Activation Density driven Energy-Efficient Pruning in Training

***Abstract*—The process of neural network pruning with suitable fine-tuning and retraining can yield networks with considerably fewer parameters than the original with comparable degrees of accuracy. Typically, pruning methods require large, pre-trained networks as a starting point from which they perform a time-intensive iterative pruning and retraining algorithm. We propose a novel pruning in-training method that prunes a network real-time during training, reducing the overall training time to achieve an optimal compressed network. To do so, we introduce an activation density based analysis that identifies the optimal relative sizing or compression for each layer of the network. Our method removes the need for pre-training and is architecture agnostic, allowing it to be employed on a wide variety of systems. For VGG-19 and ResNet18 on CIFAR-10, CIFAR-100, and TinyImageNet, we obtain exceedingly sparse networks (up to $200\times$ reduction in parameters and $> 60\times$ reduction in inference compute operations in the best case) with comparable accuracies (up to 2%-3% loss with respect to the baseline network). By reducing the network size periodically during training, we achieve total training times that are shorter than those of previously proposed pruning methods. Furthermore, training compressed networks at different epochs with our proposed method yields considerable reduction in training compute complexity ($1.6\times - 3.2\times$ lower) at near iso-accuracy as compared to a baseline network trained entirely from scratch.**

I. Introduction

Deep learning has proliferated in the past decade. Not only has it captured the public's imagination as a candidate for the development of intelligent system, but it has achieved high accuracy on difficult datasets. Particularly, deep networks have performed well on computer vision tasks and natural language processing [1], [2]. Part of their success has been attributed to the networks' depths—typical deep networks can comprise of hundreds of layers—but this comes with the cost of having a huge number of trainable parameters [3].

The concept of *network pruning*—systematically reducing the number of parameters in a given network configuration—has been around since the early 1990s [4], but it has only recently begun to receive widespread attention. Over the past five years, many network pruning strategies have been proposed and the motivations for pruning have been explored [3], [5]–[10]. Iandola et al. [11] have identified three ways in which pruned architectures are superior to the networks from which they were created: they are more efficiently trained on distributed systems, their smaller model size makes them easier to send to new clients (a self-driving car, for example), and they are more suited for deployment on edge devices such as mobile phones or embedded processors.

Most network pruning algorithms that have been proposed follow the same structure: *1) Train a large network to a high degree of accuracy, 2) Prune the model architecture while preserving weight values, and 3) Make small adjustments to the pruned model as needed*. Typically, *step 3* consists of training the pruned model for a few epochs, having taken the weights from the large network as the starting configuration. The need for a large pre-trained network slows down the entire pruning process, since it is often computationally expensive and time consuming to train large models. The necessity of *step 1* comes from the assumption that the pruned network will train better if it is initialized using the weights of its high-performing parent network than if it is randomly initialized. Furthermore, most pruning strategies rely on significance of the weight values (say, L1 norm [12]) to decide if the connection should be pruned or not. Thus, a pre-trained network is necessary in such cases to assign significance.

However, Liu et al. [3] have shown that it is not necessary to preserve weight values when moving from a large model to a smaller model. They found a negligible discrepancy between fine-tuning a pruned model and training that same model from randomly initialized weights. This implies that, in order to successfully reduce the size of a large network without a significant loss in accuracy, it is sufficient to find only the optimal architecture of the pruned network. Finding the significant weights to prune or initialize a compressed network (as in [13], [6], and [11]) is not always necessary.

Our contributions We propose an in-training pruning method that analyzes the network performance in real-time and optimizes the architecture throughout the training process. Unlike other pruning methods, the method presented here does not require a pre-trained network. It relies on the scalar metric of activation density per layer to make decisions regarding the pruning or compression criteria of each layer. This allows for a fewer number of iterations during the training process with a comparable parameter and inference compute operations (OPS) reduction to other proposed pruning methods. Our method also yields considerable reduction in training compute OPS as compared to a model trained fully from scratch. This feature is reflected in the metric of 'training complexity', which we define in section IV and which we use to evaluate the efficacy of our pruning technique.

Our method was motivated by a key observation that, for a randomly initialized network (of sufficient initial size), the total density of non-zero activations in the network (note, total density calculated across all layers) decreases during the training process with increasing epochs (see $net0$ graph in Fig. 1). This reduction in activation density indicates that the initial network $net0$ is overparameterized and that there is

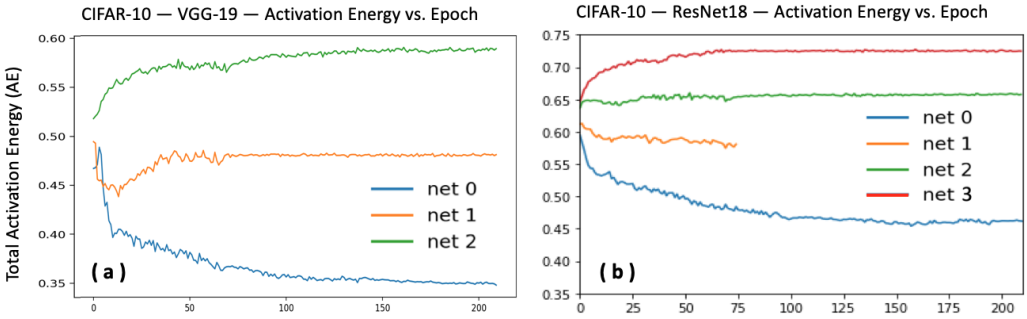

Fig. 1. Activation energy per epoch for successively pruned networks. In (a), $net0$ is VGG-19. In (b), $net0$ is ResNet18. In both (a) and (b), $net1$ refers to the pruned version of $net0$, and so on. All networks are trained on CIFAR-10. Total AE is calculated by summing the layer-wise AEs across all layers.

room for the reduction of layer sizes. We also find that the activation density trend of every layer in the network varies from one another (see Fig. 2). We interpreted the value of the density for each layer as characteristic of its representational inefficiency. Our interpretation implies, for example, that a convolution layer that only activates 44% of its neurons during a given training period is wasting 56% of its allotted capacity. In this case, our pruning method decides that the given layer only needs to be 44% of its initial size for the next training round.

II. Related Work

Much work has been done in the past few years with regards to network compression. Some of the earlier approaches focus on the idea of compressing a pre-trained network according to some salience criteria. Denton et al. [10] apply singular value decomposition to a pre-trained convolutional network. Han et al. [6] identify weights that are below a certain threshold and replace them with zeros to produce a sparse network, which is then fine-tuned for a few iterations to produce the final pruned network. Han et al. [7] also introduced Deep Compression, a technique that combines pruning methods with quantization and Huffman coding to achieve substantial improvements in energy efficiency. Other methods prune on the scale of channels or layers. Wen et al. [14] developed Structured Sparsity Learning (SSL), which regularizes the *architecture* of a pre-trained DNN to achieve speedups in inference. Zhou et al. [9] enforce channel-level sparsity during the training process.

More recently, the need for pre-trained networks has been questioned. Liu et al. [3] advocate a rethinking of structured pruning techniques, demonstrating that transferring weights from pre-trained networks to pruned networks is not as beneficial as is traditionally thought. Concurrently, Frankle & Carbin [8] introduced the "lottery ticket hypothesis," which theorizes the existence of small sub-networks ('winning tickets') that train faster and to the same degree of accuracy as the larger networks in which they were found. They propose Iterative Magnitude Pruning that finds winning tickets that are $90-95\%$ less dense (in terms of parameters) than their original counterparts. **The main difference between our technique and previous works is that the activation density driven pruning approach can be applied real-time during training and completely get rids of the need for pre-trained models to perform compression. This is a first-of-its-kind analysis that allows for structured layer-wise network compression, reducing training complexity while still producing high-performing networks.**

III. Pruning in Training Methodology

The key feature of our method is what we have called *"activation energy (AE) analysis"*, a moniker derived from the correlation between non-zero activations in the network and the number of multiply-accumulate (MAC) operations that the network performs. Periodically throughout the training process, we count the number of activations that are non-zero (equivalent to counting positive activations, since the negatives are zeroed out by the ReLU activation [15]) and divide by the number of total activations, yielding an *activation density or energy*. AE of every layer serves as a good metric to decide its compressed size. Essentially, we monitor the AEs of the layers during the training process and prune the layers based on the density at regular training intervals. For the networks that we tested on, it was sufficient to multiply the activation densities by the layer size at a current training epoch to obtain the layer sizes of the pruned network for the next training epoch. Note, activation energy and activation density are used interchangeably in the paper.

Algorithm 1 outlines our proposed approach. The key steps of our pruning in training method are: 1) Define an initial network $net_{initial}$ or $net[0]$ and train it until a pruning criteria ρ is reached; 2) Perform AE analysis on the network and obtain the density per layer; 3) For each layer, determine the new layer size by multiplying the $net[0]$ layer size by the AE of that layer; 4) Define a network $net[1]$ using the newly generated layer sizes. This network will be functionally identical to $net[0]$, just smaller; 5) Repeat steps 1-4 until a stopping criteria δ is reached.

We found that it did not make a difference to training convergence or final accuracy whether the pruned network (say, $net[1]$) was initialized with random weights or with learnt weights from the larger network (say, $net[0]$). For simplicity, we chose to randomly initialize the network in each pruning round. Note, the activation density or AE only indicates the total number of filters or weights to remove at each pruning step. There is no notion of significant weights in this analysis

Algorithm 1: Activation Density driven Pruning in Training

Input: Training dataset and randomly initialized network $net_{initial}$
Output: Trained and pruned network net_{final}
$net[0] = net_{initial}$
//Note, net[0] can be a large network like {VGG-19, ResNet18};
$epoch = 0$;
$index = 0$;
while *not* *stopping (δ) criteria* **do**
  net = Randomly Initialized ($net[index]$);
  while *not* *pruning (ρ) criteria* **do**
    train(net, $epoch$);
    for *L in net.Layers* **do**
      $\#nonzero[L] =$ count_nonzero_activations(L);
      AE[L] = $\frac{\#nonzero[L]}{\#total[L]}$;
    end
    $epoch++$;
    //Note, we train the network $net[index]$ while monitoring the layer-wise AE till ρ is satisfied.
  end
  $index++$;
  for *L in net.Layers* **do**
    $net[index].LayerSize[L] =$ AE[L] $\times net[index-1].LayerSize[L]$;
  end
  //Note, we prune the network $net[index-1]$ to get the compressed network $net[index]$ based on AE per layer. The pruning continues till δ is satisfied.
end
$net_{final} = net[index]$;

that tells us which specific filters to keep or prune. Thus, we randomly remove the filters at every layer based on the density. The fact that our approach is independent of pre-initialization and significant weights implies that network architecture is key for compression, supporting the results of Liu et al. [3].

A. Pruning (ρ) Criteria and Stopping (δ) Criteria

Algorithm 1 describes two different criteria: ρ indicates when to stop the training of the initial network as well as each successively pruned network ($net[index]$); and δ indicates when to stop the pruning process altogether. Both of these criteria are most easily understood from a visual inspection of the total AE for each network as the training progresses. Fig. 1 shows a typical graph of AE vs. epoch. In Fig. 1 (a), $net0$ is a VGG-19 model trained on CIFAR-10. $net1$ and $net2$ are successively pruned models.

AE for $net0$ decays throughout the entire training process in Fig. 1 (a). However, after a certain point (approximately 100 epochs) it flattens out. Interestingly, we observed a correspondence between this saturation of AE and a saturation of the network accuracy. Both the accuracy and the AE for a given training period can be characterized by two regimes: 1) Before the saturation point, both quantities exhibit noticeable long-term trends as well as short-term fluctuations; 2) After the saturation point, both quantities stabilize and their derivatives seem to approach zero. Based on this observation, we choose the pruning criteria ρ to be equivalent to this saturation point (100 epochs in $net0$, for example). Training any further beyond ρ results in minimal change in AE/accuracy and unnecessarily increases the training time. For $net1$ and $net2$ in Fig. 1 (a), the saturation point ρ occurs at approximately 70 epochs. Note, in Fig. 1, we trained each network well past their saturation points for the purpose of demonstrating network behavior in the post-saturation regime. In practice, we stop training each network at their respective ρ values.

The second stopping criteria δ is determined on the basis of the overall shape of AE vs. epoch curve for each network. $net0$ can be characterized as convex, $net1$ as flat, and $net2$ as concave for the VGG-19 CIFAR-10 graphs in Fig. 1 (a). We interpreted the convexity of $net0$ as an indicator of overparameterization and that, since AE went down as accuracy went up, the network learned to ignore redundant connections. This means that we can remove those redundancies without significantly damaging the network's performance. $net2$, in contrast, trended upwards; by our interpretation of the AE metric, this indicates that the network learned to utilize more connections in order to improve its accuracy. Removing connections further will drastically reduce the accuracy of the network. Thus, we decide to stop the overall pruning process when we see an upward-trending or concave AE profile.

In practice, the ρ criteria is determined by monitoring the total AE during training. If AE does not change a lot ($\Delta AE < 0.001$) between two or more consecutive epochs, we label that as the saturation point and prune the layers based on the layer-wise densities obtained at the end of that particular epoch. The δ criteria is determined by plotting the AE profile once ρ is satisfied, and labeling it as either concave, flat, or convex.

B. Layer-wise sensitivity to pruning

While the total AE (in Fig. 1) of the network provides a convenient and interpretable stopping criteria (i.e. understood intuitively), the pruning process itself relies only on the layer-wise AE profile. For VGG-19 (see Fig. 2), we noticed that the layer-wise AE profiles varied greatly: *layers 1-8* exhibited the same concavity seen in the total AE profile, though the scale of the AE profiles tended to increase towards deeper layers. That is, the AE value of deeper layers, say *layers 7 & 8*, at the saturation point ρ tended to be lower than the AE value of shallower layers, say *layers 1 & 2*. *Layers 9-16* exhibited varying degrees of convexity, in opposition to the total AE trend of the network.

For the first half of the VGG-19 network, AE decreases with layer depth. The trend reverses after the network's midpoint (after *layer 8*): AE starts to increase as the network gets deeper. This observation aligned with the discussion of layer-wise pruning in [5]. The authors in [5] found using principal component analysis (PCA) that the number of significant dimensions

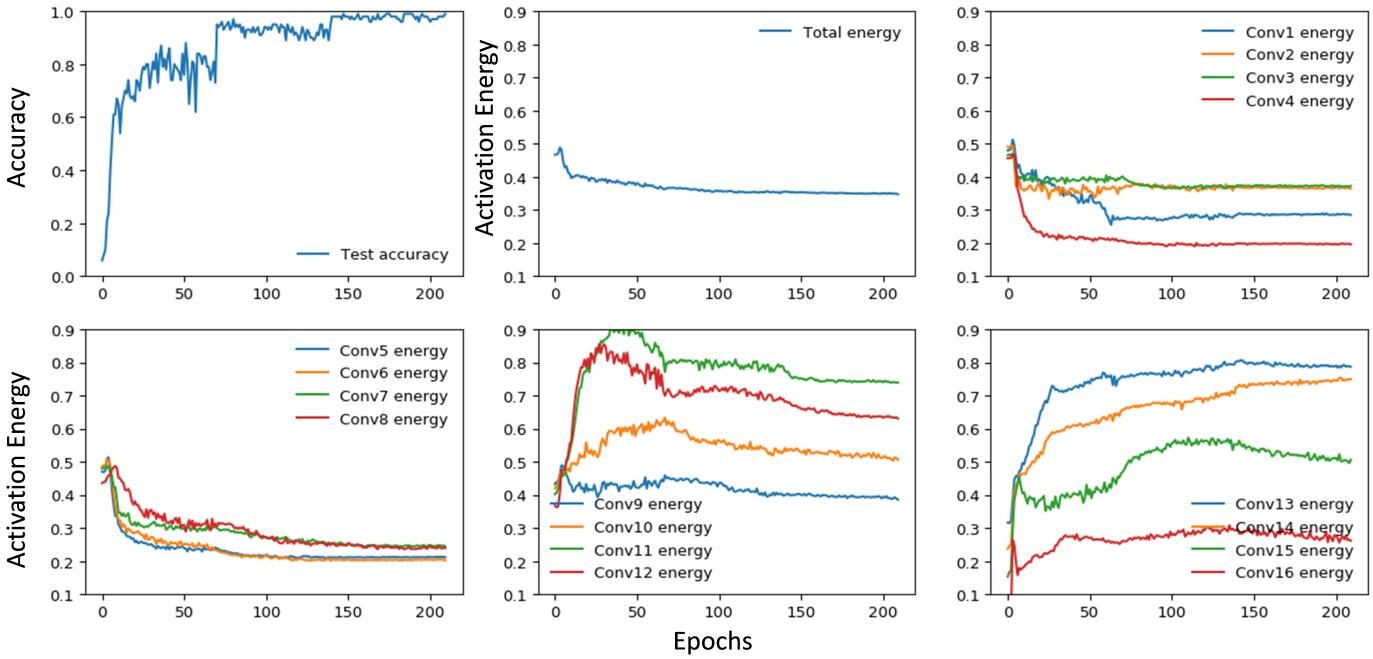

Fig. 2. Activation density per layer as training proceeds for VGG-19 on CIFAR-10. We did not include pooling layers or the final fully-connected layer, as there was no additional information present in those layers. The trend of total decreasing activation energy per layer can be seen here. Additionally, we observe a convex energy profile for the first eight layers and a concave profile for the second eight layers.

TABLE I

SUMMARY OF RESULTS.THE FINAL PRUNED MODEL OBTAINED FROM OUR METHOD FOR EACH SCENARIO HAS BEEN **HIGHLIGHTED**.

	Configuration	Accuracy	Parameters reduction	MACs reduction	Training Epochs[a] ρ
	CIFAR-10, ResNet18				
net 0	[64, 64, 64, 64, 64, 128, 128, 128, 128, 256, 256, 256, 256, 512, 512, 512, 512]	97 %	1x	1x	100 epochs
net 1	[34, 29, 41, 25, 33, 58, 78, 27, 65, 71, 83, 46, 69, 120, 191, 219, 288]	97 %	7.3x	6.0x	70 epochs
net 2	**[21, 16, 30, 10, 22, 24, 47, 9, 39, 26, 48, 12, 39, 41, 85, 63, 188]**	**95 %**	**41.2x**	**23.2x**	**70 epochs**
net 3	[14, 9, 21, 5, 15, 13, 32, 5, 26, 13, 34, 5, 25, 21, 45, 12, 142]	91 %	199.3x	67.1x	N/A
	CIFAR-10, VGG-19				
net 0	[64, 64, 128, 128, 256, 256, 256, 256, 512, 512, 512, 512, 512, 512, 512, 512]	97 %	1x	1x	100 epochs
net 1	**[18, 23, 47, 25, 54, 51, 62, 61, 197, 258, 378, 322, 402, 383, 259, 134]**	**94 %**	**3.1x**	**5.6x**	**70 epochs**
net 2	[10, 9, 30, 11, 21, 31, 22, 21, 62, 70, 113, 141, 256, 299, 194, 71]	93 %	10.3x	27.4x	N/A
	CIFAR-100, ResNet18				
net 0	[64, 64, 64, 64, 64, 128, 128, 128, 128, 256, 256, 256, 256, 512, 512, 512, 512]	81.0 %	1x	1x	25 epochs
net 1	**[39, 31, 49, 24, 44, 54, 90, 36, 84, 88, 155, 65, 136, 130, 231, 105, 300]**	**79.0 %**	**7.6x**	**5.1x**	**N/A**
	CIFAR-100, VGG-19				
net 0	[64, 64, 128, 128, 256, 256, 256, 256, 512, 512, 512, 512, 512, 512, 512, 512]	76.0 %	1x	1x	25 epochs
net 1	**[34, 23, 51, 30, 63, 63, 73, 82, 210, 285, 333, 357, 317, 259, 181, 106]**	**73.0 %**	**3.9x**	**5.3x**	**N/A**
	TinyImageNet, ResNet18				
net 0	[64, 64, 64, 64, 64, 128, 128, 128, 128, 256, 256, 256, 256, 512, 512, 512, 512]	51.54 %	1x	1x	25 epochs
net 1	**[31, 21, 47, 27, 48, 62, 99, 58, 94, 85, 161, 69, 133, 93, 152, 56, 247]**	**50.51 %**	**10.6x**	**4.7x**	**N/A**

[a]Training time until saturation point. The reported accuracies are from a full training cycle (210 epochs), except for TinyImageNet, which was trained for 60 epochs.

contributing to the variance of the activations decreased past the mid-point of a VGG network. They removed the deeper layers based on this observation, and found a negligible degradation of accuracy. Along similar lines, we also interpreted the reversed AE trend after $layer8$ of VGG-19 to mean that the last layers were no longer identifying increasingly abstract features from the input data and that removing them would have little effect on the network's performance. However, we found that removing whole layers severely degraded the accuracy of the pruned networks. For VGG-19 on CIFAR-10, the accuracy of our first pruned network (with just the first 8 layers intact and pruned based on AE) never exceeded 70%, even after a full training cycle (210 epochs with learning rate decay). This implies that depth is significant to achieving good training convergence. Furthermore, we found that the overall reduction in inference compute OPS we achieved with AE-based pruning on a VGG-19 network is higher than that of [5] at iso-accuracy (see Section IV.D), even without removing the latter layers.

Analysing the layer-wise AE profile (not shown here) of a ResNet18 model trained on CIFAR-10 shows a uniform concave trend across all layers similar to the total AE trend (as

in $net0$ of Fig. 1 (b)) with no reversal at the network's midpoint. To perform network agnostic pruning without limiting the training convergence, we, therefore, use AE as an indicator of compression width per layer and not of the overall depth.

IV. Results

We evaluate our pruning in training strategy on two commonly used networks: VGG-19 and ResNet18 for CIFAR-10, CIFAR-100 [16], and TinyImageNet datasets [17]. We imported github models from [18] for implementing our experiments in PyTorch. We used similar hyperparameters as [8] and [19] to train our models on CIFAR-10/100 and Tiny ImageNet, respectively.

A. Energy analysis

Our results are summarized in Table I. For energy calculation or net inference OPS, we specify each MAC operation at the register transfer logic (RTL) level for 45nm CMOS technology [6]. Considering 32-bit floating point operations, the energy per MAC (E_{MAC}) is estimated as $4.6pJ$. Total MAC energy across all N layers of a network net can be specified as $E_{net} = (\sum_{i=1}^{N} \text{\# MAC}_i) * E_{MAC}$. For a particular convolutional layer of a network with N input channels, M output channels, input map size I , weight kernel size k and output size O, total MAC count is # MAC $= O^2 * N * k^2 * M$. Note, this energy calculation is a rather rough estimate which does not take into account memory access energy and other hardware architectural aspects such as input-sharing, weight-sharing or zero-checker logic.

In Table I, *'MACs reduction'* is defined as $\frac{E_{prunednetwork}}{E_{baseline}}$. $net0$ in each case serves as the baseline. We also specify the *'parameters reduction'* computed with respect to the baseline. The best performance of our pruning algorithm was obtained for ResNet18 on CIFAR-10, where we achieved a 200x reduction in parameters with 6.2% reduction in accuracy compared to baseline. We observe a natural tradeoff between accuracy and MACs reduction. For all scenarios in Table I, the final pruned model chosen is $net1$ (except CIFAR-10 ResNet18 case for which we choose $net2$). Pruning beyond that (say $net2, net3$ for CIFAR-10) yields a concave AE profile (see Fig. 1) which activates the stopping criterion δ.

Across all datasets, our pruning algorithm produced more accurate ResNet-type models (with lower accuracy loss compared to baseline) than it did for VGG-type models. The compression in terms of MACs reduction achieved with VGG is slightly higher than ResNet (at equivalent pruning levels). Although not reported in Table I, we noted that the total AE for ResNet18 remained at a higher point throughout the entire pruning process than that of VGG-19, which can be seen in Fig. 1. For VGG-19 $net0$, the total AE value started around 0.5 and decreased to approximately 0.3 over the course of training. For the corresponding ResNet18 network $net0$ in Fig. 1, the AE started 10% higher. We attribute this increased density to the shortcut connections in the residual network architecture. Throughout the training process, the densities of the ResNet layers maintained a 10% higher AE over their VGG counterparts. Since the pruned network's size is determined from AE (see Algorithm 1), higher value of AE implies lower network compression which justifies our ResNet vs. VGG results.

TABLE II
Training complexity for our pruning method

Network	ResNet18			VGG-19	
	CIFAR-10	CIFAR-100	Tiny ImNet	CIFAR-10	CIFAR-100
$net0$	210.0 (1x)	210.0 (1x)	60.0 (1x)	210.0 (1x)	210.0 (1x)
$net1$	135.0 (0.64x)	66.2 (0.32x)	37.7 (0.62x)	120.2 (0.57x)	64.6 (0.31x)
$net2$	120.8 (0.58x)	-	-	-	-

B. Training complexity

Our method trains progressively smaller networks (networks pruned at each saturation point ρ) which reduces the overall training complexity, a remarkable advantage of pruning in training. We define 'training complexity' as:

$$\sum_{net_i} (\text{MACs reduction}_{net_i})^{-1} \times (\text{\# training epochs}_{net_i}) \quad (1)$$

where net_i is the set of successively pruned networks (i.e. $\{net0, net1, ...\}$) for a given starting configuration. In Table I, we specify the training epochs for each network (# training epochs$_{net_i}$) until the saturation or pruning criteria ρ is satisfied. As an example, training complexity for CIFAR-100 is calculated as $25 * (\text{MACs reduction})^{-1}_{net0} + 210 * (\text{MACs reduction})^{-1}_{net1}$.

It was our original goal to produce a pruning method for implementation in resource-limited environments; this motivated us to focus on the easily-computable heuristic of activation density and prune according to that. With the training complexity defined as above, we are essentially measuring the amount of time and training energy required to achieve a given model accuracy, compression ('parameters reduction'), and efficiency ('MACs reduction'). Table II shows the training complexity of selected networks from our pruning procedure. Note that the networks referred to by Table II across different datasets are same as that of Table I. For ResNet18 on CIFAR-10, we see that $net1$ is objectively better than $net0$, since it achieved the same accuracy as $net0$ but with fewer parameters, fewer MACs, and a lower training complexity. Note that, in our evaluation in Eqn. 1, the final pruned network as well as the baseline for CIFAR-10,100 (Tiny ImageNet) was trained for 210 (60) epochs, respectively.

C. Visualization

In addition to plotting the total AE per epoch for each successively pruned network, it proved helpful to visualize the increasing density using a colormap. The result of this visualization is shown in Fig. 3 for VGG-19 on CIFAR-10. For each network, we show the input as well as colormaps corresponding to certain layers in the network (the same layers are shown for $net0$ and $net2$). We generated the colormaps by taking the average of the output activations across all the filters in a given layer. Although certain layers break the pattern, we see an overall trend of higher AE (more color) in the layers of $net2$ than in the layers of $net0$, the baseline network.

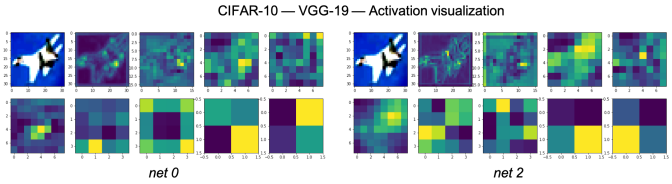

Fig. 3. Colormap visualization of the output activations of selected layers in each network. From left to right, top to bottom, the layers represented are: input, 1, 3, 5, 7, 9, 11, 13, 15, 17. For each network and layer, activations from all filters were averaged to produce the colormap shown here.

TABLE III
COMPARISON WITH PREVIOUS WORK FOR VGG-19 CIFAR-100

Authors	Training complexity	Accuracy	Parameters reduction	MACs reduction
Garg et al. [5]	206.6	71 %	9.1x	3.9x
Liu et al. [12]	260.0	73 %	8.7x	1.6x
Ours	64.6	73 %	3.9x	5.3x

D. Comparison with previous work

One of the notable differences of our proposed method against previously proposed pruning techniques is that it is independent of having pre-trained initialization. We optimize the network architecture real-time during training which in turn yields additional training complexity reduction. Table III compares our results with that of two recent works [5], [12] that aimed to reduce the total number of time intensive pruning-retraining iterations. However, both works still relied on fully or partially trained networks as a starting point.

For example, Liu et al. [12] train VGG-19 for 160 epochs, apply their pruning method, then train the pruned network (with a MACs reduction of 1.6x) for another 160 epochs. The initial training adds an unavoidable '160 epochs' of training complexity. Similarly, Garg et al. use PCA on a fully pre-trained network to find the optimal architecture in one single shot. Our method achieves a better training complexity score because we prune *in training*, allowing us to only train the original (1x MACs reduction) network for 25 epochs before pruning it. When we finally train all the way to a decent accuracy, the MACs in our pruned model have been sufficiently reduced that the final training period of 210 epochs does not incur a heavy training complexity penalty. A noteworthy observation here is that our approach yields the highest benefits in terms of MACs reduction which establishes the effectiveness of the AE driven pruning for structured layer-wise network compression. Although not reflected in Table III, we achieve comparable accuracy and compression to many of the works mentioned in the related work section with the additional benefit of lower training complexity.

V. CONCLUSION

We propose a novel pruning in training method that yields significant compression benefits on VGG, ResNet-like architectures. To conduct structured layer-wise pruning, we perform an 'Activation Energy' analysis, a simple yet powerful heuristic that provides a structured and visually interpretable way of optimizing the network architecture. Furthermore, the progressive downsizing of a network during the training process yields training benefits. We get considerable benefits in training complexity as well as MACs or compute OPS reduction over the baseline unpruned model as well as previously proposed pruning methods. This can be attributed to the pre-training and significant filter independence of our method and the structured AE driven compression. The improved training/inference complexity reduction makes our proposed algorithm most desirable for implementation on resource-limited systems, such as mobile devices.

Finally, we would like to consider other ramifications of our technique. In essence, our method penalizes networks for having zeros in their activations, forcing the pruned network to have a denser set of activations in comparison to the baseline. A possible negative consequence of enforcing higher density without zeros for ReLU-based networks is that they will not be able to learn non-linearities due to the linear profile of the ReLU function in the regime of positive-only inputs. In the limiting case where a network becomes 100% dense on a given dataset—where there is never an activation that is zero—the network will be over-fitted and have a greatly reduced ability to generalize on test data. However, the existence of an activation energy saturation point implies that it will be very difficult to create such a 100% dense network.

REFERENCES

[1] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," *arXiv:1409.1556*, 2014.
[2] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *ICCV*, 2016, pp. 770–778.
[3] Z. Liu, M. Sun, T. Zhou, G. Huang, and T. Darrell, "Rethinking the value of network pruning," *arXiv:1810.05270*, 2018.
[4] Y. LeCun, J. S. Denker, and S. A. Solla, "Optimal brain damage," in *NIPS*, 1990, pp. 598–605.
[5] I. Garg, P. Panda, and K. Roy, "A low effort approach to structured cnn design using pca," *arXiv:1812.06224*, 2018.
[6] S. Han, J. Pool, J. Tran, and W. Dally, "Learning both weights and connections for efficient neural network," in *NIPS*, 2015, pp. 1135–1143.
[7] S. Han, H. Mao, and W. J. Dally, "Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding," 2015.
[8] J. Frankle and M. Carbin, "The lottery ticket hypothesis: Finding sparse, trainable neural networks," *arXiv:1803.03635*, 2018.
[9] H. Zhou, J. M. Alvarez, and F. Porikli, "Less is more: Towards compact cnns," in *ECCV*. Springer, 2016, pp. 662–677.
[10] E. L. Denton, W. Zaremba, J. Bruna, Y. LeCun, and R. Fergus, "Exploiting linear structure within convolutional networks for efficient evaluation," in *NIPS*, 2014, pp. 1269–1277.
[11] F. N. Iandola, S. Han, M. W. Moskewicz, K. Ashraf, W. J. Dally, and K. Keutzer, "Squeezenet: Alexnet-level accuracy with 50x fewer parameters and¡ 0.5 mb model size," *arXiv:1602.07360*, 2016.
[12] Z. Liu, J. Li, Z. Shen, G. Huang, S. Yan, and C. Zhang, "Learning efficient convolutional networks through network slimming," in *IEEE ICCV*, 2017, pp. 2736–2744.
[13] J. Frankle, K. Dziugaite, A. Element, D. M. Roy, and M. Carbin, "Stabilizing the lottery ticket hypothesis," *arXiv:1903.01611v2*, 2019.
[14] W. Wen, C. Wu, Y. Wang, Y. Chen, and H. Li, "Learning structured sparsity in deep neural networks," 2016.
[15] V. Nair and G. E. Hinton, "Rectified linear units improve restricted boltzmann machines," in *ICML*, 2010, pp. 807–814.
[16] A. Krizhevsky, G. Hinton *et al.*, "Learning multiple layers of features from tiny images," Citeseer, Tech. Rep., 2009.
[17] [Online]. Available: https://tiny-imagenet.herokuapp.com/
[18] [Online]. Available: www.github.com/kuangliu/pytorch-cifar
[19] [Online]. Available: https://github.com/tjmoon0104/Tiny-ImageNet-Classifier