Structured Pruning for Multi-Task Deep Neural Networks

Siddhant Garg
Adobe Inc.
Email: siddhantg@adobe.com

Lijun Zhang
College of Information & Computer Sciences
University of Massachusetts Amherst
Amherst, Massachusetts, 01003
Email: lijunzhang@cs.umass.edu

Hui Guan
College of Information & Computer Sciences
University of Massachusetts Amherst
Amherst, Massachusetts, 01003
Email: huiguan@cs.umass.edu

***Abstract*—Although multi-task deep neural network (DNN) models have computation and storage benefits over individual single-task DNN models, they can be further optimized via model compression. Numerous structured pruning methods have already been developed that can readily achieve speedups in single-task models, but the pruning of multi-task networks has not yet been extensively studied. In this work, we investigate the effectiveness of structured pruning on multi-task models. We use an existing single-task filter pruning criterion and also introduce an MTL-based filter pruning criterion for estimating the filter importance scores. We prune the model using an iterative pruning strategy with both pruning methods. We show that, with careful hyper-parameter tuning, architectures obtained from different pruning methods do not have significant differences in their performances across tasks when the number of parameters is similar. We also show that iterative structure pruning may not be the best way to achieve a well-performing pruned model because, at extreme pruning levels, there is a high drop in performance across all tasks. But when the same models are randomly initialized and re-trained, they show better results.**

***Index Terms*—Multi-Task Learning, Structural Pruning**

I. Introduction

Multi-task learning (MTL) addresses multiple machine learning tasks simultaneously by creating a single multi-task deep neural network (DNN) [1]. Due to parameter sharing, a multi-task DNN model is more computation and memory efficient compared to multiple single-task models. The MTL framework is extremely useful for resource-constrained devices like smartphones, wearables, and self-driving cars that host AI-powered applications but have low memory resources and strict latency requirements. For example, in self-driving cars, the model needs to recognize traffic lights, objects, and lanes based on the input vision signal [2]. Multi-task models are also frequently used in other vision and content understanding tasks such as segmentation and object detection.

On the other hand, network pruning [3]–[5] is a long-standing and effective method to compress DNNs. It aims to detect the importance of model parameters and remove the ones that tend to have the least significance on the performance. Approaches for network pruning can be classified as unstructured pruning methods [3], [4] that mask individual weights in the network, and structured pruning methods [5], [6] that remove complete filters and directly lead to efficient deep neural models without requiring specialized hardware for sparse structures. Network pruning is studied rigorously in literature for single-task models and there exist many pruning criteria based on weight magnitude [4], [5], connection sensitivity [7], [8], and even learning-based pruning methods [9], [10].

However, the study on the effectiveness of structured pruning methods on multi-task models is sparse. A few works like [11], [12] propose weight sharing and merging strategies for constructing a multi-task model from multiple single-task models such that there is a minimum conflict between tasks. Then the constructed multi-task model could be effectively pruned with single-task pruning methods. But these works show very similar results when comparing their proposed pipeline with the baseline single-task pruning methods. Another work [13], proposed a method to directly prune MTL networks but while pruning those models with single-task pruning baselines, they did not try different hyperparameter settings to retrain/fine-tune the pruned models. It could be possible that different hyperparameter settings would work better because different pruning methods lead to different architectures.

In this work, we investigate the effectiveness of structured pruning on multi-task DNNs. We apply two structured pruning methods to prune multi-task models and show that regardless of the method used, we can obtain similar results from the pruned models with the same number of parameters. Specifically, we use an existing single-task pruning method as well as introduce another MTL-based pruning criterion. The proposed criterion is called CosPrune and it identifies and prunes the convolutional filters that have conflicts between tasks. It uses pairwise cosine similarity between the task-specific gradients that flow through the filter during back-propagation. We accumulate this similarity score for some training iterations for every filter in the multi-task model. The filters with the least accumulated scores are pruned away. In contrast, the single-task pruning method is called Taylor Pruning [6] which is a popular gradient-based pruning method. It determines the importance of the filter by looking at the increase in the loss function if that filter is removed. The Taylor pruning importance score is also accumulated over some training iterations before pruning.

We start our analyses by using the iterative pruning and

fine-tuning strategy which repeatedly prunes a small proportion of filters and fine-tunes the multi-task model to gain back the lost performance [4], [14]. Using this strategy we get consistently better results with CosPrune against Taylor pruning across all the tasks. The multi-task model achieved higher GFLOPs/parameter reduction with CosPrune without performance loss across all the tasks. This shows that the proposed CosPrune criterion coupled with iterative pruning is a reasonable method for pruning multi-task models.

However, when we re-train the pruned models independently with random initialization, we observe that they can give relatively better results on all the tasks when compared to the corresponding fine-tuning stage of iterative pruning. The key is to determine good learning rates for each of the pruned models. Using the same learning rates is not the best strategy to compare different pruned architectures. This is because every model has different layer-wise configurations and so, an optimal hyper-parameter setting for one model may not be best for the other models with different architectural designs. This type of analysis is generally not done in the existing literature – the same training settings are used for the dense model as well as the pruned models. A study on single-task structured pruning [14] also shows that re-training the pruned models from random initialization can lead to better results than fine-tuning the pruned architectures in the iterative pruning setting.

Furthermore, the pruned architectures from different pruning methods give similar results to each other at the same parameter level after re-training. **There are no consistent winners with respect to different pruning criteria, which is contrary to what we observe in the case of iterative pruning.** There are also some recent works [15], [16] in the context of single-task neural networks where random channel pruning is able to match the results of the dense model under appropriate settings. Similarly, another work [14] used different pruning methods on various architectures to show that randomly re-initializing the pruned models can match the performance of the respective unpruned models but they did not compare those different pruning strategies on the same model.

We go beyond existing works and apply different structured pruning methods to the same multi-task model. **We emphasize that the pruned MTL model obtained from any reasonable pruning method can perform well if it is trained from random initialization with its optimal learning rate.** And we would like researchers to pay attention to the re-training comparisons with sufficient hyper-parameter tuning when they try to propose a new pruning method.

II. Related Works

Multi-Task Learning: Deep MTL networks afford numerous benefits in terms of computation (storage and latency) [17], knowledge sharing between tasks [18] and improving generalization in the learned representations [19] due to which they have applications across many domains like Computer Vision [20]–[23], Robotics [24], [25], and Reinforcement Learning [26]–[29]. The most common MTL framework is hard parameter sharing [1], [30] where a backbone network is shared among all tasks and with individual task-specific heads. Soft parameter sharing, on the other hand, has different sets of parameters for individual tasks [19], [31] and various methods are used to effectively combine information from them [32]–[34] to make predictions.

Deep Neural Network Pruning: It has long been established that the deep neural networks are highly over-parameterized [3], [5], [35]–[37], and more than 90% of the connections can be pruned away without losing accuracy. Unstructured pruning methods for single-task models [4], [38] can achieve high sparsity with latency improvements with accelerated hardware for sparse neural networks [39]. On the other hand, structured pruning can directly lead to computational benefits by removing whole filters in the case of convolutional layers [5]. Gradient-based metrics for estimating the importance scores of the filters have recently become popular over magnitude-based criteria [5]. For example, Taylor Pruning [6] uses the dot product between the filter weights with its gradient to approximate the change in loss function that could occur if that filter is masked. Another example is SNIP [7], which determines the connection sensitivity using a similar importance metric as Taylor Pruning but it is done only once at initialization to apply single-shot pruning. Furthermore, random channel pruning methods [15], [16] are also shown to perform well under appropriate training conditions.

Pruning Multi-task Neural Networks: There is very limited study on pruning MTL neural networks but nonetheless it is an important problem because of its tremendous potential in deriving efficient deep learning models. At the same time, it is also a difficult problem because of the coexistence of complex task relationships in the shared parameter space and redundancy in the neural network. A recent work on MTL pruning is DiSparse [13] which aims to disentangle task relationships to find the unanimously least important filters across all the tasks and prune them. Another work called PAM [12] propose a method to merge single-task models into an MTL network such that the resulting MTL network can be safely pruned while considering the tasks' relatedness. Similarly, other works like [40], [41] propose different merging strategies for the construction of computationally efficient MTL networks. All of these existing works do baseline comparisons by using single-task pruning methods to prune their MTL networks. However, they use only the hyperparameters that are specified in the original study on single-task pruning methods irrespective of the different model architectures and sparsity ratios. But in our work, we do extensive analyses and hyperparameter tuning for both the pruning methods involved to show their true effectiveness in pruning deep MTL networks.

III. Pruning Methods

In this section, we will define our Multi-Task Learning model and the proposed CosPrune importance score estimation method as well as review the Taylor Pruning importance score.

Multi-task framework: Given a set of T tasks $\mathcal{T} = \{\tau_1, \ldots, \tau_T\}$, an MTL dataset with inputs, $\mathcal{X}$, and the corresponding labels $\mathcal{Y} = \{Y_1, \ldots, Y_T\}$ where Y_t is set of labels for task $\tau_t \in \mathcal{T}$, we want to learn the mapping $f_\Theta : \mathcal{X} \rightarrow \mathcal{Y}$, where $\Theta = \theta_s \cup \{\theta_t\}_{t=1}^T$. Here θ_s is the set of model parameters that are shared by all the tasks and θ_t is the set of parameters only for task τ_t. From now, we will denote $\{\theta_t\}$ as the set of all the task-specific parameters for all the tasks collectively and omit values of t for abbreviation. The parameters Θ are trained by minimizing the multi-task loss function given by

$$\mathcal{L}(\mathcal{X}, \mathcal{Y}, \Theta) = \sum_{t=1}^{T} \ell_t \left(\mathcal{X}, \mathcal{Y}, \theta_s, \theta_t\right) \tag{1}$$

where $\ell_t \left(\mathcal{X}, \mathcal{Y}, \theta_s, \theta_t\right)$ is the loss function for the task τ_t.

CosPrune Importance Score Estimation: Let $\mathbf{W}^{k \times k \times c} \in \Theta$ be a convolutional filter with kernel size k, and c channels, then the weights of this filter will be optimized using the gradient descent given by equation 2,

$$\mathbf{W} \leftarrow \mathbf{W} - \eta \nabla_{\mathbf{W}} \mathcal{L}(\mathcal{X}, \mathcal{Y}, \Theta) \tag{2}$$

where η is the learning rate and $\nabla_{\mathbf{W}} \mathcal{L}(\mathcal{X}, \mathcal{Y}, \Theta)$ is the gradient of the total loss function with respect to $\mathbf{W}$. Let $\mathbf{g}_{\mathcal{T}}^{\mathbf{W}} = \nabla_{\mathbf{W}} \mathcal{L}(\mathcal{X}, \mathcal{Y}, \Theta)$, then

$$\mathbf{g}_{\mathcal{T}}^{\mathbf{W}} = \sum_{t=1}^{T} \mathbf{g}_t^{\mathbf{W}} \tag{3}$$

where $\mathbf{g}_t^{\mathbf{W}} = \nabla_{\mathbf{W}} \ell_t \left(\mathcal{X}, \mathcal{Y}, \theta_s, \theta_t\right)$. $\mathbf{g}_{\mathcal{T}}^{\mathbf{W}}$ and $\mathbf{g}_t^{\mathbf{W}}$ can be denoted as the total gradient and task-τ_t gradient respectively with respect to the parameter $\mathbf{W}$. Note that the total gradient is the vector sum of all the individual task gradients. These individual gradients can have high differences in magnitudes as well as they can point to different directions with negative cosine similarity between them. This can lead to conflicts between different tasks and that could be detrimental to the optimization process [42]. There have been several works like PCGrad [42], MGDA [43], CAGrad [44] that apply various methods to remove the conflicts in the optimization process. In our work, we prune the filters that have the highest degree of accumulated conflicts but optimize our MTL model with the total gradient. This helps us to achieve the highest degree of optimization in terms of computation and performance.

To calculate the degree of conflict, we define the CosPrune Importance score, as the sum of pairwise cosine similarity between all the tasks for the parameter $\mathbf{W}$. Let $\tilde{\mathbf{g}}_t^{\mathbf{W}} = \frac{\mathbf{g}_t^{\mathbf{W}}}{\|\mathbf{g}_t^{\mathbf{W}}\|}$ be the normalized task gradient, and $\{\mathbf{g}_t^{\mathbf{W}}\}$ be the set of all the task gradients with respect to $\mathbf{W}$, then $\mathcal{C}\left(\mathbf{W}, \{\mathbf{g}_t^{\mathbf{W}}\}\right)$ can be calculated using:

$$\mathcal{C}\left(\mathbf{W}, \{\mathbf{g}_t^{\mathbf{W}}\}\right) = \sum_{(\tau_i, \tau_j) \in \mathcal{T} \times \mathcal{T}} \tilde{\mathbf{g}}_i^{\mathbf{W}} . \tilde{\mathbf{g}}_j^{\mathbf{W}} \tag{4}$$

We only prune the shared parameters, $\mathbf{W} \in \theta_s$, therefore $\mathbf{g}_t^{\mathbf{W}} \neq \mathbf{0}$ for all $t = 1, \ldots, T$. The value of $\mathcal{C}(.,.)$ will be higher when the pairwise gradients are in agreement and it will lower if tasks are in conflict. For calculating the final importance score of a filter, we keep on adding the CosPrune scores for all the filters for a fixed number of training iterations. After that, we rank the filters according to their accumulated importance scores and prune the lowest-scoring filters. After pruning, we reset the accumulated scores to zero and fine tune the model again along with collecting the CosPrune scores of the remaining unpruned filters. The details are provided in Algorithm 1.

Algorithm 1 MTLCosPrune

Input: Dataset: $\mathcal{X}$, $\mathcal{Y}$ with T tasks; Model Parameters: $\mathbf{W} \in \Theta$; Fine-tune Iterations – I; #Filters to prune at each pruning step – P,
Initialize Θ,
while #Unpruned Filters $> threshold$ **do**
  Set $\mathcal{AC}(\mathbf{W}) \leftarrow 0 \ \forall \mathbf{W} \in \Theta$ (accumulated scores)
  for $i = 1, 2, \ldots, I$ **do**
    $\mathbf{x}_i \leftarrow$Batch Inputs
    $\mathbf{y}_i \leftarrow$Batch Labels
    $\mathbf{g}_t^{\mathbf{W}} = \nabla_{\mathbf{W}} \ell_t \left(\mathbf{x}_i, \mathbf{y}_i, \theta_s^i, \theta_t^i\right) \ \forall t, \forall \mathbf{W} \in \theta_s^i$
    Get $\mathcal{C}(\mathbf{W}, \{\mathbf{g}_t^{\mathbf{W}}\})$ using Eq. 4 $\forall t, \forall \mathbf{W} \in \theta_s^i$
    Update $\mathcal{AC}(\mathbf{W}) \leftarrow \mathcal{AC}(\mathbf{W}) + \mathcal{C}(\mathbf{W}, \{\mathbf{g}_t^{\mathbf{W}}\}) \ \forall t, \forall \mathbf{W} \in \theta_s^i$
    Compute $\mathbf{g}_{\mathcal{T}}^{\mathbf{W}} = \sum_{t=1}^{T} \mathbf{g}_t^{\mathbf{W}}$
    Update $\mathbf{W} \leftarrow \mathbf{W} - \eta \mathbf{g}_{\mathcal{T}}^{\mathbf{W}}, \ \forall t, \forall \mathbf{W} \in \theta_s^i$
  end for
  Prune P filters $\mathbf{W}$ with lowest $\mathcal{AC}(\mathbf{W})$ scores
end while

Taylor Pruning [6]: It is a structured pruning method for single-task convolution neural networks which serves as a popular baseline for modern gradient-based pruning methods. In this method, the importance score of a filter is defined by the squared change in the loss function induced by removing the filter. To make the computation efficient, the squared change is approximated by the first-order Taylor expansion which simplifies to a dot product between the filter weights and its gradient. Let the importance score for the filter $\mathbf{W}$ be $\mathbf{I}$, then it is given by

$$\mathbf{I} = \mathbf{W} . \mathbf{g}_{\mathcal{T}}^{W} \tag{5}$$

where $\mathbf{g}_{\mathcal{T}}^{\mathbf{W}}$ is the total gradient passing through filter $\mathbf{W}$ during backpropagation as define in equation 3.

We experiment with both CosPrune and Taylor pruning methods and compare their performance in effectively pruning the MTL models.

IV. Experiments and Results

In this section, we provide the details of all the experiments and provide quantitative evidence of our claims.

A. Setup

Model Architecture: For our multi-task model, we use a hard parameter-sharing paradigm where the backbone parameters are shared across all the tasks and individual classification heads are used for each task. MTL backbone comprises of

VGG-16 [45] model without the last fully-connected layers and MTL classification heads use Artrous Spatial Pyramid Pooling (ASPP) heads [46] for each task. The backbone contains approximately $71M$ parameters and each ASPP head has approximately $13.4M$ parameters. In all of our experiments, we only prune the backbone but all the results are reported with the total model parameters (backbone + all task heads) which are approximately $84.12M$ parameters.

Multi-Task Learning (MTL) Dataset: In this work, NYUv2 [47] dataset is used for all of the experiments. It is a popular Multi-Task Learning MTL dataset with densely labeled images recorded from RGB and Depth cameras of Microsoft Kinect. It contains three tasks – Semantic Segmentation, Depth Estimation, and Surface Normal Prediction whose ground truth labels are defined in [20].

Evaluation Metrics: Semantic segmentation is evaluated using mean Intersection over Union (mIoU) and the Pixel accuracy (both the higher the better). Depth Estimation is evaluated using absolute and relative errors calculated using the L1 loss between the ground truth and predictions (both the lower the better). For this task, we also report the relative difference between the prediction and ground truth via the percentage of $\delta = \max\left(\frac{y_{pred}}{y_{gt}}, \frac{y_{gt}}{y_{pred}}\right)$ within threshold $1.25, 1.25^2$, and 1.25^3 [48] (the higher the better). Surface Normal Prediction is evaluated using the Mean and Median Angle errors calculated by cosine similarity loss (the lower the better). We also report the percentage of pixels whose predictions are within $11.25°$, $22.5°$, and $30°$ to the ground truth (higher the better) [20]. Due to space constraints and a high number of pruned model configurations, we report Pixel Accuracy for Semantic Segmentation, relative error for Depth Estimation, and Angle Mean for Surface Normal Prediction as the main evaluation metrics, for both pruning methods.

Loss Functions and Model Training: For training on the NYUv2 dataset, the model minimizes the sum of losses from the individual tasks with equal weightage. Semantic segmentation uses cross-entropy loss, Surface Normal Prediction uses cosine similarity loss and Depth Estimation uses L1 loss as its training signals. We start with a randomly initialized MTL model with VGG-16 backbone and ASPP heads and train the model for 500 epochs. We use a batch size of 16, and 0.0001 learning rate with cosine scheduling [49]. We use AdamW [50] optimizer for the iterative pruning experiments.

Model Pruning: After the model is trained on the NYUv2 dataset, we apply iterative pruning with different filter pruning criteria (CosPrune and Taylor pruning) to get pruned MTL models. We initialize the model with the NYUv2 trained weights and set the initial learning rate as 10^{-5} with cosine scheduling for 500 epochs and start the pruning process. For every iteration (batch of inputs to the model), we get the individual task gradients by applying backward propagation on task losses. Then we calculate the importance score for every filter in the model. The importance scores are accumulated for 10 epochs after which 100 filters that have the lowest accumulated scores, are pruned. This completes a single pruning iteration and after it, the accumulated filter importance scores are reset to 0. The learning rate and scheduler are also rewinded to the initial settings and the pruning process continues. Note that the parameters of the model keep on updating using the total loss function after every iteration which marks the fine-tuning phase of the model.

B. Results: Multi-Task Structure Pruning

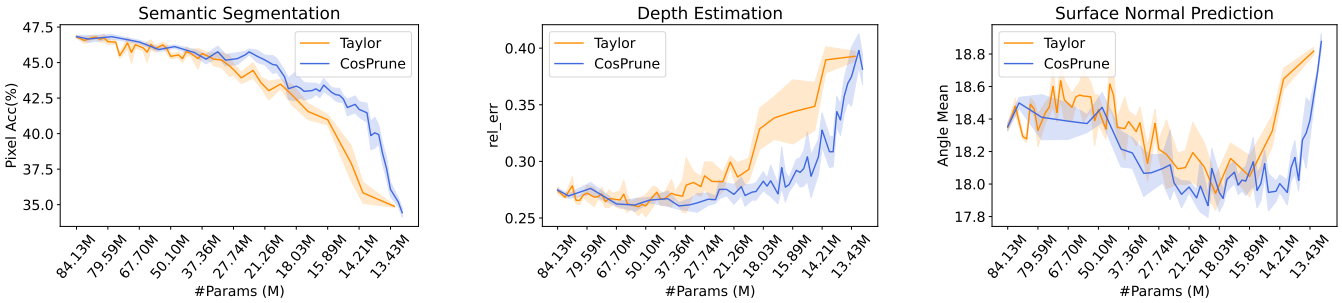
Iterative Pruning Results: We applied the iterative pruning strategy with both the Taylor Pruning criterion and the CosPrune importance criterion separately on the model initialized with the trained NYUv2 weights and ran 6 sets of experiments to accommodate for the variance in each method. We present a subset of evaluation metrics in Figure 1, where we show the best values for different task metrics (y-axis) against the number of parameters (x-axis) for both methods. We have shown plots for Pixel Accuracy (Semantic Segmentation), relative error (Depth Estimation), and Angle Mean (Normal prediction).

From the plots in Figure 1, we can see that the CosPrune method is consistently better than the Taylor pruning method across all the tasks, i.e., it has a relatively lower Pixel Accuracy drop in the case of Semantic Segmentation, and lower error increment in case of Depth Estimation and Normal prediction tasks. We also report numerical values in Table I where we compare both the methods for the pruned models with a similar number of parameters. We can see that there is the highest performance gap at $14.3M$ model parameters which is 83% parameter reduction, and Pixel Accuracy is $\mathbf{16.39}\%$ better, Depth Estimation is $\mathbf{15.93}\%$ better and Normal prediction is $\mathbf{3.45}\%$ better than the Taylor pruning method. However, when with extreme pruning at $13.4M$ parameters, the gap is significantly lesser but nonetheless still better. For $59.9M$ parameters, the Pixel Accuracy slightly drops for CosPrune but all the metrics are very close as can also be seen from the plots in Figure 1.

Re-training the Pruned models: So far we have observed that the CosPrune can efficiently prune the MTL model to achieve high parameter reduction within a few iterations while maintaining its performance over the baseline method. Both pruning criteria lead to pruned models with different architectural designs in terms of the number of filters in each layer of the backbone. Since there is a difference in the performance, we want to figure out the role of these different layer configurations in the model's performance for all the tasks. To do this, we take the pruned models at various iterations, randomly initialize the weights and train them to converge on the NYUv2 dataset. Initially, we were using the same hyperparameters for training all the models but saw that some models were severely affected. For example, a model with $15M$ parameters, obtained from CosPrune could achieve only 25.5% Pixel Accuracy even after training for 500 epochs. But when the learning rate of 0.0001 was used it reached over 42%, and the performance of the other two tasks was also improved. In another case, a model obtained from Taylor Pruning with $14M$ parameters resulted in 38% Pixel Accuracy

#Params		Semantic Segmentation Pixel Accuracy (%) ↑			Depth Estimation Relative Error ↓			Surface Normal Angle Mean ↓		
$\#(M)$	$\%_R$	Taylor	CosPrune	$\%\Delta\uparrow$	Taylor	CosPrune	$\%\Delta\downarrow$	Taylor	CosPrune	$\%\Delta(\downarrow)$
59.9	28.7	**45.96**	45.91	− 0.11	0.2616	**0.2614**	− **0.08**	18.54	**18.37**	**−0.92**
42.7	49.2	45.66	**45.70**	+ **0.09**	**0.2666**	0.2670	+ 0.17	18.35	**18.22**	**−0.73**
21.2	74.8	43.02	**44.86**	+ **4.27**	0.2862	**0.2777**	− **2.98**	18.19	**17.19**	**−1.53**
17.6	79.1	41.56	**42.98**	+ **3.41**	0.3384	**0.2829**	**−16.42**	18.16	**18.03**	**−0.71**
14.3	83.0	35.83	**41.70**	**+16.39**	0.3896	**0.3276**	**−15.93**	18.65	**18.00**	**−3.45**
13.4	84.0	34.88	**36.08**	+ **3.43**	0.3928	**0.3747**	− **4.59**	18.82	**18.39**	**−2.26**

TABLE I: Iterative pruning results at selected iterations. $\#(M)$: *Number of model parameters in millions*. $\%_R$: $\%$ *parameter reduction with respect to full model*. $\%\Delta\uparrow(\downarrow)$: $\%$ *increase (decrease) with respect to Taylor pruning*. $\uparrow$: *Higher the better* $\downarrow$: *Lower the better*.



(a) Pixel Accuracy (b) Relative Error (c) Angle Mean

Fig. 1: Iterative pruning performance trends for different tasks for the Taylor and CosPrune methods.

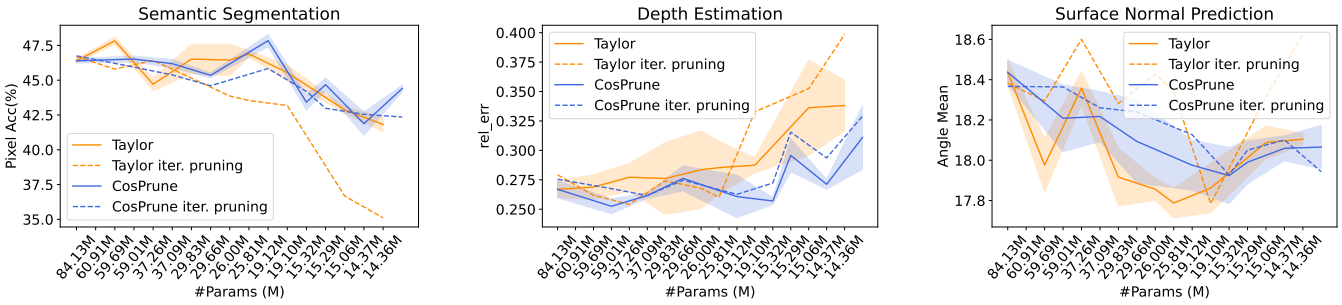
#Param		lr setting		Semantic Segmentation Pixel Accuracy (%) ↑		Depth Estimation Relative Error ↓		Surface Normal Angle Mean ↓	
$\#(M)$	$\%_R$	Taylor	CosPrune	Taylor	CosPrune	Taylor	CosPrune	Taylor	CosPrune
60.3	28.3	0.0005	0.0001	**47.82**	46.52	0.2688	**0.2525**	**17.97**	18.20
37.1	55.8	0.001	0.0001	**46.51**	46.18	0.2761	**0.2614**	**17.91**	18.21
29.7	64.6	0.001	0.0001	**46.44**	45.34	0.2835	**0.2761**	**17.85**	18.09
25.9	69.2	0.001	0.0005	46.86	**47.84**	0.2853	**0.2604**	**17.78**	17.97
19.1	77.3	0.001	0.0001	**45.54**	43.42	0.2874	**0.2570**	**17.86**	17.92
15.3	81.8	0.001	0.0001	42.86	**44.67**	0.3362	**0.2957**	18.08	**17.99**
14.4	82.9	0.001	0.001	41.82	**44.40**	0.3381	**0.3109**	18.10	**18.06**

TABLE II: Results of the pruned models trained from scratch across different tasks. $\#(M)$: *Number of model parameters in millions*. $\%_R$: $\%$ *parameter reduction with respect to full model*. lr setting: *Optimal learning rate for the respective model*. $\uparrow$: *Higher the better*; $\downarrow$: *Lower the better*.

with 0.0001 learning rate, whereas the same model reached over 44% with 0.001 learning rate. Moreover, some pruned models gave their best results for 0.0005 learning rate.

Therefore, to get the best possible performance from every pruned model, we trained them on 3 sets of learning rates which are 0.001, 0.0005, and 0.0001. We ran all the experiments for 500 epochs with cosine scheduling and used Adam [51] optimizer for training. We summarize results for a subset of models in Table II. Each row shows the performance comparison between pruned models obtained from Taylor pruning and CosPrune which have the same number of parameters. We have also given the desired learning rate for each model. It seems that Taylor pruned models favor 0.001 learning rate with a few favoring 0.0001, and 0.0005. On the other hand, CosPruned models favor smaller learning rates of 0.0001, and 0.0005. The corresponding plots for the 3 tasks are shown in Figure 2.

From the results in Table II, we can see that the models obtained from different pruning methods lead to very similar performance across all the tasks when the number of parameters is roughly equal. Figure 2, also shows this similar trend where the winning method keeps on fluctuating with changing model parameters. In some cases, Taylor pruned models perform better than CosPruned models and in other cases, it is vice-versa but the results are still very close for both methods across all tasks. We have also compared the results of the individually pruned models with the results from the corresponding iterative pruning iterations in Figure 2. It can be seen that at very low parameters, the iterative pruning results become worse than the results of the models that were trained



(a) Pixel Accuracy (b) Relative Error (c) Angle Mean

Fig. 2: Results of the pruned models, trained individually from scratch, across all the tasks for various model parameters. Solid plot lines correspond to primary results. Dashed plots correspond to the iterative pruning trends from Figure 1.

solely from scratch except in the case of normal estimation where the Angle Mean is slightly better in the case of the iterative CosPrune method.

C. Results Discussion

We observed that if we consider the pruned models obtained from two different pruning methods, randomly initializing and re-training with their corresponding optimal learning rate give similar results for the same number of parameters. At some parameter levels, the pruned model obtained from Taylor pruning performed better but at some other parameter levels, the pruned model obtained from CosPrune gave better results. But there is only a slight difference in the results which could be induced by the randomness of the training process. By looking at these results, we can conclude that *different architecture configurations with a similar number of parameters do not play a major role in the final performance across tasks as long as the models are trained with their optimal training settings.*

Furthermore, we saw that the structured iterative pruning was biased towards one pruning criterion over the other as Figure 1 shows that the CosPrune models outperformed Taylor pruned models at the same parameter level. But when the pruned models were re-trained from scratch, their final performances became similar and even closer to the unpruned model as seen in Figure 2. But for both methods, there was a steep drop in the performance at extreme pruning ratios, which was again mitigated by random initialization and re-training that led to better performance even at those extreme pruning ratios. This might be happening because, in the case of iterative structured pruning, the inherited trained weights from previous pruning iterations may act as bad initialization for the pruned models at successive iterations [14]. Even if those weights were optimal for the earlier unpruned model versions, they might not be good for the current version because of changes in the architecture after pruning and it can be difficult for the optimizer to find another good local minimum.

V. Conclusions

In this work, we used two different structured pruning methods to compress deep multi-task neural networks. We are first to comprehensively analyze their effectiveness in pruning such networks. Specifically, we showed that different architectural configurations of the pruned models can give similar results regardless of the pruning method used if the number of parameters is the same. We also showed that iterative structured pruning may not be the best strategy to compress deep multi-task models. They might favor one pruning strategy over another. But regardless of the pruning strategy used, the performance of the pruned models can take a steep drop after certain iterations. However, we also showed that after random initialization and re-training those models with their respective optimal learning rates, they can give much higher performance across all the tasks.

Acknowledgment

This material is based upon work supported by the National Science Foundation under Grant No. CNS-2312396, CNS-2338512, CNS-2224054, and DMS-2220211. Any opinions, findings, conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the National Science Foundation.

References

[1] O. Sener and V. Koltun, "Multi-task learning as multi-objective optimization," *Advances in neural information processing systems*, vol. 31, 2018. 1, 2

[2] J. Phillips, J. Martinez, I. A. Bârsan, S. Casas, A. Sadat, and R. Urtasun, "Deep multi-task learning for joint localization, perception, and prediction," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2021, pp. 4679–4689. 1

[3] Y. LeCun, J. Denker, and S. Solla, "Optimal brain damage," *Advances in neural information processing systems*, vol. 2, 1989. 1, 2

[4] J. Frankle and M. Carbin, "The lottery ticket hypothesis: Finding sparse, trainable neural networks," *arXiv preprint arXiv:1803.03635*, 2018. 1, 2

[5] H. Li, A. Kadav, I. Durdanovic, H. Samet, and H. P. Graf, "Pruning filters for efficient convnets," *arXiv preprint arXiv:1608.08710*, 2016. 1, 2

[6] P. Molchanov, A. Mallya, S. Tyree, I. Frosio, and J. Kautz, "Importance estimation for neural network pruning," in *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, 2019, pp. 11 264–11 272. 1, 2, 3

[7] N. Lee, T. Ajanthan, and P. H. Torr, "Snip: Single-shot network pruning based on connection sensitivity," *arXiv preprint arXiv:1810.02340*, 2018. 1, 2

[8] U. Evci, T. Gale, J. Menick, P. S. Castro, and E. Elsen, "Rigging the lottery: Making all tickets winners," in *International Conference on Machine Learning*. PMLR, 2020, pp. 2943–2952. 1

[9] X. Ding, T. Hao, J. Tan, J. Liu, J. Han, Y. Guo, and G. Ding, "Resrep: Lossless cnn pruning via decoupling remembering and forgetting," in *Proceedings of the IEEE/CVF International Conference on Computer Vision*, 2021, pp. 4510–4520. 1

[10] Z. Hou, M. Qin, F. Sun, X. Ma, K. Yuan, Y. Xu, Y.-K. Chen, R. Jin, Y. Xie, and S.-Y. Kung, "Chex: channel exploration for cnn model compression," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2022, pp. 12 287–12 298. 1

[11] H. Cheng, Z. Wang, L. Ma, X. Liu, and Z. Wei, "Multi-task pruning via filter index sharing: A many-objective optimization approach," *Cognitive Computation*, vol. 13, pp. 1070–1084, 2021. 1

[12] X. He, D. Gao, Z. Zhou, Y. Tong, and L. Thiele, "Pruning-aware merging for efficient multitask inference," in *Proceedings of the 27th ACM SIGKDD Conference on Knowledge Discovery & Data Mining*, 2021, pp. 585–595. 1, 2

[13] X. Sun, A. Hassani, Z. Wang, G. Huang, and H. Shi, "Disparse: Disentangled sparsification for multitask model compression," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2022, pp. 12 382–12 392. 1, 2

[14] Z. Liu, M. Sun, T. Zhou, G. Huang, and T. Darrell, "Rethinking the value of network pruning," *arXiv preprint arXiv:1810.05270*, 2018. 2, 6

[15] Y. Li, K. Adamczewski, W. Li, S. Gu, R. Timofte, and L. Van Gool, "Revisiting random channel pruning for neural network compression," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2022, pp. 191–201. 2

[16] S. Liu, T. Chen, X. Chen, L. Shen, D. C. Mocanu, Z. Wang, and M. Pechenizkiy, "The unreasonable effectiveness of random pruning: Return of the most naive baseline for sparse training," *arXiv preprint arXiv:2202.02643*, 2022. 2

[17] D. Kim, T. Lan, C. Zou, N. Xu, B. A. Plummer, S. Sclaroff, J. Eledath, and G. Medioni, "Mila: Multi-task learning from videos via efficient inter-frame attention," in *Proceedings of the IEEE/CVF International Conference on Computer Vision*, 2021, pp. 2219–2229. 2

[18] M. Abdollahzadeh, T. Malekzadeh, and N.-M. M. Cheung, "Revisit multimodal meta-learning through the lens of multi-task learning," *Advances in Neural Information Processing Systems*, vol. 34, pp. 14 632–14 644, 2021. 2

[19] I. Misra, A. Shrivastava, A. Gupta, and M. Hebert, "Cross-stitch networks for multi-task learning," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2016, pp. 3994–4003. 2

[20] D. Eigen and R. Fergus, "Predicting depth, surface normals and semantic labels with a common multi-scale convolutional architecture," in *Proceedings of the IEEE international conference on computer vision*, 2015, pp. 2650–2658. 2, 4

[21] R. Hu and A. Singh, "Unit: Multimodal multitask learning with a unified transformer," in *Proceedings of the IEEE/CVF International Conference on Computer Vision*, 2021, pp. 1439–1449. 2

[22] M. Guo, A. Haque, D.-A. Huang, S. Yeung, and L. Fei-Fei, "Dynamic task prioritization for multitask learning," in *Proceedings of the European conference on computer vision (ECCV)*, 2018, pp. 270–287. 2

[23] G. Ghiasi, B. Zoph, E. D. Cubuk, Q. V. Le, and T.-Y. Lin, "Multi-task self-training for learning general representations," in *Proceedings of the IEEE/CVF International Conference on Computer Vision*, 2021, pp. 8856–8865. 2

[24] C. Williams, S. Klanke, S. Vijayakumar, and K. Chai, "Multi-task gaussian process learning of robot inverse dynamics," *Advances in neural information processing systems*, vol. 21, 2008. 2

[25] T. Yu, D. Quillen, Z. He, R. Julian, K. Hausman, C. Finn, and S. Levine, "Meta-world: A benchmark and evaluation for multi-task and meta reinforcement learning," in *Conference on robot learning*. PMLR, 2020, pp. 1094–1100. 2

[26] Y. Teh, V. Bapst, W. M. Czarnecki, J. Quan, J. Kirkpatrick, R. Hadsell, N. Heess, and R. Pascanu, "Distral: Robust multitask reinforcement learning," *Advances in neural information processing systems*, vol. 30, 2017. 2

[27] S. Sodhani, A. Zhang, and J. Pineau, "Multi-task reinforcement learning with context-based representations," in *International Conference on Machine Learning*. PMLR, 2021, pp. 9767–9779. 2

[28] A. Wilson, A. Fern, S. Ray, and P. Tadepalli, "Multi-task reinforcement learning: a hierarchical bayesian approach," in *Proceedings of the 24th international conference on Machine learning*, 2007, pp. 1015–1022. 2

[29] R. Yang, H. Xu, Y. Wu, and X. Wang, "Multi-task reinforcement learning with soft modularization," *Advances in Neural Information Processing Systems*, vol. 33, pp. 4767–4777, 2020. 2

[30] R. Caruana, "Multitask learning: A knowledge-based source of inductive bias1," in *Proceedings of the Tenth International Conference on Machine Learning*. Citeseer, 1993, pp. 41–48. 2

[31] X. Sun, R. Panda, R. Feris, and K. Saenko, "Adashare: Learning what to share for efficient deep multi-task learning," *Advances in Neural Information Processing Systems*, vol. 33, pp. 8728–8740, 2020. 2

[32] Y. Lu, A. Kumar, S. Zhai, Y. Cheng, T. Javidi, and R. Feris, "Fully-adaptive feature sharing in multi-task networks with applications in person attribute classification," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2017, pp. 5334–5343. 2

[33] A. Kendall, Y. Gal, and R. Cipolla, "Multi-task learning using uncertainty to weigh losses for scene geometry and semantics," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2018, pp. 7482–7491. 2

[34] S. Ruder12, J. Bingel, I. Augenstein, and A. Søgaard, "Learning what to share between loosely related tasks," *arXiv preprint arXiv:1705.08142*, 2017. 2

[35] B. Hassibi and D. Stork, "Second order derivatives for network pruning: Optimal brain surgeon," *Advances in neural information processing systems*, vol. 5, 1992. 2

[36] C. Louizos, K. Ullrich, and M. Welling, "Bayesian compression for deep learning," *Advances in neural information processing systems*, vol. 30, 2017. 2

[37] S. Han, J. Pool, J. Tran, and W. Dally, "Learning both weights and connections for efficient neural network," *Advances in neural information processing systems*, vol. 28, 2015. 2

[38] J. Frankle, G. K. Dziugaite, D. M. Roy, and M. Carbin, "The lottery ticket hypothesis at scale," *arXiv preprint arXiv:1903.01611*, 2020. 2

[39] M. Zhu, T. Zhang, Z. Gu, and Y. Xie, "Sparse tensor core: Algorithm and hardware co-design for vector-wise sparse neural networks on modern gpus," in *Proceedings of the 52nd Annual IEEE/ACM International Symposium on Microarchitecture*, 2019, pp. 359–371. 2

[40] H. Cheng, Z. Wang, L. Ma, X. Liu, and Z. Wei, "Multi-task pruning via filter index sharing: A many-objective optimization approach," *Cognitive Computation*, vol. 13, pp. 1070–1084, 2021. 2

[41] X. He, Z. Zhou, and L. Thiele, "Multi-task zipping via layer-wise neuron sharing," *Advances in Neural Information Processing Systems*, vol. 31, 2018. 2

[42] T. Yu, S. Kumar, A. Gupta, S. Levine, K. Hausman, and C. Finn, "Gradient surgery for multi-task learning," *Advances in Neural Information Processing Systems*, vol. 33, pp. 5824–5836, 2020. 3

[43] J.-A. Désidéri, "Multiple-gradient descent algorithm (mgda) for multi-objective optimization," *Comptes Rendus Mathematique*, vol. 350, no. 5-6, pp. 313–318, 2012. 3

[44] B. Liu, X. Liu, X. Jin, P. Stone, and Q. Liu, "Conflict-averse gradient descent for multi-task learning," *Advances in Neural Information Processing Systems*, vol. 34, pp. 18 878–18 890, 2021. 3

[45] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," *arXiv preprint arXiv:1409.1556*, 2014. 4

[46] L.-C. Chen, G. Papandreou, I. Kokkinos, K. Murphy, and A. L. Yuille, "Semantic image segmentation with deep convolutional nets and fully connected crfs," *arXiv preprint arXiv:1412.7062*, 2014. 4

[47] N. Silberman, D. Hoiem, P. Kohli, and R. Fergus, "Indoor segmentation and support inference from rgbd images." *ECCV (5)*, vol. 7576, pp. 746–760, 2012. 4

[48] D. Eigen, C. Puhrsch, and R. Fergus, "Depth map prediction from a single image using a multi-scale deep network," *Advances in neural information processing systems*, vol. 27, 2014. 4

[49] I. Loshchilov and F. Hutter, "Sgdr: Stochastic gradient descent with warm restarts," *arXiv preprint arXiv:1608.03983*, 2016. 4

[50] ——, "Decoupled weight decay regularization," *arXiv preprint arXiv:1711.05101*, 2017. 4

[51] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," *arXiv preprint arXiv:1412.6980*, 2014. 5