe their cost assuming this is the case. This standard block—denoted as S in Table 1—contains two convolutional layers, each using N lots of $N \times k \times k$ filters where k is the kernel size. Assuming the costs of batch-norm (BN) layers and shortcut convolutions (where applicable) are negligible, the block uses a total of $2N^2k^2$ parameters.

[1] A depthwise convolution is a grouped convolution where there are as many groups as input channels.

Table 1: Convolutional blocks used in this paper: a standard block S, a grouped + pointwise block G, a bottleneck block B, and a bottleneck grouped block BG. Conv refers to a $k \times k$ convolution. GConv is a grouped $k \times k$ convolution and Conv1x1 is a pointwise convolution. We assume that the input to each block has N channels and that channel size doesn't change except when written explicitly as $(x \rightarrow y)$. Where applicable, g is the number of groups and b is the bottleneck contraction. The convolutional and BN (at inference) costs are given, although the latter is significantly smaller. BN+ReLU precedes each convolution for WideResNets and follows each convolution for standard ResNets.

Block	S	$G(g)$	$B(b)$	$BG(b, g)$
Structure	Conv	GConv (g)	Conv1x1($N \rightarrow \frac{N}{b}$)	Conv1x1($N \rightarrow \frac{N}{b}$)
	Conv	Conv1x1	Conv	GConv(g)
		GConv (g)	Conv1x1($\frac{N}{b} \rightarrow N$)	Conv1x1($\frac{N}{b} \rightarrow N$)
		Conv1x1		
Conv Params	$2N^2k^2$	$2N^2(\frac{k^2}{g} + 1)$	$N^2(\frac{k^2}{b^2} + \frac{2}{b})$	$N^2(\frac{k^2}{gb^2} + \frac{2}{b})$
BN Params	$4N$	$8N$	$N(2 + \frac{4}{b})$	$N(2 + \frac{4}{b})$

Three substitute blocks are considered: grouped + pointwise blocks $G(g)$, bottleneck blocks $B(b)$, and bottleneck grouped blocks $BG(b, g)$. In $G(g)$ each of the two convolutions in the standard block is split into g groups which reduces the cost of each convolution by a factor of g. However, each convolution is now followed by a pointwise (1×1) convolution to allow for across-group channel mixing, incurring an extra cost of $2N^2$.

In bottleneck blocks $B(b)$, a pointwise convolution is used to reduce the number of channels of the input by a factor of b before a standard convolution is applied. Then, another pointwise convolution brings the channel size back up. $BG(b, g)$ is the same as this, except the middle convolution is split into g groups. These substitute blocks are summarised in Table 1 along with their parameter costs.

3.2 Distillation via Attention Transfer

Attention transfer (Zagoruyko & Komodakis, 2017) is a distillation technique whereby a student network is trained such that its *attention maps* at several distinct *attention points* are made to be similar to those produced by a large teacher network. In Crowley et al. (2018a) it was shown to consistently outperform knowledge distillation (Ba & Caruana, 2014; Hinton et al., 2015). We therefore use it as an exemplar technique in our experiments.

A formal definition of attention transfer follows: Consider a choice of layers $i = 1, 2, ..., N_L$ in a teacher network, and the corresponding layers in the student network. At each chosen layer i of the teacher network, collect the spatial map of the activations for channel j into the vector $\mathbf{a}_{ij}^t$. Let A_i^t collect $\mathbf{a}_{ij}^t$ for all j. Likewise for the student network we correspondingly collect into $\mathbf{a}_{ij}^s$ and A_i^s. Now given some choice of mapping $\mathbf{f}(A_i)$ that maps each collection of the form A_i into a vector, attention transfer involves learning the student network by minimising

$$\mathcal{L}_{AT} = \mathcal{L}_{CE} + \beta \sum_{i=1}^{N_L} \left\| \frac{\mathbf{f}(A_i^t)}{||\mathbf{f}(A_i^t)||_2} - \frac{\mathbf{f}(A_i^s)}{||\mathbf{f}(A_i^s)||_2} \right\|_2, \tag{1}$$

where β is a hyperparameter, and $\mathcal{L}_{CE}$ is the standard cross-entropy loss. In Zagoruyko & Komodakis (2017) the authors use $\mathbf{f}(A_i) = (1/N_{A_i}) \sum_{j=1}^{N_{A_i}} \mathbf{a}_{ij}^2$, where N_{A_i} is the number of channels at layer i.

3.3 Fisher Information

Theis et al. (2018) derive a second order approximation of the change in loss that would occur on the removal of a particular channel activation in a neural network. They use this signal Δ_c to identify the least important activation channels, and remove their corresponding weights while performing channel pruning. Formally, let us consider a single channel of an activation in a network due to some input minibatch of N examples. Let us denote the values for this channel as A: a $N \times W \times H$ tensor where W and H are the channel's spatial width and height. Let us refer to the entry corresponding to

example n in the mini-batch at location (i, j) as A_{nij}. If the network has a loss function $\mathcal{L}$, then we can back-propagate to get the gradient of the loss with respect to this activation channel $\frac{\partial \mathcal{L}}{\partial A}$. Let us denote this gradient as g and index it as g_{nij}. Δ_c can then be computed by

$$\Delta_c = \frac{1}{2N} \sum_{n}^{N} \left(- \sum_{i}^{W} \sum_{j}^{H} A_{nij} g_{nij} \right)^2 . \tag{2}$$

Using an approximation to the Taylor expansion of the loss around an activation originated in LeCun et al. (1989) and has inspired many works in pruning (Hassibi & Stork, 1993; Molchanov et al., 2017; Guo et al., 2016; Srinivas & Babu, 2015) and quantisation (Choi et al., 2017; Hou et al., 2017).

4 Method

Let us denote a large teacher network T composed of B blocks each of type S as $T = [S_1, S_2, ..., S_B]$. In Moonshine, a single cheap block replacement C_r is chosen from a list of candidates $C_1, C_2, ..., C_N$ of various representational capacities. A smaller model M is then constructed using **only that block**, such that $M = [C_r, C_r, ..., C_r]$. BlockSwap, by contrast, allows blocks of different representational capacities to be chosen for different stages of the network such that $M = [C_{r1}, C_{r2}, ..., C_{rB}]$.

Unfortunately, the space of possible block configurations is astronomically large ($\sim 10^{23}$ in Section 5). Even when using a cheap network evaluation strategy it is not possible to exhaustively search through the available network architectures. What we require is a method to quickly propose and score possible configurations with as little training as possible.

BlockSwap therefore relies on randomly sampling block configurations that satisfy some constraint e.g. parameter budget, as depicted in Figure 2. Such samples can then be ranked by Fisher potential to decide which architectures to train. We obtain a score for each network as follows: we begin by placing *Fisher probes* after the last convolution in each block. When a single minibatch of training data is passed through the network and the cross-entropy loss is taken, the probe measures the Fisher potential of the block by summing Δ_c (Equation 2) for each channel in the layer it is placed after. For this step, minibatch size is set equal to the size used during training. We then sum the Fisher potential of each block to give us a score for the whole network.

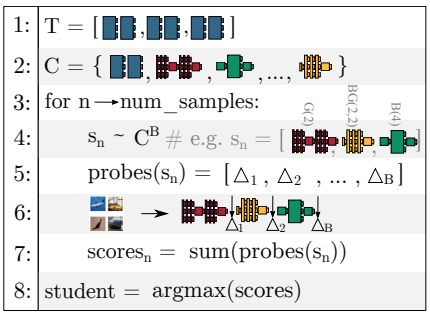

Figure 2: How BlockSwap chooses reduced architectures. In this example, the teacher T, shown on line 1, has three standard blocks. We iteratively take samples of three random blocks, push a single minibatch through (line 6), and rank based on Fisher score (line 8).

The reason for this choice of metric is straightforward: the Fisher information measures the sensitivity of the loss to each parameter at initialisation. Where parameters are insensitive at initialisation, they commonly remain insensitive as the other parameters are adjusted to adapt to the signal. Hence the accumulated Fisher potential measures the parameter wastage: how much of the parameter space is likely to be redundant for learning. We wish to choose architectures which are efficient (low parameter numbers) and have low wastage (use the parameters they have well).

Once we have scored each candidate architecture using the Fisher potential, we select the best one and train it using attention transfer from T. The training hyperparameters can be mirrored from T.

We use the following blocks from Moonshine (defined in Section 3.1) as candidate blocks:

- $B(b)$ for $b \in \{2, 4\}$
- $G(g)$ for $g \in \{2, 4, 8, 16, N/16, N/8, N/4, N/2, N\}$
- $BG(2, g)$ for $g \in \{2, 4, 8, 16, M/16, M/8, M/4, M/2, M\}$

where N is the number of channels in a block, and M is the number of channels after a bottleneck. We also use the standard block S as a candidate choice, so as to not force blocks to reduce capacity where it is imperative.

Why use a single minibatch? The choice to rank architectures after a single minibatch is not an obvious one. To verify that this is a reasonable estimator, we performed an *oracle study* where block substitution choices were limited to just S and $G(4)$. For such block choices, a WideResNet with 16 layers (and a width multiplier of 2) has 64 possible configurations. We measured the Fisher potential of all such 64 configurations after 1, 10, and 100 minibatches. We then trained each of these configurations for 200 epochs on CIFAR-10 three times to get an actual mean test error for each configuration. The Pearson correlation coefficient between Fisher potential (our proxy for error) and actual test error after a single minibatch was 0.7, but it dropped to 0.5 after 10 minibatches and 0.48 after 100 minibatches. That is to say, after a single minibatch we are able to more accurately rank the final converged test error using Fisher potential than after 10 or 100 minibatches.

Is the probe location important? Another more subtle choice is the location at which Fisher potential is measured within blocks with multiple convolutions. In our oracle example, blocks which are not bottlenecked have two convolutions (excluding pointwise convolutions) and therefore two possible locations for Fisher probing. Our oracle study showed that the effect of probe location was negligible, so for consistency, we chose to put the Fisher probe at the end of each block.

How many random samples are required? Empirically, we found that 1000 samples was enough to obtain performant networks for both the CIFAR-10 and ImageNet datasets. To consider this in more detail, take the example of BlockSwap when tasked with choosing a CIFAR-10 reduction of a WideResNet-40-2 with 400K parameters. The closest related Moonshine network is G(16), which averages 5.44% top-1 error. Taking a single random sample from the configuration space at this budget gives a network with a mean error of 5.75%. BlockSwap found a network with a mean error (after three training runs) of 5.72% after 10 samples, 5.39% after 100 samples and 5.11% after 1000 samples.

What kinds of networks does BlockSwap choose? We provide illustrations of typical network configurations from our CIFAR-10 and ImageNet experiments in Figure 4 and Figure 5 respectively. Given that BlockSwap samples block choices uniformly at random, it is perhaps unsurprising that no obvious pattern emerges. It is possible that regularity could emerge under a larger sample size.

5 CIFAR Experiments

Here, we evaluate student networks obtained using *BlockSwap* on the CIFAR-10 image classification dataset (Krizhevsky, 2009). We benchmark these against the student networks in Moonshine for a range of parameter budgets. To recapitulate, the BlockSwap networks are found by taking 1000 random samples from the space of possible block combinations that satisfy our constraint (in this case, parameter budget). These points are ranked by Fisher information after a single minibatch of training data, and the network with the highest Fisher potential is chosen.

Following the setup of Crowley et al. (2018a), a WideResNet (Zagoruyko & Komodakis, 2016) with depth 40, and width multiplier 2—WRN-40-2—is trained and used as a teacher. It consists of 18 blocks and has 2.2 million parameters. A student network is generated and is trained from scratch using attention transfer with the teacher. The following students are used:

- Naive reduced width/depth versions of the teacher: WRN-16-1,WRN-16-2, WRN-40-1
- Moonshine students — WRN-40-2 where all of its 18 S blocks are replaced with **one** of the blocks outlined in Section 4.
- *BlockSwap students* — WRN-40-2 where each of its 18 blocks are determined using BlockSwap using the blocks outlined in Section 4.

First, we train three teacher networks independently. These are used to train all of our students; each student network is trained three times, once with each of these teachers. Figure 3 shows the mean test errors of BlockSwap students at various parameter counts, compared to those of Moonshine students, which we have locally reproduced. Full results with standard deviations are listed in Table 2, along with the number of Multiply-Accumulate (MAC) operations each network uses. Some of the BlockSwap networks found are illustrated in Figure 4.

Table 2: CIFAR-10 top-1 test error for the students considered, with parameter count (in thousands, as P.(K)) and total MAC operations (in millions, as Ops(M)). D-W specifies the number of layers and the width multiplier of the student. S, $B(b)$, $G(g)$, $BG(b, g)$ represent Moonshine networks composed of a single block type. Random represents uniformly random block choices, and BlockSwap is our method. BlockSwap is able to choose the networks with the lowest mean error for all parameter budgets. The rows in bold provide a comparison point for a fixed parameter budget (∼650K). The BlockSwap student at this budget surpasses the other students, and **even the original teacher** which uses many more parameters.

D-W	Block	P. (K)	Ops (M)	Err. ($\mu \pm \sigma$)
40-2	S	2243.5	328.3	4.97 ±0.05
16-2	S	691.7	101.4	5.69 ±0.24
40-1	S	563.9	83.6	5.55 ±0.08
16-1	S	175.1	26.8	8.22 ±0.08
40-2	B(2)	431.8	64.5	5.59 ±0.12
40-2	B(4)	150.9	22.8	7.16 ±0.23
40-2	G(2)	1359.0	198.2	4.97 ±0.15
40-2	G(4)	814.6	118.6	5.14 ±0.07
40-2	**G(N/16)**	**641.3**	**133.9**	**5.14 ±0.14**
40-2	G(8)	542.5	78.7	5.08 ±0.11
40-2	G(N/8)	455.8	86.4	5.24 ±0.22
40-2	G(16)	406.4	58.8	5.44 ±0.13
40-2	G(N/4)	363.1	62.6	5.50 ±0.18
40-2	G(N/2)	316.7	50.8	5.74 ±0.12
40-2	G(N)	293.5	44.8	6.63 ±0.06
40-2	BG(2,2)	286.7	43.3	5.77 ±0.04
40-2	BG(2,M/16)	238.3	46.8	5.82 ±0.32
40-2	BG(2,4)	214.1	32.7	6.15 ±0.14

D-W	Block	P. (K)	Ops (M)	Err. ($\mu \pm \sigma$)
40-2	BG(2,M/8)	189.9	34.4	6.01 ±0.18
40-2	BG(2,8)	177.8	27.3	6.04 ±0.15
40-2	BG(2,M/4)	165.7	28.2	6.03 ±0.04
40-2	BG(2,16)	159.7	24.7	6.23 ±0.30
40-2	BG(2,M/2)	153.6	25.1	6.32 ±0.12
40-2	Random	173.4	34.8	6.19 ±0.14
40-2	Random	244.3	52.1	6.43 ±0.21
40-2	Random	397.4	59.5	5.75 ±0.14
40-2	**Random**	**641.4**	**96.6**	**5.57 ±0.08**
40-2	Random	973.9	129.4	5.07 ±0.05
40-2	BlockSwap	178.3	34.5	6.01 ±0.14
40-2	BlockSwap	218.6	46.5	6.06 ±0.07
40-2	BlockSwap	252.2	55.9	5.48 ±0.05
40-2	BlockSwap	322.2	58.2	5.56 ±0.06
40-2	BlockSwap	407.5	54.2	5.11 ±0.22
40-2	BlockSwap	544.9	82.2	5.01 ±0.18
40-2	**BlockSwap**	**657.9**	**97.9**	**4.84 ±0.16**
40-2	BlockSwap	797.0	86.1	5.27 ±0.11
40-2	BlockSwap	962.4	99.2	5.04 ±0.06

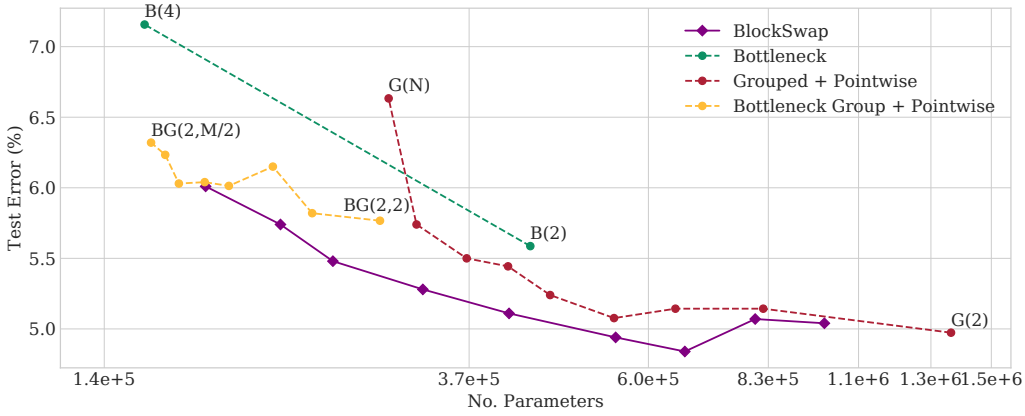

Figure 3: CIFAR-10 top-1 test error of students versus parameters. We locally reproduce the Moonshine networks and compare against BlockSwap. Each point represents the mean of three runs, where every network is trained with the same three teachers. Note that BlockSwap outperforms Moonshine across a spectrum of parameter budgets.

Our results confirm the observation of Crowley et al. (2018a), that block cheapening is more effective than using smaller architectures. We further show that BlockSwap chooses better performing networks than those from Moonshine across the full spectrum of available parameter budgets, with interleaved block types giving lower error rates than the use of a single block-type throughout. For example, at a budget of 650K parameters the Moonshine network ($G(N/16)$) achieves 5.14% mean error, whereas

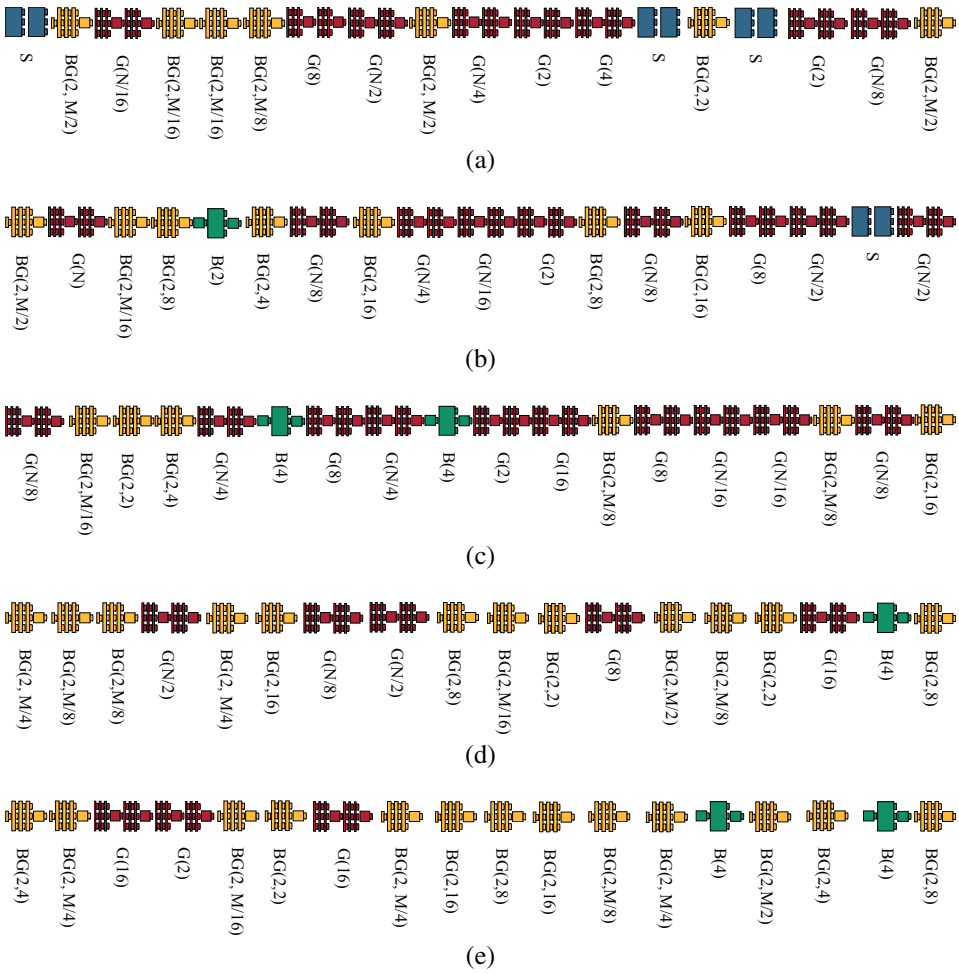

Figure 4: WRN-40-2 block substitutions chosen by BlockSwap for parameter budgets (a) 950K, (b) 650K, (c) 400K, (d) 250K, and (e) 150K on CIFAR-10. Errors are given in Table 2.

BlockSwap chooses a block configuration with a mean error of 4.84%. Note that the mean error of the teacher is 4.97%; **BlockSwap chooses a student that is better than the teacher, despite using 3.5$\times$ fewer parameters and 3$\times$ fewer MACs**.

Given the undeniable strength of random baselines in architecture search settings (Liu et al., 2019a; Sciuto et al., 2019; Li & Talwalkar, 2019), we additionally compare BlockSwap against randomly generated block configurations. It is of course possible that interleaving block types is simply more powerful than using a single block type, accounting for BlockSwap's success. However, Table 2 shows that BlockSwap consistently outperforms random configurations, proving that our aggregated Fisher potential metric is an effective means for choosing optimal block structures.

Implementation Details: Networks are trained for 200 epochs using SGD with momentum 0.9. The initial learning rate of 0.1 is reduced by a factor of 5 every 60 epochs. Minibatches of size 128 are used with standard crop + flip data augmentation. The weight decay factor is set to 0.0005. For attention transfer β is set to 1000 using the output of each of the three sections of the network.

6 ImageNet Experiments

Here, we demonstrate that students chosen by BlockSwap succeed on the more challenging ImageNet dataset (Russakovsky et al., 2015). We use a pretrained ResNet-34 (16 blocks) as a teacher, and

compare students at two parameter budgets (3M and 8M). We train a Blockswap student at each of these budgets and compare their validation errors to those of:

- simple reduced depth/width students: ResNet-18 and a ResNet-18 where the channel width of its last three sections has been halved (ResNet-18-0.5).
- Moonshine students: ResNet-34-G(4)—i.e. a ResNet-34 where each block has been replaced by G(4)—and ResNet-34-G(N)

The student networks found by BlockSwap for these two budgets are illustrated in Figure 5. Top-1 and top-5 validation errors are presented in Table 3.

Consider the students at the 3M mark: ResNet-18-0.5 has a top-1 error of 37.20% and a top-5 error of 15.02% when trained with attention transfer from the teacher network. As in Section 5, producing a student by naively substituting all blocks in the teacher for a single option—in this case $G(N)$—results in a more accurate network (30.16% top-1 error, 10.66% top-5 error) with a similar number of parameters. BlockSwap is able to reduce this error further to 29.57% top-1 and 10.20% top-5. It is worth noting that the BlockSwap network uses more MAC operations than ResNet-34-G(N). This is because there were no MAC constraints placed on the BlockSwap search space.

At 8M parameters our BlockSwap model achieves a top-1 error of 26.24% and top-5 error of 7.75%. It surpasses its comparators *and the teacher* by quite a margin. Specifically, it beats the teacher by 0.49% in top-1 error and 0.82% in top-5 error despite using almost $3\times$ fewer parameters. The phenomenon of students beating their teachers has been previously noted by Furlanello et al. (2018).

Implementation Details: Networks are trained with a cross-entropy loss for 100 epochs using SGD with momentum 0.9. The initial learning rate of 0.1 is reduced by $10\times$ every 30 epochs. Minibatches of size 256 are used with standard crop + flip augmentation. The weight decay factor is set to 0.0001. For attention transfer β is set to 750 using the output of each of the four sections of network.

Table 3: Top-1 and Top-5 classification errors (%) on the validation set of ImageNet for students trained with attention transfer from a ResNet-34. We can see that for a similar number of parameters, the student found from BlockSwap outperforms its counterparts, and in one instance, the teacher.

Model	Params	MACs	Top-1 Error(%)	Top-5 Error(%)
ResNet-34 **Teacher**	21.8M	3.669G	26.73	8.57
ResNet-18	11.7M	1.818G	29.18	10.05
ResNet-34-G(4)	8.1M	1.395G	26.58	8.43
BlockSwap	8.1M	1.242G	**26.24**	**7.75**
ResNet-18-0.5	3.2M	909M	37.20	15.02
ResNet-34-G(N)	3.1M	559M	30.16	10.66
BlockSwap	3.1M	812M	**29.57**	**10.20**

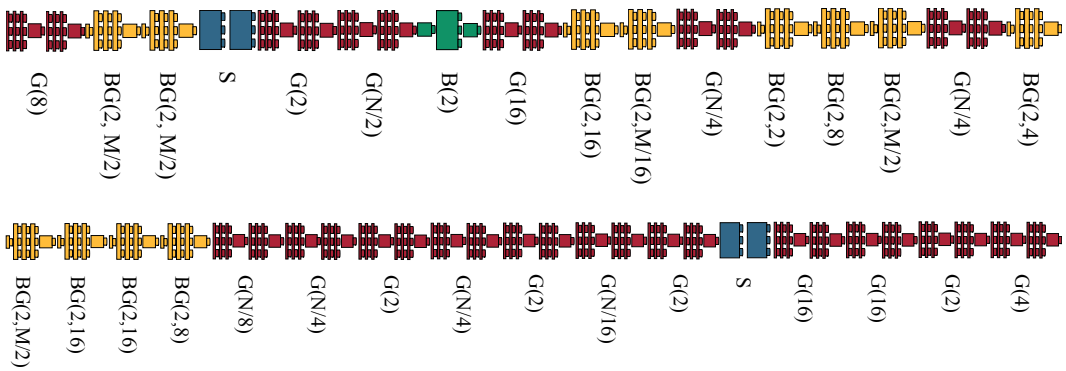

Figure 5: ResNet-34 block substitutions chosen by BlockSwap at 3M (top) and 8M (bottom) parameters on ImageNet.

7 Conclusion

We have developed a simple algorithm for reducing large neural networks to suit very flexible targets based on block substitution. These reduced networks make for excellent students. We have shown that our algorithm outperforms both small standard networks and fixed block type networks. However, we have only considered a simple, non-exhaustive selection of blocks. Future work could mix-and-match powerful blocks obtained through neural architecture search (Liu et al., 2019a; Zoph et al., 2018) as well as selecting networks based on inference time, or energy cost instead of parameter count.

Acknowledgements. This work was supported in part by the EPSRC Centre for Doctoral Training in Pervasive Parallelism and a Huawei DDMPLab Innovation Research Grant, as well as funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No. 732204 (Bonseyes). This work is supported by the Swiss State Secretariat for Education, Research and Innovation (SERI) under contract number 16.0159. The opinions expressed and arguments employed herein do not necessarily reflect the official views of these funding bodies. The authors are grateful to Joseph Mellor, David Sterratt, Paul Micaelli, and Luke Darlow for their helpful suggestions.

References

Ba, L. J. and Caruana, R. Do deep nets really need to be deep? In *Advances in Neural Information Processing Systems*, 2014.

Chen, L.-C., Collins, M., Zhu, Y., Papandreou, G., Zoph, B., Schroff, F., Adam, H., and Shlens, J. Searching for efficient multi-scale architectures for dense image prediction. In *Advances in Neural Information Processing Systems*, 2018.

Choi, Y., El-Khamy, M., and Lee, J. Towards the limit of network quantization. In *International Conference on Learning Representations*, 2017.

Chollet, F. Xception: Deep learning with depthwise separable convolutions. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2017.

Crowley, E. J., Gray, G., and Storkey, A. Moonshine: Distilling with cheap convolutions. In *Advances in Neural Information Processing Systems*, 2018a.

Crowley, E. J., Turner, J., Storkey, A., and O'Boyle, M. A closer look at structured pruning for neural network compression. *arXiv preprint arXiv:1810.04622*, 2018b.

Denil, M., Shakibi, B., Dinh, L., Marc'Aurelio, R., and de Freitas, N. Predicting parameters in deep learning. In *Advances in Neural Information Processing Systems*, 2013.

Frankle, J. and Carbin, M. The lottery ticket hypothesis: Finding sparse, trainable neural networks. In *International Conference on Learning Representations*, 2019.

Furlanello, T., Lipton, Z. C., Tschannen, M., Itti, L., and Anandkumar, A. Born again neural networks. In *International Conference on Machine Learning*, 2018.

Gilbert, E. Theory of shuffling. Technical report, Bell Labs, Murray Hill, New Jersey, U.S., 1955.

Guo, Y., Yao, A., and Chen, Y. Dynamic network surgery for efficient DNNs. In *Advances in Neural Information Processing Systems*, 2016.

Hassibi, B. and Stork, D. G. Second order derivatives for network pruning: Optimal brain surgeon. In *Advances in Neural Information Processing Systems*, 1993.

He, K., Zhang, X., Ren, S., and Sun, J. Deep residual learning for image recognition. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2016.

Hinton, G., Vinyals, O., and Dean, J. Distilling the knowledge in a neural network. *arXiv preprint arXiv:1503.02531*, 2015.

Hou, L., Yao, Q., and Kwok, J. T. Loss-aware binarization of deep networks. In *International Conference on Learning Representations*, 2017.

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

Huang, G., Liu, S., van der Maaten, L., and Weinberger, K. Q. CondenseNet: An efficient densenet using learned group convolutions. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2018.

Ioannou, Y., Robertson, D., Cipolla, R., and Criminisi, A. Deep roots: Improving CNN efficiency with hierarchical filter groups. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2017.

Ioffe, S. and Szegedy, C. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *International Conference on Machine Learning*, 2015.

Kim, J., Park, S., and Kwak, N. Paraphrasing complex network: Network compression via factor transfer. In *Advances in Neural Information Processing Systems*, 2018.

Krizhevsky, A. Learning multiple layers of features from tiny images. Master's thesis, University of Toronto, 2009.

Krizhevsky, A., Sutskever, I., and Hinton, G. ImageNet classification with deep convolutional neural networks. In *Advances in Neural Information Processing Systems*, 2012.

LeCun, Y., Denker, J. S., and Solla, S. A. Optimal brain damage. In *Advances in Neural Information Processing Systems*, 1989.

Lee, N., Ajanthan, T., and Torr, P. H. SNIP: Single-shot network pruning based on connection sensitivity. In *International Conference on Learning Representations*, 2019.

Li, L. and Talwalkar, A. Random search and reproducibility for neural architecture search. *arXiv preprint arXiv:1902.07638*, 2019.

Liu, H., Simonyan, K., and Yang, Y. DARTS: Differentiable architecture search. In *International Conference on Learning Representations*, 2019a.

Liu, Z., Sun, M., Zhou, T., Huang, G., and Darrell, T. Rethinking the value of network pruning. In *International Conference on Learning Representations*, 2019b.

Luo, R., Tian, F., Qin, T., Chen, E., and Liu, T.-Y. Neural architecture optimization. In *Advances in Neural Information Processing Systems*, 2018.

Molchanov, P., Tyree, S., Karras, T., Aila, T., and Kautz, J. Pruning convolutional neural networks for resource efficient inference. In *International Conference on Learning Representations*, 2017.

Pham, H., Guan, M. Y., Zoph, B., Le, Q. V., and Dean, J. Efficient neural architecture search via parameter sharing. *arXiv preprint arXiv:1802.03268*, 2018.

Romero, A., Ballas, N., Kahou, S. E., Chassang, A., Gatta, C., and Bengio, Y. Fitnets: Hints for thin deep nets. In *International Conference on Learning Representations*, 2015.

Russakovsky, O., Deng, J., Su, H., Krause, J., Satheesh, S., Ma, S., Huang, Z., Karpathy, A., Khosla, A., Bernstein, M., Berg, A. C., and Fei-Fei, L. ImageNet large scale visual recognition challenge. *Int. Journal of Computer Vision (IJCV)*, 115(3):211–252, 2015. doi: 10.1007/s11263-015-0816-y.

Sciuto, C., Yu, K., Jaggi, M., Musat, C., and Salzmann, M. Evaluating the search phase of neural architecture search. *arXiv preprint arXiv:1902.08142*, 2019.

Srinivas, S. and Babu, R. V. Data-free parameter pruning for deep neural networks. In *British Machine Vision Conference*, 2015.

Theis, L., Korshunova, I., Tejani, A., and Huszár, F. Faster gaze prediction with dense networks and Fisher pruning. *arXiv preprint arXiv:1801.05787*, 2018.

Urban, G., Geras, K. J., Kahou, S. E., Aslan, O., Wang, S., Caruana, R., Mohamed, A., Philipose, M., and Richardson, M. Do deep convolutional nets really need to be deep and convolutional? In *International Conference on Learning Representations*, 2017.

Xie, S., Girshick, R., Dollár, P., Tu, Z., and He, K. Aggregated residual transformations for deep neural networks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2017.

Zagoruyko, S. and Komodakis, N. Wide residual networks. In *British Machine Vision Conference*, 2016.

Zagoruyko, S. and Komodakis, N. Paying more attention to attention: Improving the performance of convolutional neural networks via attention transfer. In *International Conference on Learning Representations*, 2017.

Zhang, X., Zhou, X., Lin, M., and Sun, J. ShuffleNet: An extremely efficient convolutional neural network for mobile devices. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2018.

Zoph, B. and Le, Q. V. Neural architecture search with reinforcement learning. In *International Conference on Learning Representations*, 2017.

Zoph, B., Vasudevan, V., Shlens, J., and Le, Q. V. Learning transferable architectures for scalable image recognition. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2018.