Energy Guided DNN Compression for GPU Systems

Nicodemus M.J. Mbwambo   
Computer Science Department  
*Clemson University*SC, USA  
nmbwamb@clemson.edu

Thomas Randall   
Computer Science Department  
*Clemson University*SC, USA  
tlranda@clemson.edu

*Abstract*—This work proposes an algorithm that automatically compresses pre-trained DNNs for use on energy-constrained systems such as mobile phones, embedded computers, and wearable technology. Existing algorithms are guided by theoretical representations of the chief contributing factors to resource utilization and performance, or by subsets of direct metrics that contribute to energy utilization. By utilizing device-specific energy measurements, our algorithm is able to iteratively optimize the DNN compression process to reduce energy consumption on GPU platforms while preserving the original model accuracy, all without utilizing expert knowledge of the platform’s energy characteristics or the DNN architecture itself. A proof-of-concept implementation is able to reduce the energy consumption of pre-trained MobileNet DNNs on NVIDIA Tesla V100 GPUs by up to 50% while maintaining over 99% of the original model’s accuracy, and achieves over 60% compression while maintaining nearly 83% of the original model’s accuracy.

Keywords—DNN compression, energy-aware systems, resource-constrained computing.

# Introduction

Deep Neural Networks (DNNs) are leading the state-of-art for many modern AI applications such as image recognition and classification. Deep learning provides near-human accuracy on these tasks and are increasingly being employed as means of enriching user experiences on devices such as mobile phones, embedded computers and wearable technology [1, 2, 3]. Despite this growing trend, it is challenging to employ DNNs on these devices within acceptable resource utilization tolerances. This challenge exists largely because DNNs are designed and trained using large, distributed systems that have far more resources than the platforms that the trained network will ultimately be employed upon. This means that the DNN model usually requires some form of compression to allow it to be used on resource-constrained platforms, but care must be taken as naïve changes may significantly reduce the trained model’s accuracy and undermine its usefulness. The amount of energy consumed with each inference is a critically important characteristic for DNN usage on resource-constrained devices. For instance, as reported in [5], modern smartphones cannot perform real-time object classification with uncompressed AlexNet models for more than one hour, severely limiting the usefulness of the technology.

Existing algorithms that compress DNN models for energy efficiency are guided by “indirect metrics,” such as the number of multiply-accumulate operations (MACs) or the number of weights in the DNN model. These algorithms rely on these features as appropriate theoretical representations of real energy consumption, but [5] shows that indirect metrics do not sufficiently represent the direct metrics that most compression tools seek to optimize. This work by Tien-Ju Yang et. al. introduces a compression tool, called NetAdapt, that utilizes platform-specific direct measurements to guide the compression process for inference latency and a variety of other resource targets. While impressive, their tool did not support DNN compression for energy usage at inference.

We are interested in a DNN compression framework that reliably decreases target resource utilization for DNN architectures in as platform-agnostic approach as possible. We begin our approach to such a framework with this work, where we present a tool based upon NetAdapt that shows how empirical, platform-driven energy measurements can be utilized to optimize DNNs for energy-constrained systems while maintaining the desired accuracy of the original DNN model. We analyze the effectiveness of a proof-of-concept implementation and examine the opportunities and challenges that it presents for the future.

# RELATED WORK

Pre-trained DNN models can be compressed by pruning filters or weights. DNN compressing tools that seek to bring DNN models to resource-constrained devices utilize either pruning method to produce a smaller model with similar classification ability but better performance in some regard, such as the latency for inferencing, energy expenditure, or total memory utilization. We limit our discussion to filter and weight pruning for energy expenditure at inference, and refer readers to the survey in [6] for a more complete discussion of DNN compression techniques and goals.

Compressing a DNN model’s size by pruning weights, known as fine-grained pruning, removes individual weights on a network. This is typically accomplished by setting the pruned weight values to zero. The specific weights to be pruned may be selected by a number of different criterion to different effect. Yann LeCun’s work in [7] focuses on removing unimportant weight, while [8] aims to remove redundant connections and [9] removes weights with the least sensitivity first. All fine-grained pruning methods, regardless of which weight selection algorithm is selected, create unstructured sparse filters that may reduce performance rather than improve it, as reported in [9]. In terms of energy expenditure, sparse filters introduce additional memory that can degrade the efficiency of a platform’s memory hierarchy, making this method of pruning less than ideal for our purposes.

The other approach to DNN pruning is coarse-grained pruning, which removes entire filters instead of individual weights. As with fine-grained pruning, different criterion for pruning have been explored to different effects. Filters that frequently generate zero outputs after the ReLU layer in the validation set are removed in [10]. Other works such as [11] and [12] have explored reducing the number of filters by combining similar filters. As with fine-grained pruning, removing entire filters does not always translate to a reduction in energy consumption because not all filters interact with the system in the same manner. Because the compression algorithms in these works do not select which filters to remove based on energy consumption, filters that consume the most energy may not be pruned and energy-conservative filters may be removed instead. Furthermore, coarse-grained pruning reduces the network size much faster than fine-grained pruning, so it cannot be utilized to compress models to the same degree as fine-grained pruning approaches. However, the approach is more useful for energy-based pruning because the removal of entire filters is more likely than fine-grained pruning to result in decrease of both memory utilization and computation and by extension energy usage. This makes coarse-grained filtering generally more suitable for energy-based compression across a wide variety of systems.

In all of the aforementioned works, regardless of the kind of pruning used, the compression process is guided by theoretical models that rely on indirect metrics, such as the number of weights or MAC operations, to function as indicators that will reliably predict actual changes in the model’s performance. However, we know that these particular indirect measures do not completely capture the actual behavior behind direct resource utilization, so tools such as NetAdapt [5] utilize direct metrics to ensure that the proper correlation is both recognized and acted upon by the compression algorithm.

Energy-Aware Pruning [4] introduces a fine-grained pruning algorithm that specifically targets energy-efficiecy, estimating the energy usage of a CNN as being composed of computation energy and data movement energy. The compression algorithm orders layer pruning based on layer-specific energy estimations that are weighted based on network characteristics. The order is important because the algorithm compresses each layer with each iteration and the first layers to be compressed will have the largest impact on both the model’s change in accuracy and the model’s change in resource consumption. Weights in each layer are removed based on the magnitude, then some weights are restored to reduce output error before locally fine-tuning the compressed network to restore accuracy. Once all layers have been pruned, the iteration is completed with a global weight fine-tuning. Iterations continue until the compressed model accuracy falls beneath a specified threshold.

A major drawback to this approach is the requirement to design a framework that estimates energy consumption for each individual target platform. This requires detailed knowledge of the target platform, network architecture [13] and toolchains such as network-to-array mapping [14]. To overcome these difficulties, the same group produced [5], a compression algorithm that instead uses coarse-grained pruning and direct measurements instead of an estimation framework.

NetAdapt automatically compresses a pre-trained DNN models to taget platforms, where the compression process is driven by set of target resource requirements to satisfy. Empirical measurements from the platform and DNN architecture combination determine the success of any given change to the DNN model, permitting NetAdapt to support any combination of platform and network architecture without requiring detailed knowledge of the target platform’s characteristics. We find this to be a desirable manner to approach the problem due to its flexibility and ease of use.

NetAdapt receives a pre-trained DNN with a specified resource budget, such as reducing inference latency by 25%, and iteratively progresses towards this goal, producing a family of optimized networks that exhibit tradeoffs observed at each iteration that the algorithm completes.

During each iteration, NetAdapt compresses either one convolution or one fully connected layer. Each layer independently creates a proposal for the compression that is estimated to satisfy the current iteration’s resource budget. The resource usage estimation is derived from empirical measurements of individual filter behavior gathered from analyzing the performance of individual components of the network on the target platform.

Once the number of filters to be removed is determined, NetAdapt removes filters with the lowest *L*2 norm. This preserves filters that contribute the most to differentiable output, theoretically contributing the most to the model’s accuracy. Each layer’s proposal is then allowed to re-train to recover some accuracy and evaluated against a holdout set of data to determine the best layer proposal to choose for the iteration. Unlike Energy-Aware Pruning, NetAdapt only prunes one layer per iteration, because the coarse-grained pruning is a more significant change than several fine-grained pruning selections across all layers of the model. NetAdapt considers the best layer proposal as the proposal that meets the resource reduction requirement and has the highest accuracy after short-term retraining, with ties broken by proposals that achieve better resource reduction.

Once an iteration completely satisfies the given resource reduction budget or the maximum number of iterations is completed, the highest accuracy model is allowed to train on both the training set and the hold-out set for a longer duration to restore as much accuracy as possible to the final compressed model.

NetAdapt’s strength lies in its empirically-driven filter selection process, which does not require complex theoretical models and the fact that its proposal selection technique means that NetAdapt always makes process towards the target resource optimization while maintaining the highest possible accuracy. Because the model’s filter selection process considers the number of filters to keep separately from which filters to keep, the “search space” of the algorithm is much smaller than other approaches, resulting in much faster compression. The empirical measurements NetAdapt supports are gathered in platform-independent manners, making NetAdapt very easy to introduce to new systems. While expanding NetAdapt to support new DNN architectures requires some additional programming effort and network architectural knowledge, the barrier for doing so is relatively low.

There are some disadvantages to NetAdapt as it was originally proposed. While platform-specific knowledge is not required to utilize NetAdapt as a compression tool, every unique DNN architecture and platform architecture combination requires NetAdapt to build the lookup table it uses to speed up compression, which is a time- and resource-expensive process that involves automated testing of every decomposable portion of the DNN architecture on the platform architecture. In our experience, this process could easily take a day or more to complete, requiring much more time on weaker systems. While any model trained using known platform-architecture combinations can be compressed with NetAdapt using the combination’s lookup table, the immense initial cost means this tool is not very appropriate for settings where DNN architectures are likely to change.

Furthermore, NetAdapt’s platform-agnostic approach meant that it does not support some direct metrics that would require platform-specific means of measurement, such as energy consumption per inference. This means that NetAdapt is not necessarily better than theoretically-driven approaches when it comes to compressing DNN models for these purposes. Finally, NetAdapt’s separate consideration of how many filters to keep and which filters it should keep means that it does not explore as many possibilities as other methods, and it can fail to identify some pruning choices that other methods identify.

Overall, NetAdapt excels at meeting target resource reduction requirements in a platform-agnostic and easily-extensible manner. Its usage of coarse-grained pruning makes it more appropriate for energy-efficient DNN compression, making it our preferred starting point over Energy-Aware Pruning.

# METHODOLOGY

We introduce energy consumption at inference as a new resource metric for NetAdapt, allowing it to compress DNNs such that they support better energy efficiency, and do so without requiring the input of any platform-specific knowledge. This ability is critically important for resource-constrained systems but not necessarily measurable across all platforms in the same manner, so we begin by utilizing the *nvidia-smi* utility provided by NVIDIA for their GPUs to represent a class of platforms that can be optimized with respect to energy consumption. GPUs are power-hungry hardware devices, and the *nvidia-smi* utility provides us with power draw information in a consistent manner across many different platforms, enabling us to gather the direct empirical measurements we will need to guide our compression algorithm. While the *nvidia-smi* tool does not allow us to support other platforms, such as CPU-only systems or mobile phones, we utilize it as an appropriate tool to demonstrate a proof-of-concept implementation of our idea.

We use *nvidia-smi* to monitor the power draw of NVIDIA GPUs and calculate the average energy expended per DNN inference according to the equation:



This platform-specific metric can be used by our NetAdapt-derived tool to iteratively compress DNN models in a manner that preserves the original accuracy on a best-effort basis while decreasing the energy usage. Following a similar methodology as the original NetAdapt implementation, we create a lookup table of layer and filter measurements for different network configurations based on the DNN architecture and measured on the target platform. This process involves performing a series of tests that measure the energy expenditure per inference on the target platform. The lookup table records an associated resource cost for each layer and filter within the target DNN architecture. As was the case with NetAdapt, this process is expensive to perform both in terms of time and resource commitment, but the pre-processing allows later compression searching to proceed much faster, and only has to be performed once for each platform-architecture combination.

As was the case with NetAdapt, our compression algorithm utilizes the lookup table to identify the maximum number of filters that can be maintained for the current iteration budget, then the identified number of excess filters are removed by least *L2* Norm, prioritizing the removal of layers contributing the least magnitude to differentiable outputs from the model.

# EVALUATION

We evaluate our implementation by compressing DNN models to utilize 60% less energy per inference on our NVIDIA Tesla V100-SXM3-32GB GPUs. These experiments show how our implementation can measure the energy expenditure of inferences performed on the device and utilize that information to guide the DNN compression algorithm in such a way as to reduce its energy consumption with minimal loss in accuracy.

As was the case for NetAdapt’s evaluations, we hold out a ten-thousand image subset of the CIFAR-10 dataset for periodic accuracy evaluations and use the rest of the dataset for short-term fine-tuning of compression proposals. The full CIFAR-10 dataset is used for the final long-term fine-tuning to restore as much accuracy as possible to the final network. Our results are shown in Tables 1 and 2.

1. MobileNet Energy Consumption Reduction

|  |  |  |  |
| --- | --- | --- | --- |
| Iteration # | Energy Per Inference (Joules) | Model Accuracy (CIFAR-10) | Energy Reduction |
| 0 | 6.021 | 100.000% | 0.000% |
| 1 | 5.852 | 100.000% | 2.807% |
| 5 | 5.070 | 100.000% | 15.795% |
| 10 | 4.421 | 100.000% | 26.574% |
| 15 | 3.890 | 99.998% | 35.393% |
| 20 | 3.485 | 99.924% | 42.119% |
| 25 | 3.145 | 99.456% | 47.766% |
| 30 | 2.831 | 97.118% | 52.981% |
| 35 | 2.652 | 95.028% | 55.954% |
| 40 | 2.463 | 87.248% | 59.093% |
| 42 | 2.369 | 82.630% | 60.647% |

Target resource reduction: 60%

Maximum Iterations: 45

1. AlexNet Energy Consumption Reduction

|  |  |  |  |
| --- | --- | --- | --- |
| Iteration # | Energy Per Inference (Joules) | Model Accuracy (CIFAR-10) | Energy Reduction |
| 0 | 1.819 | 98.982% | 0.000% |
| 1 | 1.773 | 98.990% | 2.529% |
| 2 | 1.714 | 98.724% | 5.772% |
| 3 | 1.667 | 98.606% | 8.356% |
| 4 | 1.623 | 98.554% | 10.775% |
| 5 | 1.583 | 98.450% | 12.974% |
| 6 | 1.545 | 98.364% | 15.063% |

Target resource reduction: 60%

Maximum Iterations: 45

## Discussion of MobileNet Evaluation

Our implementation is able to reach a 60.647% energy reduction for MobileNet using 42 of its allotted 45 iterations as shown in Table 1. At iteration 29, we pass 50% compression (2.999 Joules per inference) with 98.954% accuracy, meaning that we were able to reduce the energy utilization by half while maintaining nearly 99% of the original model’s accuracy.

We designed a second experiment like the first, but with no maximum iteration cap and an infeasible goal for energy reduction to see how far our network could actually compress the MobileNets model. Surprisingly, it could not go much further, and was forced to halt after its 48th iteration failed to improve upon the 47th in terms of resource reduction. The final network had reduced energy per inference to 2.233 Joules (a 62.92% reduction) and had a very poor accuracy of 18.78%. However, the previous iteration was able to maintain 64.27% accuracy with a measured 0.004 additional Joules per inference.

This likely indicates that the compression tool was either being too aggressive with the final iterations or was reaching the theoretical limitations of what it could achieve via coarse-grained pruning without destroying critical portions of the original model. A more extensive parameter sweep may have been able to locate a better solution, but we did not have time to perform that search.

## Discussion of AlexNet Evaluation

Our implementation was halted early due to time constraints on the submission of this report, but was able to achieve 15% energy reduction for our pretrained AlexNet model using 6 of its allotted 45 iterations as shown in Table 2. These results should be considered incomplete, as testing was unable to finish prior to the deadline for this report.

AlexNet uses significantly less energy per inference on our test platform than MobileNet, making it much more difficult to compress, as the compression tool is neither aware of nor compensating for the hardware’s idle energy consumption. Despite the increased challenge, we are able to improve AlexNet significantly. The iterations AlexNet did complete produce less energy reduction as a percentage of the original energy expenditure than MobileNet, but are close enough for us to believe that similar results are achievable with this framework as well, given additional time. We assume that there is a maximum iteration at which the AlexNet compression would fail to reduce the energy consumption, as was observed with MobileNet. We believe that AlexNet would be more likely to encounter the issue due to falling within the error tolerance of *nvidia-smi’s* power reporting and the idle power draw of the device, but were unable to complete a sufficient number of iterations to have data supporting that hypothesis included in the report.

# CONCLUSION AND FUTURE WORK

In this work, we created a DNN compression tool based on NetAdapt that prioritizes model accuracy while reducing the energy expenditure per inference on GPU platforms. We were able to achieve great success with pretrained MobileNets and AlexNet models without providing the tool with any platform-specific knowledge outside of the direct energy measurements. We are able to reduce the energy consumption per inference of MobileNets by 50% while only losing 1% accuracy, and can increase this to 60% compression at the cost of 17% accuracy. With AlexNet, we are able to reduce the energy consumption per inference by 15% with a 2% reduction in accuracy.

This work should be considered preliminary in nature and is still quite limited in scope and effectiveness. In particular, it only supports energy-based compression for NVIDIA GPUs, a small subset of the resource-constrained systems that are interesting to examine for this problem. In the future, we plan to utilize energy measurement software for mobile phones and other resource-constrained devices, where the results of our work are more interesting and meritous.

Our results showed that continued compression using only coarse-grained pruning makes it increasingly difficult to maintain accuracy, both as the network size decreases and as it becomes more difficult for the compression tool to keep up with its scheduled increments. In the future, we want to see if NetAdapt’s pruning methods can be expanded to utilize fine-grained pruning on later iterations when coarse-grained pruning is not necessarily the best way to proceed.

The largest obstacle to producing results for this report proved to be the prohibitively expensive process of generating lookup tables for platform-architecture combinations. Future work may greatly speed up this process by utilizing better measurement tools than *nvidida-smi* and exploiting more parallelism for measurement across homogeneous devices, which neither NetAdapt nor our implementation do when building lookup tables.

##### References

|  |  |
| --- | --- |
| [1] | A. Krizhevsky, I. Sutskever and G. E. Hinton, "ImageNet Classification with Deep Convolutional Neural Networks," in *Advanced Neural Information Processing Systems*, 2012. |
| [2] | K. Simonyan and A. Zisserman, "Very Deep Convolutional Networks For Large-Scale Image Recognition," in *ICLR*, 2015. |
| [3] | K. He, X. Zhang, S. Ren and J. Sun, "Deep Residual Learning for Image Recognition," in *IEEE Conference Computer Vision Pattern Recognition (CVPR)*, 2016. |
| [4] | T.-J. Yang, Y.-H. Chen and V. Sze, "Designing Energy-Efficient Convolutional Neutral Networks Using Energ-Aware Pruning," in *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)* , 2017. |
| [5] | T.-J. Yang, A. Howard, B. Chen, X. Zhang, A. Go, M. Sandler, V. Sze and H. Adam, "NetAdapt: Platform-Aware Neural Network Adaptation for Mobile Applications," in *ECCV*, 2018. |
| [6] | J. S. Emer, Y. Tien-Ju and Y.-H. Chen, "Efficient Processing of Deep: A Tutorial and Survey," *Proceedings of the IEEE 105(12),* pp. 2295-2329, December 2017. |
| [7] | Y. L. Cun, J. S. Denker and S. A. Solla, "Optimal Brain Damage," in *Advances in Neural Information Processing Systems*, 1990. |
| [8] | S. Han, J. Pool, J. Tran and W. J. Dally, "Learning bothWeights and Connections for Efficient Neural Networks," in *Advances in Neural Information Processing*, 2015. |
| [9] | H. Hassibi and D. G. Stork, "Second order derivatives for network prunning: Optimal Brain Surgeon," in *NIPS*, 1993. |
| [10] | H. Hu, R. Peng, Y.-W. Tai and C.-K. Tang, "Network Trimming: A Data-Driven Neuron Pruning Approach towards Efficient Deep Architectures," in *arXiv preprint arXiv:1607.03250*, 2016. |
| [11] | S. Srinivas and V. R. Babu, "Data-free Parameter Pruning for Deep Neural Networks," in *BMVC*, 2015. |
| [12] | Z. Meriet and S. Sra, "Diversity Networks: Neural Network Compression Using Determinantal Point Process," in *ICLR*, 2016. |
| [13] | Y.-H. Chen, T. Krishna, J. S. Emer and V. Sze, "Eyeriss: An Energy-Efficient Reconfigurable Accelerator for Deep Convolutional Neural Networks," in *43rd Annual International Symposium on Computer Architecture (ISCA)*, 2016. |
| [14] | Y.-H. Chen, J. Emer and V. Sze, "Eyeriss: A Spatial Architecture for Energy-Efficient Dataflow for Convolutional Neural Networks," in *43rd Annual International Symposium on Computer Architecture (ISCA)*, 2016. |
| [15] | J. Yu, A. Lukefahr, D. Palframan, G. Dasika, R. Das and S. Mahlke, "Scapel: Customizing DNN Prunning to the Underlying Hardware Parallelism," in *Proceedings of the 44th Annual International Symposium on Computer Architecture* , 2017. |
| [16] | W. Wen, C. Wu, Y. Wang, Y. Chen and H. Li, "Learning Structured Sparsity in Deep Neural Network," in *CoRR*, 2016. |