HAKD: Hardware Aware Knowledge Distillation

Jack Turner*  
School of Informatics  
University of Edinburgh  
jack.turner@ed.ac.uk

Elliot J. Crowley*  
School of Informatics  
University of Edinburgh  
elliot.j.crowley@ed.ac.uk

Valentin Radu  
School of Informatics  
University of Edinburgh  
valentin.radu@ed.ac.uk

José Cano  
School of Computing Science  
University of Glasgow  
jose.canoreyes@glasgow.ac.uk

Amos Storkey  
School of Informatics  
University of Edinburgh  
a.storkey@ed.ac.uk

Michael O’Boyle  
School of Informatics  
University of Edinburgh  
mob@inf.ed.ac.uk

Abstract

Despite recent developments, deploying deep neural networks on resource constrained general purpose hardware remains a significant challenge. There has been much work in developing methods for reshaping neural networks, usually with a focus on minimising total parameter count. These methods are typically developed in a hardware-agnostic manner and do not exploit hardware behaviour. In this paper we propose a new approach, Hardware Aware Knowledge Distillation (HAKD) which uses empirical observations of hardware behaviour to design efficient student networks which are then trained with knowledge distillation. This allows the trade-off between accuracy and performance to be managed explicitly. We have applied this approach across three platforms and evaluated it on two networks, MobileNet and DenseNet, on CIFAR-10. We show that HAKD outperforms Deep Compression and Fisher pruning in terms of size, accuracy and performance.

1 Introduction

Deep neural networks (DNNs) have recently experienced a surge in popularity due to their exceptional performance on a range of tasks [1], such as image classification, language translation, and speech recognition. This is partly due to the availability of GPU clusters which provide the computational power that DNNs typically demand. Such demands present a significant barrier to the adoption of DNNs in embedded edge devices (e.g. mobile phones), where their most interesting applications lie [2]. Given such large workloads, it is important to utilise any available hardware efficiently when deploying DNNs on resource constrained devices. The most prevalent solution to this problem is to reduce the parameter count, and hence the number of weights stored in the DNN, which reduces its memory footprint.

It is well established that significant redundancy exists in many network architectures [3], and that often up to 90% of parameters, or weights, stored in the network can be removed with minimal effect on accuracy. A variety of frameworks have since been developed for removing redundant weights systematically to accelerate inference. These promising techniques may allow us to deploy highly accurate neural networks with fast inference times, opening the door to a wide range of potential applications.
One such compression technique which has proved exceptionally popular is Deep Compression \cite{4}, a weight pruning method that induces very sparse weight matrices for inference acceleration on low power devices. Deep Compression was later extended in Scalpel \cite{5} to utilise device-specific parallelism; for example, when deploying on CPUs, Scalpel prunes weights into groups that match the width of the SIMD lanes on the device. By properly exploiting device-level parallelism, Scalpel is able to give up to 2× speedup over the device-agnostic methodology of Deep Compression.

However, recent work has shown that reducing network size while keeping weight matrices dense performs better than introducing weight sparsity on resource-constrained devices \cite{6}, exposing channel pruning methods (e.g. Fisher pruning \cite{7}) as a preferable means of compression.

Although channel pruning is effective, hardware structure has a significant impact on its performance. In this paper, we propose a novel channel pruning approach that uses hardware behaviour to reshape networks for accelerating inferences. We use knowledge distillation \cite{8, 9} to train students that are hardware-aware reductions of their original teacher models; we call such an approach Hardware Aware Knowledge Distillation (HAKD). HAKD achieves between 1.1× and 7× speedup over the original models and up to 3× speedup over Deep Compression while also improving performance and accuracy relative to other pruning methods.

HAKD provides the ability to optimise for different objectives. For instance, in some cases we may be willing to sacrifice a small amount of accuracy for a large inference time speedup, while in other scenarios, maximal accuracy for a given inference time may be critical. We combine insights from developments in neural compression techniques and computer architecture to develop a new methodology for reshaping neural networks to fit specific hardware platforms, tasks, and deployment priorities.

The contributions of this paper are as follows:

- A novel device-aware channel optimisation approach based on empirical execution time, incorporating knowledge distillation.
- Evaluation and comparison against popular pruning techniques on CIFAR-10 classification for MobileNet and DenseNet architectures.
- Evaluation on resource-constrained hardware (Nvidia Jetson Tx2 ARM v7), showing significant performance improvement.

2 Motivation

At the heart of our approach is the need to guide channel pruning based on actual hardware behaviour. To illustrate this, consider the graph in Figure\ref{fig:1} It shows the inference time on an Intel Core i7 of a single layer of a ResNet-50 \cite{10} network pruned to different layer widths (number of channels). With a larger number of channels, we have more weights and would expect greater accuracy, but as the figure shows, the associated extra computation increases execution time. However, as the figure also shows, the relationship between the number of channels and inference time is not linear. In fact a staircase pattern appears, where weight matrices stop utilising memory efficiently. The figure highlights the optimal points on each step of the staircase. These are the points that allow for the maximum number of channels (and hence accuracy) for a given inference time budget.

Our approach is to first apply channel pruning to reach a step of the staircase, and then learn a new model with the number of channels set to the optimal point on that step based on knowledge distillation. This is described in greater detail in section 4.
Figure 1: Inference time for a layer of a ResNet-50 vs. the number of channels in that layer on an Intel i7. Notice the staircase pattern that emerges. For a given inference time, it is preferable to pick the points in green to maximise the network’s capacity.

Figure 2: Student networks are often produced by reducing the number of filters per layer (width) and by reducing the number of layers (depth) starting from the structure of a teacher network. In this paper, we only consider width reduction.

3 Related Work

3.1 Neural Compression Techniques

3.1.1 Weight Pruning

Weight pruning is a popular technique for reducing the parameter count in neural networks, and has been shown to reduce overfitting in some cases [11]. Weight pruning techniques generally fall into two categories: (i) unstructured removal of weights deemed unnecessary and (ii) structured regularisation — pushing the network weights towards a regular sparse format. [12] first proposed the removal of redundant weights by penalising the error function with a count of the sum of parameters being optimised. Minimising this error function thus jointly optimises accuracy and parameter count, analogous to finding the minimum description length of the network.

In Deep Compression [13], the authors promote an iterative process which separates the optimisation of parameter count and accuracy. In order to learn a minimal weight structure, they propose cycling between steps of traditional training and single steps where low magnitude weights are removed. This is applied layer-by-layer, with the threshold for removal tuned by the standard deviation of the layer. The authors show that it is possible to prune up to 90% of the weights without affecting
accuracy. In [4], this is developed using a three stage method for storing the network involving pruning, quantisation, and Huffman coding.

Deep Compression relies on the assumption that low magnitude weights are unimportant. However, the validity of this assumption has recently been challenged [14] and it has been shown that these small weights often carry more importance than their magnitude suggests. The large FLOP saving Deep Compression provides is also often not leveraged into speedup because of the sparse format the weight matrices are left in [5, 6]. Moreover, the process of iteratively pruning and retraining is very expensive.

Gaussian dropout [15] is a form of dropout that adds noise sampled from a random distribution instead of using a binary mask. In [16], the authors show that Gaussian dropout can be modelled as a form of stochastic gradient variational Bayes using a specific prior and variational posterior over the weights. This means that they can optimise a dropout rate for each individual weight using the variational lower bound. This was extended in [17], where a new method for approximating the KL divergence of the prior and posterior distribution was exploited to remove weights from the network. The new approximation allows for the noise variance to be unbounded, and in such cases that the noise variance for a parameter tends to infinity, the parameter is removed. This yields impressive results and was shown to reduce the parameters up to $280 \times$ on LeNet [18] architectures, but has yet to be extended to larger models.

### 3.1.2 Channel Pruning

A natural extension of weight pruning is to not only remove individual weights in a network, but entire channels from the convolutional layers and nodes from the fully connected layers. A common approach to this is to apply Lasso regression to the problem of channel selection [19] since low regression coefficients may be viewed as corresponding to low impact on the error function. This view of channel pruning was questioned by [14], who applied Lasso instead to the batch-norm [20] layers of the network and then removed any channels with low batch-norm coefficients, resulting in greater accuracy.

Another approach altogether is to approximate the effect of removing certain channels on the error using a Taylor series expansion [21] (which may be seen as an extension of [22]) to give a *pruning signal*. If weight this pruning signal with a FLOP penalty to bias the method towards removing computationally expensive neurons. While effective at providing a very regular set of small, dense matrices, we show that the solution found by Fisher pruning is very rarely optimal for specific deployment targets. Indeed, recent work has advocated training such networks from scratch [23].

### 3.1.3 Hardware Aware Reshaping

DPP-Net [24] is a platform-aware method for performing neural architecture search that uses Sequential Model-Based Optimisation to generate candidate architectures and a Recurrent Neural Network (RNN) to estimate potential accuracy that could be reached by training the network. This search includes soft targets of memory budget, energy consumption, and inference time. It is able to generate highly inference-efficient network architectures that also achieve high accuracy on the example datasets. However, the architecture search process is both very expensive and usually only applicable to constrained spaces.

A search-free approach was outlined in NetAdapt [25], which takes as input pre-trained models and iteratively applies pruning and retraining steps to each layer, suggesting several new layer sizes and empirically testing inference time on the target hardware. The authors compare their empirical testing against using FLOPs as a proxy for inference time and show that their method finds networks with faster inference times. While this is a very effective step towards hardware aware network architecture
3.1.4 Knowledge Distillation

Networks can also be made smaller through knowledge distillation \([8,9]\) whereby a small student network is trained both on the data and on the outputs of a larger pre-trained teacher network. This has been shown empirically to yield higher accuracy than training on just the data. Alternatively, the student can utilise the teacher’s intermediate activations \([26,27]\) for the same purpose. Typically the student network is thinner or has fewer layers than the teacher (as shown in Figure 2), but recent work has demonstrated the effectiveness of simply using the same network with inexpensive convolutions \([28]\).

3.2 Parallelism in Neural Networks

The most common operation performed in state-of-the-art image processing networks is a 2D convolution, which demands a significant majority of the computation time. Unfortunately, direct convolution has received very little attention from the perspective of acceleration and is therefore not well-optimised for a range of devices. For this reason, deep learning acceleration libraries often reduce convolution to matrix-matrix multiplication by stretching out the input image and kernels into matrices via the im2col algorithm \([29]\).

Using the matrix-matrix multiplication form of convolution makes data movement the bottleneck of inference for many hardware platforms. However, matrix-matrix multiplication has been heavily optimised by the compiler community and many techniques have been developed for hiding the latency of memory accesses while doing large multiplications. On GPUs, the most common approach is to use tiling and register blocking to maximise the ratio of data reuse to data movement (i.e. to keep parameters in memory for as long as possible and reuse them where possible to avoid reloading them later); instead of reading one element at a time, blocks of elements are read together so that they can be reused in the calculation. However, tile size selection can have a very significant impact on the efficiency of the operation \([30]\). Finding a tile size that correctly balances efficient use of the cache line width on the device and efficient fragmentation of the matrix is often a challenging task; given that GPUs usually have cache line widths set to powers of two, it is common to see significant speedups when the tiles can fit into power-of-two length words. Thus, when a tile does not perfectly fit the available cache it is common practice to pad the line of cache until it fits the word length (a similar technique is used to accelerate convolution on CPUs \([31]\)).

However, with convolutional neural networks as our workload we are in a unique position, because we have shown that we are effectively able to choose the sizes of the matrices to fit the device we are deploying to by reshaping the widths of the layers.

4 Hardware Aware Distillation

Neural compression techniques have largely been developed in a hardware-agnostic manner, with a strong focus on reducing the number of parameters (weights) without affecting accuracy. Conversely, systems researchers have focused on improving inference time for a fixed network structure.

We aim to unify both approaches using hardware aware knowledge distillation. Networks can be made smaller through knowledge distillation \([8,9]\) where a small student network is trained both on the data and on the outputs of a larger teacher network (Figure 2). We use knowledge of hardware behaviour to distil a smaller student network reduced in width from a teacher, which is likely to perform well on a specific device.
4.1 Distillation

To describe knowledge distillation \cite{8, 9} more precisely, let’s assume we have a dataset of pairs of images and class labels, where each image is denoted as $x$ and each label is a one-hot vector $y$. Given a pre-trained teacher network that outputs logits $t = \text{teacher}(x)$, we can learn the parameters for a student network (outputting logits $s = \text{student}(x)$) by minimising:

$$L_{KD} = (1 - \alpha)L_{CE}(y, \sigma(s)) + \alpha T^2 L_{CE} \left(\frac{t}{T}, \sigma \left(\frac{s}{T}\right)\right)$$

where $L_{CE}(p, q) = -\sum_k p_k \log q_k$ is the cross-entropy function, and $\sigma(.)$ is the softmax function (transforming the logits for each network into a vector of probabilities). $T$ is a fixed temperature parameter \cite{9}.

The first term is a standard cross-entropy loss for the student network, the second term however is minimised when the student network emulates the teacher. This technique allows the student to achieve a greater generalisation capability than training without a teacher. Note that $\alpha$ controls the ratio of the two terms.

4.2 Hardware Aware Knowledge Distillation

HAKD is a 2 step process as shown in Figure 3. First, we determine the best number of channels for an inference task based on hardware behaviour and then distil a network that best matches the function represented by the original teacher network.

**Step 1** We first iteratively apply Fisher pruning \cite{7} to our target network. This means we incrementally remove a channel from the network design, retrain the model and measure its inference time. From this we have a curve describing accuracy vs. channel count.

Given this trade-off curve, we then determine the best channel count. This is scenario specific. There are two possible strategies: maximising accuracy for a given inference time, and minimising inference time for a given accuracy.

The first strategy is to cluster the layer width choices towards maximising parameter counts; this means that any time a pruning technique converges to a point that sits on a step (see Figure 1), we choose the rightmost element on that step as our layer width. This technique will maximise the number of parameters in the model without affecting the inference time, usually resulting in greater

Figure 3: The Hardware Aware Distillation pipeline. In step 1, we use the Fisher pruned model and our staircase optimiser to design a student network that fits a specific deployment platform. In step 2 we use the original model to train the hardware constrained student using knowledge distillation.
accuracy performance (or, lower error) for a moderate increase in memory cost. We refer to this policy as **Accuracy-maximisation (ACC-MAX)**.

The second possible strategy is to minimise inference time in scenarios where lower accuracy is acceptable. In this case, we cluster the decisions of a pruning technique towards the rightmost element of the step below the current step. This means that we incur a slight hit in accuracy (due to a decreased parameter count) but leverage this into a significant speedup on the hardware in question. We refer to this clustering policy as **Inference-minimisation (INF-MIN)**.

**Step 2** Once we have determined the ideal number of channels, we apply knowledge distillation to train a student which has this fixed number of channels. At this stage, as the size and cost of the network is fixed, we simply aim to maximise accuracy.

In the remainder of the paper we consider only ACC-MAX and leave INF-MIN to be explored in future work.

5 Experimental Setup

In this section we describe the network architectures — MobileNet and DenseNet — and the dataset (CIFAR-10) we use for our experiments. We also provide details for how we train, and prune these networks, as well as the the hardware platforms we evaluate them on.

**Networks** MobileNet [32] is an architecture developed specifically for mobile devices. Instead of full convolutional kernels it utilises depthwise separable convolutions: a convolutional filter is applied individually to each input channel, the channels are then mixed through a $1 \times 1$ convolution [33]. DenseNet [34] is a very popular modern architecture that consists of convolutional blocks. We use bottleneck blocks whereby each block contains a $1 \times 1$ convolution, followed by a $3 \times 3$ convolution. Block outputs are concatenated and form the input to later blocks, and this encourages feature reuse to allow for powerful representations. Specifically, we use a 121 layer DenseNet with a growth rate of 32, and a transition reduction rate of 0.5.

**CIFAR-10** For our experiments we train and evaluate network models on CIFAR-10 [35], a standard machine learning benchmark dataset. It consists of 60,000 $32 \times 32$ pixel images across 10 object categories split into a train and test set of 50,000 and 10,000 images respectively.

**Training** To train networks from scratch, we use Stochastic Gradient Descent (SGD) with momentum to minimise the cross-entropy loss between class labels and network outputs. We use padding and left-right flips to augment the training data. We train for 200 epochs using mini-batches of size 128. The initial learning rate is 0.1 and is decayed by a factor of 0.2 every 60 epochs. We use weight decay of 0.0005 and momentum of 0.9. When performing knowledge distillation, we train the student network to instead minimise the loss in Equation 1 with $\alpha$ as 0.9 and $T$ as 4.

**Deep Compression** To perform Deep Compression, we take the trained network and lower the learning rate to $1/100$th of the last recorded learning rate. Every 30 epochs we increase the sparsity level by 10% by increasing the magnitude threshold, and retrain using L2 regularisation with a penalty of 0.0005. When the weights for a convolutional filter are all zero-valued, we remove the whole filter.

**Fisher Pruning** We perform Fisher pruning by fine-tuning the trained network with a learning rate of 0.0008. Every 10 steps, a single channel — the channel with the smallest effect on the loss — is pruned. For MobileNet we prune the connections between adjacent depthwise separable pairs. For DenseNet we prune the connections between the $1 \times 1$ and $3 \times 3$ convolution within each block.
Figure 4: Error curves for (a) DenseNet and (b) MobileNet on CIFAR-10 when Fisher pruning and Deep Compression are applied. We highlight the points chosen for later comparisons for both techniques, as well as our Hardware Aware Knowledge Distilled architectures.

### Hardware Platforms

We evaluated our approach on 3 different hardware platforms. The first one is an Intel Core i7-3820 CPU with 4 cores @ 3.60GHz and 16GB of DDR2 RAM. We also consider the quad-core ARM Cortex-A57 CPU, and the 256-core Pascal GPU, both present on the Nvidia Jetson TX2 development board with 8GB of LPDDR4 RAM.

### 6 Results

In this section, we first consider the performance of our distillation technique on the test set of CIFAR-10 in terms of error. We then explore the inference time for each approach and present speedups over the default networks. This is followed by an evaluation of how network size and accuracy trade off for the main compression techniques.

#### 6.1 Accuracy

Consider the bar plot in [Figure 5]. Here we plot the CIFAR-10 top-1 classification error for each compression technique when applied to DenseNet-121 and MobileNet. The original DenseNet-121 network has an error-rate of 4.7%. Fisher pruning reduces the size of the network but at a cost of increased error-rate (5.5%). Deep Compression increases error still further (5.8%). The 3 HAKD schemes have superior accuracy: between 5 and 5.25% error depending on platform.

A similar trend is observed for MobileNet. Once again HAKD has improved error-rates relative to Deep Compression and Fisher pruning.

We show this again in [Figure 4] which shows the error curves for each compression technique on DenseNet-121 and MobileNet. If we slice the architecture found by Fisher pruning at the highlighted points and reshape it for each of our hardware platforms then we are able to leverage increased parameter count into a significant decrease in error, for a fixed inference time.

#### 6.2 Speedup over Original Networks

Reducing the number of weights in a network reduces the amount of computation and hence we expect to reduce inference time at the cost of the increased error-rate shown above.
Figure 5: Top-1 Error on the CIFAR-10 dataset with compression techniques applied. (a) is DenseNet, (b) is MobileNet. We see that for both networks, our Hardware Aware Knowledge Distilled architectures outperform both pruning techniques.

Figure 6: DenseNet-121 speedup over the original model on three devices: (a) is the ARM Cortex, (b) is the Jetson TX2, (c) is the Intel Core i7. Our speedup over Deep Compression is large enough that we would also outperform the hardware-aware Scalpel approach.

In Figure 6 we show the relative performance of the 3 compression techniques on 3 different platforms for DenseNet-121. While Deep Compression is able to decrease inference time by 1.1x to 2x, this is outperformed by Fisher pruning which gives speedups from 2x to 7x. Given that Fisher pruning also has reduced error, it clearly outperforms Deep Compression.

A similar pattern holds for MobileNet (Figure 7). However, given that the Jetson board and the i7 platform are large relative to the network, pruning has little impact on execution time here.

In all cases HAKD has the same speedup as Fisher pruning, while having superior accuracy as shown above.

6.3 Parameter Size

Here we explore in more detail the trade off in network size vs. error. Consider Figure 4 this shows the error rate for two networks for a varying number of network parameters. As expected, Fisher pruning gives increased accuracy over Deep Compression for a particular network size. On
Figure 7: MobileNet speedup over the original model on three devices: (a) is the ARM Cortex, (b) is the Jetson TX2, (c) is the Intel Core i7. Again, our speedup over Deep Compression is large enough that we would also outperform the hardware-aware Scalpel approach.

both graphs are the solution points found by HAKD for the 3 platforms after network size has been selected. In each case HAKD gives reduced error rates for a particular size. Alternatively, this graph can be read as: for a particular error rate, HAKD gives a smaller network.

7 Conclusion

It has long been understood that significant parameter redundancy exists in many deep neural networks. Now that compression techniques have matured, we are able to take advantage of insights from both developments in compiler optimisation and improvements in neural network acceleration schemes to provide an across-stack approach to optimising CNNs for specific tasks and devices. We show that taking an across-stack approach observing hardware behaviour allows us to outperform popular pruning techniques in terms of both accuracy and inference speed.

Acknowledgements. This project has received 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.

References

[1] Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. Deep learning. Nature, 521(7553):436–444, 2015.

[2] Pete Warden. Why the future of machine learning is tiny. https://tinyurl.com/yculf7wu, 2018.

[3] Misha Denil, Babak Shakibi, Laurent Dinh, Nando de Freitas, et al. Predicting parameters in deep learning. In Advances in Neural Information Processing Systems, pages 2148–2156, 2013.

[4] Song Han, Huizi Mao, and William J Dally. Deep compression: Compressing deep neural networks with pruning, trained quantization and Huffman coding. arXiv preprint arXiv:1510.00149, 2015.

[5] Jiecao Yu, Andrew Lukefahr, David Pulframan, Ganesh Dasika, Reetuparna Das, and Scott Mahlke. Scalpel: Customizing DNN pruning to the underlying hardware parallelism. In Proceedings of the 44th Annual International Symposium on Computer Architecture, pages 548–560. ACM, 2017.

[6] Jack Turner, José Cano, Valentin Radu, Elliot J. Crowley, Michael O’Boyle, and Amos Storkey. Characterising across stack optimisations for deep convolutional neural networks. In IEEE International Symposium on Workload Characterization, 2018.
[7] Lucas Theis, Iryna Korshunova, Alykhan Tejani, and Ferenc Huszár. Faster gaze prediction with dense networks and fisher pruning. arXiv preprint arXiv:1801.05787, 2018.

[8] Lei Jimmy Ba and Rich Caruana. Do deep nets really need to be deep? In Advances in Neural Information Processing Systems, 2014.

[9] Geoffrey Hinton, Oriol Vinyals, and Jeff Dean. Distilling the knowledge in a neural network. arXiv preprint arXiv:1503.02531, 2015.

[10] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In Proceedings of the IEEE conference on computer vision and pattern recognition, pages 770–778, 2016.

[11] Simone Scardapane, Danilo Comminiello, Amir Hussain, and Aurelio Uncini. Group sparse regularization for deep neural networks. Neurocomputing, 241:81–89, 2017.

[12] Stephen Jose Hanson and Lorien Y. Pratt. Comparing biases for minimal network construction with back-propagation. In D. S. Touretzky, editor, Advances in Neural Information Processing Systems 1, pages 177–185. Morgan-Kaufmann, 1989.

[13] Song Han, Jeff Pool, John Tran, and William Dally. Learning both weights and connections for efficient neural network. In Advances in Neural Information Processing Systems, pages 1135–1143, 2015.

[14] Jianbo Ye, Xin Lu, Zhe Lin, and James Z Wang. Rethinking the smaller-norm-less-informative assumption in channel pruning of convolution layers. arXiv preprint arXiv:1802.00124, 2018.

[15] Sida Wang and Christopher Manning. Fast dropout training. In Proceedings of the 30th International Conference on Machine Learning (ICML-13), pages 118–126, 2013.

[16] Diederik P Kingma, Tim Salimans, and Max Welling. Variational dropout and the local reparameterization trick. In Advances in Neural Information Processing Systems, pages 2575–2583, 2015.

[17] Dmitry Molchanov, Arsenii Ashukha, and Dmitry Vetrov. Variational dropout sparsifies deep neural networks. arXiv preprint arXiv:1701.05369, 2017.

[18] Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. Gradient-based learning applied to document recognition. In Proceedings of the IEEE, pages 2278–2324, 1998.

[19] Yihui He, Xiangyu Zhang, and Jian Sun. Channel pruning for accelerating very deep neural networks. In International Conference on Computer Vision (ICCV), volume 2, page 6, 2017.

[20] Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In International Conference on Machine Learning, pages 448–456, 2015.

[21] Pavlo Molchanov, Stephen Tyree, Tero Karras, Timo Aila, and Jan Kautz. Pruning convolutional neural networks for resource efficient inference. In International Conference on Learning Representations, 2017.

[22] Yann LeCun, John S Denker, and Sara A Solla. Optimal brain damage. In Advances in Neural Information Processing Systems, pages 598–605, 1990.

[23] Elliot J. Crowley, Jack Turner, Amos Storkey, and Michael O’Boyle. Pruning neural networks: is it time to nip it in the bud? arXiv preprint 1810.04622, 2018.

[24] Jin-Dong Dong, An-Chieh Cheng, Da-Cheng Juan, Wei Wei, and Min Sun. DPP-Net: Device-aware progressive search for Pareto-optimal neural architectures. arXiv preprint arXiv:1806.08198, 2018.
[25] Tien-Ju Yang, Andrew Howard, Bo Chen, Xiao Zhang, Alec Go, Vivienne Sze, and Hartwig Adam. Netadapt: Platform-aware neural network adaptation for mobile applications. arXiv preprint arXiv:1804.03230, 2018.

[26] Adriana Romero, Nicolas Ballas, Samira Ebrahimi Kahou, Antoine Chassang, Carlo Gatta, and Yoshua Bengio. FitNets: Hints for thin deep nets. CoRR, abs/1412.6550, 2014. URL http://arxiv.org/abs/1412.6550.

[27] Sergey Zagoruyko and Nikos Komodakis. Paying more attention to attention: Improving the performance of convolutional neural networks via attention transfer. In International Conference on Learning Representations, 2017.

[28] Elliot J. Crowley, Gavin Gray, and Amos Storkey. Moonshine: Distilling with cheap convolutions. In Advances in Neural Information Processing Systems, 2018.

[29] Sharan Chetlur, Cliff Woolley, Philippe Vandermersch, Jonathan Cohen, John Tran, Bryan Catanzaro, and Evan Shelhamer. cudnn: Efficient primitives for deep learning. arXiv preprint arXiv:1410.0759, 2014.

[30] Kazushige Goto and Robert A Geijn. Anatomy of high-performance matrix multiplication. ACM Transactions on Mathematical Software (TOMS), 34(3):12, 2008.

[31] Evangelos Georganas, Sasikanth Avancha, Kunal Banerjee, Dhiraj Kalamkar, Greg Henry, Hans Pabst, and Alexander Heinecke. Anatomy of high-performance deep learning convolutions on simd architectures. arXiv preprint arXiv:1808.05567, 2018.

[32] Andrew G. Howard, Menglong Zhu, Bo Chen, Dmitry Kalenichenko, Weijun Wang, Tobias Weyand, Marco Andreetto, and Hartwig Adam. MobileNets: Efficient convolutional neural networks for mobile vision applications. arXiv preprint arXiv:1704.04861, 2017.

[33] Min Lin, Qiang Chen, and Shuicheng Yan. Network in network. In International Conference on Learning Representations, 2014.

[34] Gao Huang, Zhuang Liu, Laurens van der Maaten, and Kilian Q. Weinberger. Densely connected convolutional networks. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, 2017.

[35] Alex Krizhevsky. Learning multiple layers of features from tiny images. Master’s thesis, University of Toronto, 2009.