End-to-End Neural Network Compression via $\frac{\ell_1}{\ell_2}$ Regularized Latency Surrogates

Anshul Nasery[1†] Hardik Shah[2†] Arun Sai Suggala[3] Prateek Jain[3]

[1]University of Washington [2] ETH, Zurich [3]Google Research, India

Abstract

Neural network (NN) compression via techniques such as pruning, quantization requires setting compression hyperparameters (e.g., *number of channels to be pruned, bitwidths for quantization) for each layer either manually or via neural architecture search (NAS) which can be computationally expensive. We address this problem by providing an end-to-end technique that optimizes for model's Floating Point Operations (FLOPs) via a novel* $\frac{\ell_1}{\ell_2}$ *latency surrogate. Our algorithm is versatile and can be used with many popular compression methods including pruning, low-rank factorization, and quantization, and can optimize for on-device latency. Crucially, it is fast and runs in almost the same amount of time as a* single model training run*; which is a significant training speed-up over standard NAS methods. For BERT compression on GLUE fine-tuning tasks, we achieve* 50% *reduction in FLOPs with only* 1% *drop in performance. For compressing MobileNetV3 on ImageNet-1K, we achieve* 15% *reduction in FLOPs* without drop in accuracy*, while still requiring* $3\times$ *less training compute than SOTA NAS techniques. Finally, for transfer learning on smaller datasets, our technique identifies* $1.2\times$-$1.4\times$ *cheaper architectures than standard MobileNetV3, EfficientNet suite of architectures at almost the same training cost and accuracy.*

1. Introduction

Large-scale neural networks consistently provide state-of-the-art performance on complex learning tasks [22, 29, 63]. But they place heavy burden on compute resources such as battery, memory or processor making them hard to deploy on edge devices such as phones, cameras and wearables. Several recent works have designed techniques to compress ML models and make them efficient for inference. However, as detailed below, many of these techniques are hard to use in practice, and often achieve sub-optimal accuracy *vs* inference time trade-offs.

†Work done at while Google Research, India. Correspondence to anshulnasery@gmail.com

Hyperparameter search for compression. Existing works typically rely on one of the following building blocks to design efficient models: unstructured weights sparsity [21, 32, 66], pruning entire neurons or low-rank factorization [25, 72], quantization [48], distillation [7, 23]. Figuring out an optimal way to combine these building blocks (or to figure out hyper-parameters such as amount of sparsity associated with each block) while satisfying a global FLOPs/latency/resource constraint is difficult and involves a combinatorial search. This problem is further exacerbated when multiple building blocks are used for model compression (*e.g.,* simultaneous low rank factorization, sparsity/pruning of weights).

Over the past few years, there has been a large body of work that addresses the problem of finding hyperparameters for model compression. Existing literature in this space can be broadly classified into two categories depending on the style of optimization techniques employed: blackbox, and whitebox techniques.

Blackbox Compression Techniques. Several works in this category formulate model compression as a blackbox Neural Architecture Search (NAS) problem and rely on state-of-the-art NAS techniques to search for efficient models [28, 79, 88]. These techniques directly take the FLOPs/latency into account and have the potential to identify the optimal per-layer budget allocation for a wide variety of efficient blocks/compression mechanisms. However, these approaches are often computationally expensive as they take a blackbox view of the problem and perform combinatorial search over the space of architectures. Recent works have tried to open this blackbox to speed up the search process. One prominent line of work here is based on weight sharing which involves training a large surrogate network with many redundant operations to quickly evaluate the quality of an architecture in the search space [8, 41, 60, 64]. However, these techniques do not scale well to large search spaces, as they require storing a gigantic network. Despite recent advances such as TuNAS [2] for reducing the size of the network, these techniques can be an order of magnitude slower and less accurate than our proposed method (see Fig 1). See Section 2 for a thorough discussion on other related works.

Whitebox Compression Techniques. Among the category (b) techniques mentioned above, a prominent line of work has focused on unstructured pruning of weights with non-uniform budget allocation across layers [21, 32, 39, 55]. However, any gain in FLOPs using unstructured pruning is hard to translate to real latency gain as modern hardware – like GPUs, TPUs – are more geared towards dense matrix operations. So it is more fruitful to focus on structured building blocks such as neuron pruning, which removes entire neurons/channels, and low-rank factorization of weights, which is closely related to neuron pruning. Recent techniques in this line of work add a latency/FLOPs regularizer to the standard cross entropy loss [8, 9, 19] to bias the model towards lower number of neurons. Unfortunately the resulting objective is discrete and difficult to optimize. To alleviate this, existing works have designed continuous surrogates that are more amenable to SGD style optimization. These methods either work in the space of probability distributions over pruned models and optimize the "expected objective" [9, 43, 72] or replace the discontinuous FLOPs regularizer with a continuous surrogate such as ℓ_1 norms of the weights of the network [19]. However, the former class of techniques are often unstable, hard to implement in practice, and empirical studies indicate that their performance is similar to that of simple magnitude based pruning [18] (also see left plot of Fig. 1). Furthermore, as we show in this work, the latter class of techniques fail to enforce sparsity in the presence of batch, layer normalization (see Section 3). Even in the absence of batch, layer normalization, these techniques require adhoc post-processing steps to output exact sparse solutions.

Our Approach: In this work, we propose a whitebox compression technique that addresses the above described optimization issues. Specifically, we propose a novel FLOPs/latency surrogate based on $\frac{\ell_1}{\ell_2}$ norm that works even in the presence of batchnorm, layernorm. Our approach applies to a large class of efficient building blocks – like unstructured sparsity, neuron pruning, quantization – for which we can express the FLOPs of the model with a $\frac{\ell_1}{\ell_2}$ surrogate (see Table 1). While our surrogates are continuous, they are non-differentiable. In such cases standard optimizers such as SGD, Adam can be quite slow to converge [51]. To overcome this, we propose a projection operation on the mask variables, after each SGD step. Our proposed method speeds up the convergence and also outputs *exact sparse solutions* thus eradicating need for post-hoc thresholding. Finally, our approach is much faster than SOTA blackbox optimization techniques and runs in almost the same amount of time as single model training run.

We implement our algorithm with multiple building blocks including pruning, low-rank factorization, quantization, and apply it on multiple problems in the domain of image classification and NLP. In particular, we demonstrate the effectiveness of our technique for MobileNetV3 compression on ImageNet (see Fig. 1), where our method can learn an architecture with up to 15% (11%) lower FLOPs (latency) on Pixel 6 mobile phones, without any drop in accuracy. Here our approach is more accurate than MorphNet, a SOTA technique which focuses exclusively on neuron-pruning, as well as, TuNAS, a SOTA NAS technique. Furthermore, in terms of training time, our method is $3\times$ cheaper than TuNAS. We would like to highlight that MobileNetv3 is a highly optimized architecture found using efficient NAS techniques [24], and our technique is able to compress this architecture further.

One exciting application of our work is that we can apply it to optimize certain "foundational" baseline models for individual fine-tuning tasks. For example, for compression of BERT on GLUE benchmarks, our method achieved $40-50\%$ reduction in FLOPs with only 1% drop in accuracy (see Fig 1). Moreover, our technique dominates standard model compression baselines. Similarly for smaller vision classification tasks, our technique compresses MobileNetV3, EfficientNet suite of architectures and identifies $1.2\times$-$1.4\times$ cheaper architectures without significant loss in accuracy (see Figure 5). Our technique also outperforms SOTA model compression techniques for ResNet by upto 1.5% on ImageNet (see Figure 4) We would like to note that all these results are obtained at almost the same cost as that of training a single model for the task. Finally, we also demonstrate the versatility of our method by using it to quantize a CNN on CIFAR-10, and learning optimal bitwidths for each of its layers. Our technique found a model that is 55% smaller than the baseline float-16 model, while achieving the same accuracy (see Figure 6). Here is a summary of our contributions:

(1). We provide an end-to-end neural network compression technique that directly optimizes the FLOPs regularized objective leading to compression during training. Our algorithm can be used with many popular efficient building blocks including pruning, low-rank factorization, quantization, and can optimize for on-device inference latency.

(2). We design a novel $\frac{\ell_1}{\ell_2}$ regularized surrogate for latency that works even in the presence of batchnorm, layernorm. We also provide a a simple algorithm for solving this objective. Our algorithm is fast and runs in the same amount of time as single model training, and doesn't require any post-processing steps.

(3). We demonstrate the performance of our technique on both language and vision tasks. Moreover, for transfer learning settings where the goal is to take a baseline architecture and optimize it for individual tasks, our techniques outperform SOTA techniques in the broad-domain of automated neural compression.

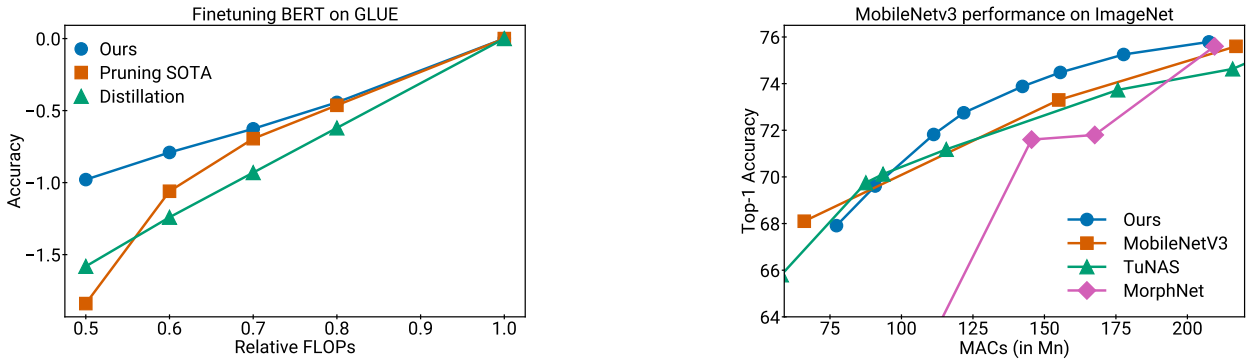

Figure 1. Left plot compares various techniques for BERT compression on GLUE tasks (averaged across tasks). x-axis is the relative number of FLOPs as compared to $\text{BERT}_{\text{BASE}}$. y-axis is the relative drop in accuracy from the baseline. Pruning SOTA numbers are taken from [33], while distillation baselines are from [57, 62]. Right plot compares various techniques for MobileNetV3 compression on ImageNet-1K dataset. *MobileNetV3* corresponds to MobileNetV3 models with different width multiplier. *TuNAS, MorphNet* are SOTA techniques for scalable compression. TuNAS takes a blackbox approach to model compression, whereas MorphNet takes a more direct approach by optimizing FLOPs regularized objective.

2. Related Work

2.1. Neural Architecture Search

Early works on NAS treated the problem as a purely blackbox optimization (BO) problem. These works relied on BO techniques such as random search [35], Gaussian process optimization [28], and zeroth-order gradient descent [64, 88], evolutionary algorithms to optimize the NAS objective and identify a good architecture. Several works have improved upon these algorithms using heuristics such as early stopping [35]. Nonetheless, these techniques are computationally expensive, as evaluating the optimization objective at any point requires training a neural network from scratch. Moreover, due to computational complexity, these techniques perform a very coarse grained search and are not suited for fine-grained search over sparsity or low-rank structures.

One-Shot NAS - Recent works have tried to open the blackbox a bit. These techniques, termed as One-Shot NAS, aim to return the searched architecture as well as its optimal weights in a single pass. In these techniques, the search space is first transformed to the space of probability distributions over architectures. Next, a surrogate model is trained to quickly evaluate the optimization objective at any input [2, 9, 41, 45, 52]. While these techniques are fast, they involve joint training of the surrogate model during the search process. This joint training often makes the optimization process unstable [16]. Since our method uses a gradient descent like paradigm, it sidesteps such issues. Further, prior work has shown evidence that such auxiliary models do not often correlate with the actual model performance [53, 81, 84, 86] in various settings.

Zero-Cost Proxies - There have also been techniques which look at data-independent zero-cost proxies for estimating the performance and latency of a network. These rely on proxy tasks[36, 71] to come up with an estimate of the actual performance. However, recent work has shown that simple baselines such as "number of parameters" and "FLOPs" are surprisingly competitive with all leading techniques [75]. The main downsides of using zero-cost proxies are that they may be unreliable, especially on larger search spaces [74, 75]. They also may have biases, such as preferring larger models [50] or wide channels [10]. It has been shown that zero-cost proxies for CNNs do not transfer well to transformers[87]. In contrast, our method provides a simple regularizer and training recipe which can be applied to a wide range of base architectures and tasks, as we demonstrate in our experiments. We further refer the reader to a recent survey[75] for a more thorough view on the landscape of NAS.

Hardware-aware NAS for Efficient ML Several recent works at the intersection of efficient ML and NAS have realized the importance of explicitly accounting for the hardware in the search process [4, 8, 12, 15, 38, 64, 85]. These works incorporate the actual inference time in their search objectives, instead of surrogates such as FLOPs. The inference time maybe estimated using another neural network [89], or through latency tables for basic arithmetic operations on the target platform [79]. Many of these works rely on greedy, random search heuristics to solve the resulting objective [15, 38]. However, these heuristics either take a lot of time to find the optimal architecture or are not guaranteed to converge to an optimal solution. There are some works that rely on the NAS algorithms described above [2, 12, 64]. However, these techniques face the same scalability and optimization issues as previously mentioned.

2.2. Model Compression

The field of model compression is vast. Here, we focus on techniques that perform training-time compression (as opposed to post-training compression) using the following building blocks: unstructured sparsity, pruning and low-rank factorization. Early works in unstructured sparsity and pruning relied on magnitude, gradient based pruning [17, 18, 21]. Several works have explored more sophisticated scoring metrics for pruning [14, 20, 30, 46, 47]. Other techniques include adding sparsity inducing norms such as $\ell_0, \ell_1$to the training objective [32, 43, 66]. A number of works have also explored low-rank factorization for model compression [25, 27, 44, 77]. Some of these techniques again rely on sparsity inducing regularizers to induce the low-rank structure [25, 72]. Others rely on SVD based pruning. Some recent works try and optimize FLOPs regularized objective to perform pruning, low-rank factorization [9, 19]. However, as we discussed in the introduction, the resulting optimization techniques are often unstable and difficult to use in practice, in particular due to the large number of hyper-paramters needed by them. There have also been specialized methods developed for particular architecture types and modalities. [82] present a unified compression framework for vision transformers, and [59] present a similar pruning framework for multiple modalities. While our method is similar to these works, we note that our method can work across architecture types, modalities and training paradigms, and is agnostic to particular quirks of each of these domains.

3. Method

In this section, we describe our approach for model compression. For simplicity of presentation, we illustrate our technique on feed-forward networks and restrict ourselves to structured pruning. The ideas here can be extended to other architectures (*e.g.,* 1x1 convolutions in CNNs), and other efficient building blocks (*e.g.,* unstructured sparsity, low-rank factorization, quantization) in a straightforward manner (see Table 1 for details).

3.1. Regularizing the FLOPs

Consider the following problem: we are given a pre-trained feed forward neural network (FFN) $f^*(x) = \sigma(W_D^*\sigma(W_{D-1}^*\sigma(\dots\sigma(W_1^*x))))$, where $W_i^* \in \mathbb{R}^{d_{i+1}\times d_i}$ for all $i \in [D]$, and a dataset $\{(x_i, y_i)\}_{i=1}^n$. Our goal is to compress f^* while simultaneously performing well on the learning task. This problem can be formulated as the following optimization problem

$$\min_{\mathcal{W}} \frac{1}{n}\sum_{i=1}^{n} \ell(x_i, y_i; \mathcal{W}) + \lambda \times \text{Latency}(\mathcal{W}). \tag{1}$$

Here $\mathcal{W} = \{W_i\}_{i=1}^D$, with $W_i \in \mathbb{R}^{d'_{i+1}\times d'_i}$ being the weight matrix at layer i, λ is the regularization parameter which trades-off latency with accuracy and ℓ is the supervised loss. Directly optimizing the above objective is intractable because Latency$(\mathcal{W})$ is a discrete function of the dimensions of weight matrices, and is hardware specific.

We now present a technique for solving Equation (1). To begin with, we substitute Latency$(\mathcal{W})$ with FLOPs$(\mathcal{W})$[†]. In App B.1, we extend it to actual latency. The objective in this case is given by

$$\min_{\mathcal{W}} \frac{1}{n}\sum_{i=1}^{n} \ell(x_i, y_i; \mathcal{W}) + \lambda \sum_{i=1}^{D} d'_i d'_{i+1}. \tag{2}$$

To solve this objective, we associate masks with each neuron in the network. In particular, we parameterize the weight matrix in the i^{th} layer as $W_i \times \text{diag}(\alpha_i)$. Here $\alpha_i \in \{0,1\}^{d_i}$ are the mask variables of layer i. If $\alpha_{i,j}$ is set to 0, then the j^{th} neuron in the $(i-1)^{th}$ layer will be pruned. The FLOPs regularizer[†] can now be written in terms of masks as $\sum_{i=1}^{D} \|\alpha_i\|_0\|\alpha_{i+1}\|_0$, where α_{D+1} is the static vector of all 1's. To make this objective continuous and amenable to gradient based optimization, one class of techniques place a Bernoulli distribution Bern$(p_{i,j})$ over each of the masks $\alpha_{i,j}$ and solve the resulting smoothed objective where expectation is taken w.r.t. the random masks α_i's [9, 43, 72] The resulting problem is NP-hard, and the discrete nature of α_i's makes the optimization unstable. To overcome this, [9, 43, 72] rely on a heuristic which involves relaxing Bernoulli distribution to a continuous distribution such as LogisticSigmoid. However, the main drawback of the resulting algorithm is that it is hard to implement in practice and requires very careful annealing of the parameters of LogisticSigmoid distribution. Further, the performance of such techniques is not well understood theoretically, even for simple and fundamental problems such as sparse linear regression.

Another common approach to convert the discrete objective in Equation (2) into a continuous function is to replace the ℓ_0 norm on α_i's with ℓ_1 norm

$$\min_{\mathcal{W}, \alpha_i \in \mathbb{R}^{d_i}} \frac{1}{n}\sum_{i=1}^{n} \ell(x_i, y_i; \alpha, \mathcal{W}) + \lambda \sum_{i=1}^{D} \|\alpha_i\|_1\|\alpha_{i+1}\|_1. \tag{3}$$

This approach is both theoretically grounded [49, 65] and easier to implement in practice [51, 83]. Consequently, recent SOTA compression techniques relied

[†]FLOPs is also a discrete function of dimensions of W_i, and the resulting optimization problem is still intractable.

[†]The expression we write here actually corresponds to the Multiply-Accumulate Operations (MACs). Each MAC usually corresponds to two FLOPs. However, we abuse notation slightly and use FLOPs throughout the paper, since this term is more widely used in prior literature.

on ℓ_1 norm surrogates to compute the FLOPs regularizer [19, 59]. A major drawback of ℓ_1 norm though is that it does not promote sparsity in the presence of batch normalization and layer normalization [1, 26]. To see this, consider the following 1-hidden layer network: $\sigma(\text{BN}(W_2\text{diag}(\alpha_2)\sigma(\text{BN}(W_1\text{diag}(\alpha_1)x))))$. One can scale down all entries of α_1 and scale up the weights W_1 without affecting the output of the network. Doing this reduces the objective value in Equation (3), but doesn't induce any sparsity in the network. In practice, we in fact notice this behaviour during optimization of Equation (3), which leads to sub-optimal solutions. We demonstrate this phenomenon empirically in Section 3.3. Note that adding ℓ_2 penalty on the weights (*i.e.,* weight decay) doesn't mitigate this issue as any scaling of α's can be absorbed by the batch norm parameters without changing the output of the network. Further, such approaches also need a post-training thresholding step on the masks to achieve sparsity in practice, adding another hyper-parameter to the method.

3.2. Inducing sparsity through $\frac{\ell_1}{\ell_2}$ regularizer

We now introduce our approach for making the objective in Equation (2) continuous. Instead of using ℓ_1 as a proxy, we replace ℓ_0 norm over masks ($\|\alpha_i\|_0$) with $\frac{\ell_1}{\ell_2}$ penalty ($\sqrt{d_i}\|\alpha_i\|_1/\|\alpha_i\|_2$) and solve the following optimization problem

$$\min_{\mathcal{W},\alpha_i\in\mathbb{R}^{d_i}} \frac{1}{n}\sum_{i=1}^{n}\ell(x_i,y_i;\alpha,\mathcal{W}) + \lambda\sum_{i=1}^{D}\frac{\sqrt{d_i d_{i+1}}\|\alpha_i\|_1}{\|\alpha_i\|_2}\frac{\|\alpha_{i+1}\|_1}{\|\alpha_{i+1}\|_2} \tag{4}$$

The $\sqrt{d_i}$ term in the numerator normalizes the penalty to lie between $[0, d_i]$. When α_i's are all 1's, the regularizer evaluates to FLOPs. Observe that this regularizer is invariant to scaling of α's. Consequently, the value of the regularizer cannot simply be reduced by scaling down α_i's. In our experiments in Sections 3.3 and 4.3, we show that this handles batch, layer normalizations better than ℓ_1 regularizer. Several works have studied this regularizer in the context of sparse linear regression and showed that is recovers the underlying sparse signal under mild conditions on the data [54, 70, 80]. [78] and [13] used a similar $\frac{\ell_1}{\ell_2}$ regularizer for network pruning, but their techniques don't optimize latency or FLOPs, and rely on post-training thresholding to get sparsity.

For certain technical reasons described in the next paragraph, we add a positivity constraint on α_i's and solve the above objective Note that we consider $\alpha_i \in \mathbb{R}^{d_i}_+$ rather than discrete or bounded values. We would like to highlight that this change doesn't reduce the representational power of our model. It is mainly done for computational reasons.

Importance of positivity constraints. The objective in Equation (4) is continuous, but not smooth. For such losses, standard optimization techniques such as SGD, Adam are slow to converge to stationary points [6]. Furthermore, these algorithms don't output exact sparse solutions. This forces additional post-processing steps to be introduced into the compression pipeline. For example, [19, 78] rely on Adam optimizer and add a pruning step at the end, where masks that are close to 0 are pruned away. This is quite cumbersome in practice as one needs to choose appropriate thresholds for pruning, which introduces an additional tunable hyper-parameter, and needs re-training after pruning. To overcome this, we add a positivity constraint to the mask variables and modify the objective to replace the ℓ_1 norm accordingly. This makes the regularizer smooth (except at all 0's vector), and easy to optimize using SGD, Adam. After each SGD/Adam update, we simply project the masks back to the space of positive real numbers. The overall update looks as follows

$$\mathcal{W} \leftarrow \mathcal{W} - \eta\nabla_{\mathcal{W}}(\mathcal{L}(\alpha,\mathcal{W}) + \lambda\mathcal{R}(\alpha)),$$
$$\alpha \leftarrow \max(0, \alpha - \eta\nabla_{\alpha}(\mathcal{L}(\alpha,\mathcal{W}) + \lambda\mathcal{R}(\alpha))).$$

Here $\mathcal{L}(\alpha,\mathcal{W})$ is the empirical risk and $\mathcal{R}(\alpha)$ is the regularizer. Notice, the only additional step compared to traditional optimization, is the clipping of α's. In our ablation studies in Sections 3.3 and 4.3, we validate the importance of this projection step, together with $\frac{\ell_1}{\ell_2}$ norm, in encouraging sparse solutions.

Hardware-aware compression - While we deal with FLOPs in this section, our method can also be extended to optimize the actual latency. We model the on-device latency as a sum of latencies of the individual matrix multiplications involved in the model. The latencies are looked up from a linearly interpolated latency table constructed from on-device measurements. The $\frac{\ell_1}{\ell_2}$ regularizer is crucial to this interpolation, as it is normalized and lies between $[0, d_i]$, leading to easy lookups. In our experiments, we perform on-device latency evaluations using the CPU on the Pixel6 phone as well. We refer the reader to Appendix B.1 for more details on our approach.

3.3. Verification of design choices

To empirically demonstrate the drawbacks of using ℓ_1 penalty for model compression, we perform experiments on the FashionMNIST dataset with a single hidden layer fully connected network which has a batch norm layer after the first linear layer. We prune out the input to the network using a mask α on the input. We compare the performance of networks compressed using FLOPs regularizer induced by ℓ_1 and $\frac{\ell_1}{\ell_2}$ norms. We use SGD for optimization of both the objectives. Furthermore, we pre-train the network using standard CE loss, and initialize $\alpha = \mathbf{1}$. We track the variance of the absolute values of the entries of α, i.e. $\frac{\sum_{i=1}^{d}(|\alpha_i|-\mu_\alpha)^2}{d}$, where $\mu_\alpha = \frac{\sum_{i=1}^{d}|\alpha_i|}{d}$. We also

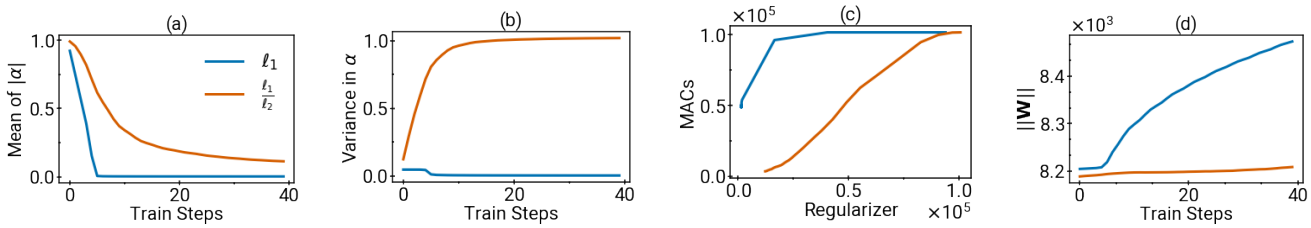

Figure 2. **Comparison of ℓ_1, $\frac{\ell_1}{\ell_2}$ induced FLOPs regularizer for pruning on FashionMNIST**: Figures (a) and (b) depict the evolution of the statistics of the mask variables (α) as training progresses. Figure (c) shows the relation between the actual FLOPs of the model and the value of the proxy computed by Equations 3, 4. Figure (d) shows the evolution of the Frobenius norm of the weight matrix.

track the mean μ_α of the absolute values of the entries of α. Finally, we plot out the curve between FLOPs and the considered norm of α (*i.e.,* ℓ_1, $\frac{\ell_1}{\ell_2}$). Figure 2 presents the results from these experiments. We can see that the ℓ_1 objective is mis-aligned with the actual value of FLOPs, while the regularizer computed using $\frac{\ell_1}{\ell_2}$ is a better proxy. We also find that the mean and variance of α's sharply decreases when ℓ_1 induced FLOPs regularizer is used for compression. This indicates that all entries of α are uniformly scaled down to a small, non-zero value, reducing the value of the regularizer, while not providing any sparsity. As seen from the figure, $\frac{\ell_1}{\ell_2}$ does not suffer from this drawback. Finally, we note that the frobenius norm of the weight matrix W increases when ℓ_1 regularization is used on α, suggesting that the network is simply scaling down α's and scaling up the weights to evade the regularizer.

4. Experiments

In this section, we apply our framework to large scale pre-training and transfer learning tasks on standard language and vision benchmarks. To demonstrate the versatility of our technique, we perform experiments on multiple model families (MobileNet, EfficientNet, ResNet, BERT), and multiple building blocks (pruning, low-rank factorization). Note that in this section, we provide accuracy v/s MACs (Multiply-Accumulate operations) trade-off for various tasks[†]. Since we focus on *structured pruning*, a decrease in FLOPs (or MACs) would correlate with a decreased latency as well. In addition to this, in Sec 4.3, we also present experiments using the actual on-device latency instead of FLOPs and show that our searched models are indeed faster on device. Further, we also present a case study integrating quantization into our framework in Appendix A.1, demonstrating its versatility.

4.1. Large scale classification on ImageNet

MobileNet Family - We begin by comparing the performance of our technique with baselines on MobileNetV3 compression, for ImageNet classification. We rely on low-rank factorization + pruning for the compression. The results from this experiment are presented in Figure 1. By varying the strength of our regularization, we obtain models with different MACs and accuracies. We find that models produced by our method significantly outperform MobileNetV3 and TuNAS in the high and mid-MACs regime. In particular, for the same accuracy as MobileNetV3Large, our approach finds a model with 15% fewer MACs. In comparison with TuNAS, we achieve 30% reduction in MACs at the same level of accuracy. We however find that our model is at par with MobileNetV3Small in the low MACs regime, indicating that the former is already well-tuned for this task. In terms of compute needed for training, TuNAS is the most expensive among all the techniques we tried; it took 2 days to train with our hardware setup. In contrast, our method took 13 hours ($3-4\times$ faster than TuNAS), and MorphNet took 10 hours. Note that MobileNetv3 is a highly compressed model for edge deployment, and previous works have found it challenging to compress the model further. Our method can still provide a better FLOPs v/s accuracy trade-off, providing evidence for its efficacy.

ResNet Family - We also compress the ResNet architecture for ImageNet classification, using our method. In particular, we compress the 1×1 convolutions using pruning and low-rank factorization. We compare our method against HALP [58] and PAS [37], two state of the art methods for neural architecture search and model compression for ResNet. Our method compresses ResNet-101 to a model with similar FLOPs as ResNet-50, while simultaneously achieving better performance than the baseline ResNet-50. Furthermore, our technique outperformns SOTA methods for the same number of FLOPs by up to 1.5%, as seen in Fig 4.

4.2. Transfer Learning

A common paradigm in deploying machine learning models today is to first pre-train them on a large scale dataset such as ImageNet, and then fine-tune them for the desired target task. However, deploying large models is not feasible on edge devices. Our technique provides a light-weight modification to the standard fine-tuning procedure by producing

[†]Note that while we use the term "FLOPs" to describe computational cost in the paper, we report MACs for computer vision models, in line with prior work.

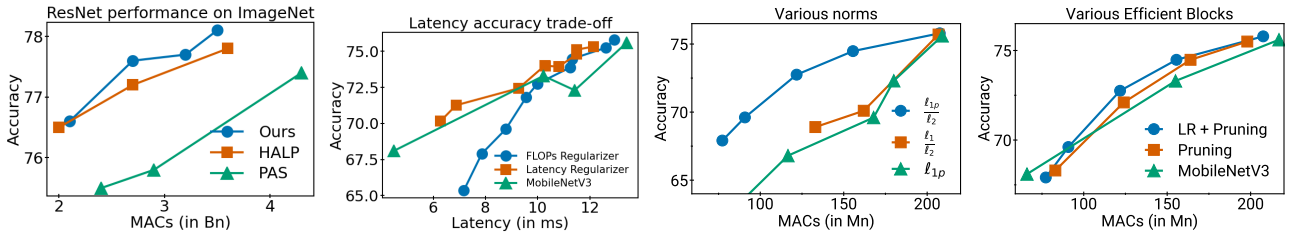

Figure 4. **(a) Pruning ResNet on ImageNet** - We compare against HALP and PAS, two recent SOTA techniques to prune ResNet-50, and achieve better performance over different FLOP regimes. **(b),(c),(d) - Ablation studies on Mobilenetv3** We compare using ℓ_1 and $\frac{\ell_1}{\ell_2}$ norms in our regularizer, with subscript p indicating that projected-Adam was used for optimization. We also experiment with combining low-rank (LR) factorization with channel pruning. Finally, we show on device latency-accuracy tradeoff with using the actual latency regularizer for compressing MobileNetv3

a highly compressed model with comparable transfer learning performance on the specific task. We demonstrate this on vision and language tasks.

Vision tasks. We consider the task of fine-tuning an ImageNet pre-trained model for a smaller dataset. We consider Cars196 [31] and Food101 [5] as the target datasets, and compare against the MobileNetV3 and EfficientNet families of models. We use ImageNet pre-trained models for initialization. We plot the FLOP-accuracy curves in Fig 5. We compress MobileNetv3Large and EfficientNet-B4 and EfficientNet-B2 architectures while fine-tuning them on the target target task. We find that our method consistently improves over baseline architectures across various FLOPs regimes. This is because our technique is able to adaptively prune the model based on the difficulty of the classification task. On both the tasks, we see 1% accuracy gains over MobileNetV3 small. The accuracy gains persist at the latency footprint of MobileNetV3Large-0.75, where we see over 1.5% accuracy gains on both datasets. On EfficientNet, we see upto 40% reduction in FLOPs without any drop in accuracy on Food101, and around 20% reduction in FLOPs on the Cars196 dataset for the largest models (B4). We also see around 30% FLOP reduction while maintaining the transfer learning performance of the B1 and B0 variants. This demonstrates that our learnt models can scale better than the heuristic scaling described in [63].

Fine-tuning BERT on GLUE. We consider 5 datasets of the GLUE benchmark [69] that are commonly used in the literature, and fine-tune a pre-trained BERT-Base model with our FLOPs regularizer. We re-parameterize the weight matrices of the feed forward network of each transformer block with our low-rank+sparse parameterization. We compare our approach against model pruning, where SOTA numbers are taken from Fig. 6 of [33], reporting the maximum accuracy among [34, 40, 42, 56, 73, 76]. We also report the performance of widely-used distillation based baselines [57, 62]. Figure 1 presents the average performance on the 5 datasets, and Figure 8 in appendix presents the performance on each dataset. In both these figures, we plot the relative non-embedding FLOPs of the compressed model w.r.t BERT-base against the drop in accuracy w.r.t BERT-base (similar to [33]). We find that on 4 of the 5 datasets considered, our technique provides a higher accuracy for the same number of FLOPs, indicating the efficacy of our method. On MRPC, a dataset with very few samples, our method is worse off for models with higher FLOPs, but outperforms the baselines in the low FLOP regime.

4.3. Ablation Studies on MobileNetv3

Effect of optimization choices. In section 3 we provided small scale experiments to justify our design choices of using projected-Adam and $\frac{\ell_1}{\ell_2}$ norm. In this section we perform large-scale ablation studies on MobileNetV3 for ImageNet training. The results from this experiment are presented in Figure 4. Without projected-Adam, we notice that the optimization algorithm doesn't converge to sparse solutions. Consequently, the resulting models do not have large reduction in MACs. The accuracy of these models also takes a big hit. On the other hand, using ℓ_1 norm based FLOPs regularizer with projected-Adam suffers from the scaling issue described in Sec 3.3. This leads to a large fraction of channels being pruned for some blocks, producing a model with deteriorated accuracy. Our method has 2-4% better accuracy in the high and mid FLOPs regimes than these alternatives.

Comparing different building blocks. In Table 1, we described ways to integrate various building blocks into our framework. In Figure 4, we demonstrate the accuracy *vs* inference time trade-offs of using two of these building blocks in our framework, namely Pruning and Pruning+Low-rank Factorization. We find that the extra flexibility provided by the Low-Rank Factorization leads to models with fewer MACs for the same accuracy, and the difference is even more pronounced for smaller models. We note that channel pruning alone can give us 10% reduction in MACs over the MobileNetV3 family at the same accuracy level. In particular, at 73.4% accuracy, our model has 136Mn MACs compared to 155Mn MACs of the MobileNetV3 family model.

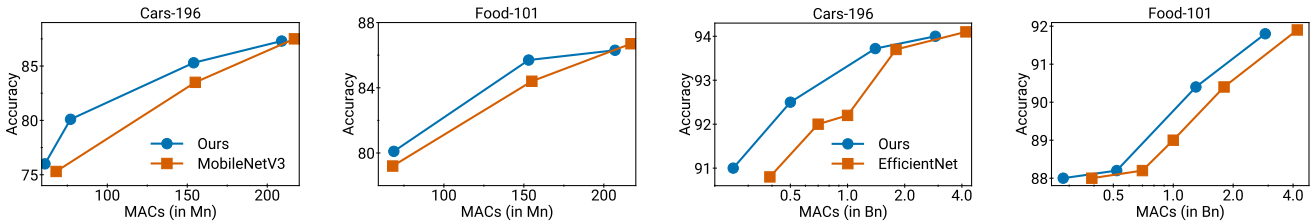

Figure 5. **Accuracy-FLOPs trade-off on vision transfer learning tasks**: Figures (a) and (b) depict the the fine-tuning performance of models found by our method while compressing MobileNetv3Large and baseline MobileNetV3 on Cars-196 and Food-101 datasets. Figures (c) and (d) show the performance on the EfficientNet family of architectures, where baselines are EfficientNetB0-B4, while our method compresses EfficientNet B4 and B2.

Similarly, at 75.5% accuracy, our model has 198Mn MACs compared to 216Mn MACs of MobileNetV3 family model. Adding Low-Rank structure introduces another 5% reduction in MACs, with no loss in accuracy. This also shows the effectiveness of our algorithm across multiple building blocks.

Hardware-aware compression. We now optimize for actual on-deive latency by considering latency based ℓ_1/ℓ_2 surrogates (see Eq 6 in Appendix for more details on the surrogate). We provide empirical evidence on the effectiveness of this approach for MobileNetV3 on Pixel 6. We measure the latency on the device's CPU. We use out-of-the box models, and do not quantize or add any other latency optimizations for these. We compare the accuracy-latency curves of models produced using FLOPs, latency regularizers (see Fig 4). Observe that using the latency regularizer leads to models with smaller latencies and consequently better latency-accuracy tradeoff compared to using the FLOP regularizer. We also find these models to have better performance than MobileNetV3 ($0.5 - 2\%$ improvement in accuracy for similar latency), despite MobileNetV3 being hand-crafted for faster inference on mobile devices. Note that latencies here are actual on-device inference latencies of the models.

5. Conclusion and Future Work

In this work, we presented an end-to-end technique for neural network compression. Our approach applies to a wide variety of efficient blocks including pruning, unstructured sparsity, quantization. At the core of our algorithm is a novel surrogate for FLOPs, latency that relies on $\frac{\ell_1}{\ell_2}$ norms, and works with batchnorm, layernorm. Our algorithm is computationally efficient and runs in same amount of time as needed for training a single model. We demonstrated the efficacy of our approach on various pre-training and transfer learning tasks on standard language and vision benchmarks. As a future work, it will useful to incorporate more efficient building blocks such as block diagonal matrices into our framework. Another interesting direction would be to make our technique more hardware aware by incorporating hardware level parameters such as tiling into our search process.

References

[1] Jimmy Lei Ba, Jamie Ryan Kiros, and Geoffrey E Hinton. Layer normalization. *arXiv preprint arXiv:1607.06450*, 2016. 5

[2] Gabriel Bender, Hanxiao Liu, Bo Chen, Grace Chu, Shuyang Cheng, Pieter-Jan Kindermans, and Quoc V Le. Can weight sharing outperform random architecture search? an investigation with tunas. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 14323–14332, 2020. 1, 3

[3] Yoshua Bengio, Nicholas Léonard, and Aaron Courville. Estimating or propagating gradients through stochastic neurons for conditional computation. *arXiv preprint arXiv:1308.3432*, 2013. 13

[4] Hadjer Benmeziane, Kaoutar El Maghraoui, Hamza Ouarnoughi, Smail Niar, Martin Wistuba, and Naigang Wang. A comprehensive survey on hardware-aware neural architecture search. *arXiv preprint arXiv:2101.09336*, 2021. 3

[5] Lukas Bossard, Matthieu Guillaumin, and Luc Van Gool. Food-101 – mining discriminative components with random forests. In *European Conference on Computer Vision*, 2014. 7

[6] Stephen Boyd, Stephen P Boyd, and Lieven Vandenberghe. *Convex optimization*. Cambridge university press, 2004. 5

[7] Cristian Bucila, Rich Caruana, and Alexandru Niculescu-Mizil. Model compression. In *Knowledge Discovery and Data Mining*, 2006. 1

[8] Han Cai, Ligeng Zhu, and Song Han. Proxylessnas: Direct neural architecture search on target task and hardware. *arXiv preprint arXiv:1812.00332*, 2018. 1, 2, 3

[9] Shraman Ray Chaudhuri, Elad Eban, Hanhan Li, Max Moroz, and Yair Movshovitz-Attias. Fine-grained stochastic architecture search. *arXiv preprint arXiv:2006.09581*, 2020. 2, 3, 4

[10] Hanlin Chen, Ming Lin, Xiuyu Sun, and Hao Li. NAS-bench-zero: A large scale dataset for understanding zero-shot neural architecture search, 2022. 3

[11] Weihan Chen, Peisong Wang, and Jian Cheng. Towards mixed-precision quantization of neural networks via constrained optimization. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 5350–5359, 2021. 13

[12] Grace Chu, Okan Arikan, Gabriel Bender, Weijun Wang, Achille Brighton, Pieter-Jan Kindermans, Hanxiao Liu, Berkin Akin, Suyog Gupta, and Andrew Howard. Discovering multi-hardware mobile models via architecture search. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 3022–3031, 2021. 3

[13] Enmao Diao, Ganghua Wang, Jiawei Zhan, Yuhong Yang, Jie Ding, and Vahid Tarokh. Pruning deep neural networks from a sparsity perspective. *arXiv preprint arXiv:2302.05601*, 2023. 5

[14] Xin Dong, Shangyu Chen, and Sinno Pan. Learning to prune deep neural networks via layer-wise optimal brain surgeon. *Advances in Neural Information Processing Systems*, 30, 2017. 4

[15] Zhen Dong, Yizhao Gao, Qijing Huang, John Wawrzynek, Hayden KH So, and Kurt Keutzer. Hao: Hardware-aware neural architecture optimization for efficient inference. In *2021 IEEE 29th Annual International Symposium on Field-Programmable Custom Computing Machines (FCCM)*, pages 50–59. IEEE, 2021. 3

[16] Thomas Elsken, Jan Hendrik Metzen, and Frank Hutter. Neural architecture search: A survey. *The Journal of Machine Learning Research*, 20(1):1997–2017, 2019. 3

[17] Jonathan Frankle and Michael Carbin. The lottery ticket hypothesis: Finding sparse, trainable neural networks. *arXiv preprint arXiv:1803.03635*, 2018. 4

[18] Trevor Gale, Erich Elsen, and Sara Hooker. The state of sparsity in deep neural networks. *arXiv preprint arXiv:1902.09574*, 2019. 2, 4

[19] Ariel Gordon, Elad Eban, Ofir Nachum, Bo Chen, Hao Wu, Tien-Ju Yang, and Edward Choi. Morphnet: Fast & simple resource-constrained structure learning of deep networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 1586–1595, 2018. 2, 4, 5

[20] Yiwen Guo, Anbang Yao, and Yurong Chen. Dynamic network surgery for efficient dnns. *Advances in neural information processing systems*, 29, 2016. 4

[21] Song Han, Jeff Pool, John Tran, and William Dally. Learning both weights and connections for efficient neural network. *Advances in neural information processing systems*, 28, 2015. 1, 2, 4

[22] 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. 1

[23] Geoffrey Hinton, Oriol Vinyals, Jeff Dean, et al. Distilling the knowledge in a neural network. *arXiv preprint arXiv:1503.02531*, 2(7), 2015. 1, 15

[24] Andrew Howard, Mark Sandler, Grace Chu, Liang-Chieh Chen, Bo Chen, Mingxing Tan, Weijun Wang, Yukun Zhu, Ruoming Pang, Vijay Vasudevan, et al. Searching for mobilenetv3. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 1314–1324, 2019. 2

[25] Yen-Chang Hsu, Ting Hua, Sungen Chang, Qian Lou, Yilin Shen, and Hongxia Jin. Language model compression with weighted low-rank factorization. In *International Conference on Learning Representations*, 2021. 1, 4

[26] 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. PMLR, 2015. 5

[27] Max Jaderberg, Andrea Vedaldi, and Andrew Zisserman. Speeding up convolutional neural networks with low rank expansions. *arXiv preprint arXiv:1405.3866*, 2014. 4

[28] Kirthevasan Kandasamy, Willie Neiswanger, Jeff Schneider, Barnabas Poczos, and Eric Xing. Neural architecture search with bayesian optimisation and optimal transport. *arXiv preprint arXiv:1802.07191*, 2018. 1, 3

[29] Jared Kaplan, Sam McCandlish, Tom Henighan, Tom B Brown, Benjamin Chess, Rewon Child, Scott Gray, Alec Radford, Jeffrey Wu, and Dario Amodei. Scaling laws for neural language models. *arXiv preprint arXiv:2001.08361*, 2020. 1

[30] Ehud D Karnin. A simple procedure for pruning back-propagation trained neural networks. *IEEE transactions on neural networks*, 1(2):239–242, 1990. 4

[31] Jonathan Krause, Michael Stark, Jia Deng, and Li Fei-Fei. 3d object representations for fine-grained categorization. In *4th International IEEE Workshop on 3D Representation and Recognition (3dRR-13)*, Sydney, Australia, 2013. 7

[32] Aditya Kusupati, Vivek Ramanujan, Raghav Somani, Mitchell Wortsman, Prateek Jain, Sham Kakade, and Ali Farhadi. Soft threshold weight reparameterization for learnable sparsity. In *International Conference on Machine Learning*, pages 5544–5555. PMLR, 2020. 1, 2, 4

[33] Woosuk Kwon, Sehoon Kim, Michael W Mahoney, Joseph Hassoun, Kurt Keutzer, and Amir Gholami. A fast post-training pruning framework for transformers. *arXiv preprint arXiv:2204.09656*, 2022. 3, 7

[34] François Lagunas, Ella Charlaix, Victor Sanh, and Alexander Rush. Block pruning for faster transformers. In *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing*, pages 10619–10629, Online and Punta Cana, Dominican Republic, 2021. Association for Computational Linguistics. 7

[35] Liam Li and Ameet Talwalkar. Random search and reproducibility for neural architecture search. In *Uncertainty in artificial intelligence*, pages 367–377. PMLR, 2020. 3

[36] Yuhong Li, Cong Hao, Pan Li, Jinjun Xiong, and Deming Chen. Generic neural architecture search via regression. *Advances in Neural Information Processing Systems*, 34:20476–20490, 2021. 3

[37] Yanyu Li, Pu Zhao, Geng Yuan, Xue Lin, Yanzhi Wang, and Xin Chen. Pruning-as-search: Efficient neural architecture search via channel pruning and structural reparameterization. *arXiv preprint arXiv:2206.01198*, 2022. 6

[38] Ji Lin, Wei-Ming Chen, Yujun Lin, John Cohn, Chuang Gan, and Song Han. Mcunet: Tiny deep learning on iot devices. *arXiv preprint arXiv:2007.10319*, 2020. 3

[39] Tao Lin, Sebastian U Stich, Luis Barba, Daniil Dmitriev, and Martin Jaggi. Dynamic model pruning with feedback. *arXiv preprint arXiv:2006.07253*, 2020. 2

[40] Zi Lin, Jeremiah Liu, Zi Yang, Nan Hua, and Dan Roth. Pruning redundant mappings in transformer models via spectral-normalized identity prior. In *Findings of the Association for Computational Linguistics: EMNLP 2020*, pages 719–730, Online, 2020. Association for Computational Linguistics. 7

[41] Hanxiao Liu, Karen Simonyan, and Yiming Yang. Darts: Differentiable architecture search. *arXiv preprint arXiv:1806.09055*, 2018. 1, 3

[42] Zejian Liu, Fanrong Li, Gang Li, and Jian Cheng. EBERT: Efficient BERT inference with dynamic structured pruning. In *Findings of the Association for Computational Linguistics: ACL-IJCNLP 2021*, pages 4814–4823, Online, 2021. Association for Computational Linguistics. 7

[43] Christos Louizos, Max Welling, and Diederik P Kingma. Learning sparse neural networks through l_0 regularization. *arXiv preprint arXiv:1712.01312*, 2017. 2, 4

[44] Zhiyun Lu, Vikas Sindhwani, and Tara N Sainath. Learning compact recurrent neural networks. In *2016 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 5960–5964. IEEE, 2016. 4

[45] Jieru Mei, Yingwei Li, Xiaochen Lian, Xiaojie Jin, Linjie Yang, Alan Yuille, and Jianchao Yang. Atomnas: Fine-grained end-to-end neural architecture search. *arXiv preprint arXiv:1912.09640*, 2019. 3

[46] Pavlo Molchanov, Stephen Tyree, Tero Karras, Timo Aila, and Jan Kautz. Pruning convolutional neural networks for resource efficient inference. *arXiv preprint arXiv:1611.06440*, 2016. 4

[47] Pavlo Molchanov, Arun Mallya, Stephen Tyree, Iuri Frosio, and Jan Kautz. Importance estimation for neural network pruning. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 11264–11272, 2019. 4

[48] Markus Nagel, Marios Fournarakis, Rana Ali Amjad, Yelysei Bondarenko, Mart van Baalen, and Tijmen Blankevoort. A white paper on neural network quantization. *arXiv preprint arXiv:2106.08295*, 2021. 1

[49] Sahand Negahban, Bin Yu, Martin J Wainwright, and Pradeep Ravikumar. A unified framework for high-dimensional analysis of m-estimators with decomposable regularizers. *Advances in neural information processing systems*, 22, 2009. 4

[50] Xuefei Ning, Changcheng Tang, Wenshuo Li, Zixuan Zhou, Shuang Liang, Huazhong Yang, and Yu Wang. Evaluating efficient performance estimators of neural architectures. In *Advances in Neural Information Processing Systems*, pages 12265–12277. Curran Associates, Inc., 2021. 3

[51] Neal Parikh, Stephen Boyd, et al. Proximal algorithms. *Foundations and trends® in Optimization*, 1 (3):127–239, 2014. 2, 4

[52] Hieu Pham, Melody Guan, Barret Zoph, Quoc Le, and Jeff Dean. Efficient neural architecture search via parameters sharing. In *International Conference on Machine Learning*, pages 4095–4104. PMLR, 2018. 3

[53] Aloïs Pourchot, Alexis Ducarouge, and Olivier Sigaud. To share or not to share: A comprehensive appraisal of weight-sharing, 2020. 3

[54] Yaghoub Rahimi, Chao Wang, Hongbo Dong, and Yifei Lou. A scale-invariant approach for sparse signal recovery. *SIAM Journal on Scientific Computing*, 41(6):A3649–A3672, 2019. 5

[55] Alex Renda, Jonathan Frankle, and Michael Carbin. Comparing fine-tuning and rewinding in neural network pruning. In *International Conference on Learning Representations*, 2020. 2

[56] Hassan Sajjad, Fahim Dalvi, Nadir Durrani, and Preslav Nakov. On the effect of dropping layers of pre-trained transformer models. *Computer Speech I& Language*, 77:101429, 2023. 7

[57] Victor Sanh, Lysandre Debut, Julien Chaumond, and Thomas Wolf. Distilbert, a distilled version of bert: smaller, faster, cheaper and lighter, 2020. 3, 7

[58] Maying Shen, Hongxu Yin, Pavlo Molchanov, Lei Mao, Jianna Liu, and Jose M Alvarez. Halp: hardware-aware latency pruning. *arXiv preprint arXiv:2110.10811*, 2021. 6

[59] Dachuan Shi, Chaofan Tao, Ying Jin, Zhendong Yang, Chun Yuan, and Jiaqi Wang. Upop: Unified and progressive pruning for compressing vision-language transformers, 2023. 4, 5

[60] Richard Shin, Charles Packer, and Dawn Song. Differentiable neural network architecture search. 2018. 1

[61] Helmuth Späth. *One dimensional spline interpolation algorithms*. AK Peters/CRC Press, 1995. 14

[62] Siqi Sun, Yu Cheng, Zhe Gan, and Jingjing Liu. Patient knowledge distillation for bert model compression. *arXiv preprint arXiv:1908.09355*, 2019. 3, 7

[63] Mingxing Tan and Quoc Le. Efficientnet: Rethinking model scaling for convolutional neural networks. In *International conference on machine learning*, pages 6105–6114. PMLR, 2019. 1, 7

[64] Mingxing Tan, Bo Chen, Ruoming Pang, Vijay Vasudevan, Mark Sandler, Andrew Howard, and Quoc V Le. Mnasnet: Platform-aware neural architecture search for mobile. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 2820–2828, 2019. 1, 3

[65] Robert Tibshirani. Regression shrinkage and selection via the lasso. *Journal of the Royal Statistical Society: Series B (Methodological)*, 58(1):267–288, 1996. 4

[66] Rishabh Tiwari, Udbhav Bamba, Arnav Chavan, and Deepak K. Gupta. Chipnet: Budget-aware pruning with heaviside continuous approximations, 2021. 1, 4

[67] Stefan Uhlich, Lukas Mauch, Fabien Cardinaux, Kazuki Yoshiyama, Javier Alonso Garcia, Stephen Tiedemann, Thomas Kemp, and Akira Nakamura. Mixed precision dnns: All you need is a good parametrization. *arXiv preprint arXiv:1905.11452*, 2019. 13

[68] Mart Van Baalen, Christos Louizos, Markus Nagel, Rana Ali Amjad, Ying Wang, Tijmen Blankevoort, and Max Welling. Bayesian bits: Unifying quantization and pruning. *Advances in neural information processing systems*, 33:5741–5752, 2020. 13

[69] Alex Wang, Amanpreet Singh, Julian Michael, Felix Hill, Omer Levy, and Samuel Bowman. GLUE: A multi-task benchmark and analysis platform for natural language understanding. In *Proceedings of the 2018 EMNLP Workshop BlackboxNLP: Analyzing and Interpreting Neural Networks for NLP*, pages 353–355, Brussels, Belgium, 2018. Association for Computational Linguistics. 7

[70] Chao Wang, Ming Yan, Yaghoub Rahimi, and Yifei Lou. Accelerated schemes for the l_1/l_2 minimization. *IEEE Transactions on Signal Processing*, 68: 2660–2669, 2020. 5

[71] Haibin Wang, Ce Ge, Hesen Chen, and Xiuyu Sun. Prenas: Preferred one-shot learning towards efficient neural architecture search. *arXiv preprint arXiv:2304.14636*, 2023. 3

[72] Ziheng Wang, Jeremy Wohlwend, and Tao Lei. Structured pruning of large language models. *arXiv preprint arXiv:1910.04732*, 2019. 1, 2, 4

[73] Ziheng Wang, Jeremy Wohlwend, and Tao Lei. Structured pruning of large language models. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 6151–6162, Online, 2020. Association for Computational Linguistics. 7

[74] Colin White, Mikhail Khodak, Renbo Tu, Shital Shah, Sébastien Bubeck, and Debadeepta Dey. A deeper look at zero-cost proxies for lightweight nas. 3

[75] Colin White, Mahmoud Safari, Rhea Sukthanker, Binxin Ru, Thomas Elsken, Arber Zela, Debadeepta Dey, and Frank Hutter. Neural architecture search: Insights from 1000 papers, 2023. 3

[76] Mengzhou Xia, Zexuan Zhong, and Danqi Chen. Structured pruning learns compact and accurate models. In *Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 1513–1528, Dublin, Ireland, 2022. Association for Computational Linguistics. 7

[77] Yuhui Xu, Yuxi Li, Shuai Zhang, Wei Wen, Botao Wang, Wenrui Dai, Yingyong Qi, Yiran Chen, Weiyao Lin, and Hongkai Xiong. Trained rank pruning for efficient deep neural networks. In *2019 Fifth Workshop on Energy Efficient Machine Learning and Cognitive Computing-NeurIPS Edition (EMC2-NIPS)*, pages 14–17. IEEE, 2019. 4

[78] Huanrui Yang, Wei Wen, and Hai Li. Deephoyer: Learning sparser neural network with differentiable scale-invariant sparsity measures. *arXiv preprint arXiv:1908.09979*, 2019. 5

[79] Tien-Ju Yang, Andrew Howard, Bo Chen, Xiao Zhang, Alec Go, Mark Sandler, Vivienne Sze, and Hartwig Adam. Netadapt: Platform-aware neural network adaptation for mobile applications. In *Proceedings of the European Conference on Computer Vision (ECCV)*, pages 285–300, 2018. 1, 3, 14

[80] Penghang Yin, Ernie Esser, and Jack Xin. Ratio and difference of l_1 and l_2 norms and sparse representation with coherent dictionaries. *Communications in Information and Systems*, 14(2):87–109, 2014. 5

[81] Kaicheng Yu, Christian Sciuto, Martin Jaggi, Claudiu Musat, and Mathieu Salzmann. Evaluating the search phase of neural architecture search. *arXiv preprint arXiv:1902.08142*, 2019. 3

[82] Shixing Yu, Tianlong Chen, Jiayi Shen, Huan Yuan, Jianchao Tan, Sen Yang, Ji Liu, and Zhangyang Wang. Unified visual transformer compression. *arXiv preprint arXiv:2203.08243*, 2022. 4

[83] Jihun Yun, Aurélie C Lozano, and Eunho Yang. Adaptive proximal gradient methods for structured neural networks. *Advances in Neural Information Processing Systems*, 34:24365–24378, 2021. 4

[84] Arber Zela, Julien Siems, and Frank Hutter. Nas-bench-1shot1: Benchmarking and dissecting one-shot neural architecture search. *arXiv preprint arXiv:2001.10422*, 2020. 3

[85] Li Lyna Zhang, Yuqing Yang, Yuhang Jiang, Wenwu Zhu, and Yunxin Liu. Fast hardware-aware neural architecture search. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops*, pages 692–693, 2020. 3

[86] Yuge Zhang, Zejun Lin, Junyang Jiang, Quanlu Zhang, Yujing Wang, Hui Xue, Chen Zhang, and Yaming Yang. Deeper insights into weight sharing in neural architecture search. *arXiv preprint arXiv:2001.01431*, 2020. 3

[87] Qinqin Zhou, Kekai Sheng, Xiawu Zheng, Ke Li, Xing Sun, Yonghong Tian, Jie Chen, and Rongrong Ji. Training-free transformer architecture search. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 10894–10903, 2022. 3

[88] Barret Zoph and Quoc V Le. Neural architecture search with reinforcement learning. *arXiv preprint arXiv:1611.01578*, 2016. 1, 3

[89] Łukasz Dudziak, Thomas Chau, Mohamed S. Abdelfattah, Royson Lee, Hyeji Kim, and Nicholas D. Lane. Brp-nas: Prediction-based nas using gcns, 2021. 3