CS 330 Fall 2023 - Project Report: Efficient Task Relationship Estimation with Dominant Subnet Structures

Nikil Ravi[1], Prateek Varshney[1], Onkar Deshpande[1], and Alex Sun (Mentor)[2]

[1]{nravi,vprateek,onkard}@stanford.edu

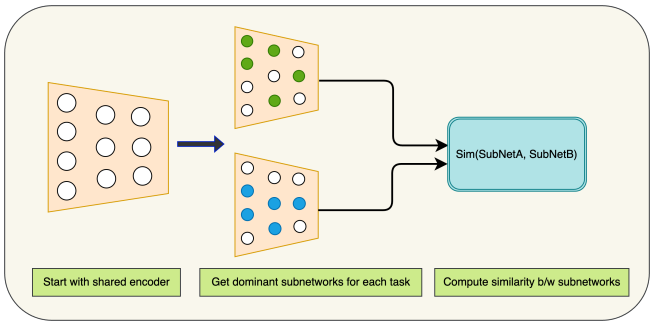
Extended Abstract

Deep multi-task learning involves training a single neural network to handle multiple tasks by leveraging commonalities and shared structure between tasks. The success of multi-task learning relies on the task relationships; the neural network must learn to exploit common information that are useful for multiple tasks. An important notion in this regard is that of *task similarity*. If we could know in advance which tasks are similar, and/or which tasks are unrelated/dissimilar, we could leverage this structure to select tasks to co-train our multi-task network on, allowing it to maximize the amount of information it can reuse across tasks. This becomes especially relevant when computational costs are prohibitive since one would have to train over all possible combinations of task groupings and select the grouping with the best co-training performance.

Previous methods to calculate task similarity Zamir et al. (2018) Fifty et al. (2021) involve expensive computational training or optimization procedures, rendering it extremely difficult for users to determine task relationships before beginning training. Our project explores a novel, computationally efficient approach for determining task similarity that requires no training.

Our method draws inspiration from model pruning literature. Intuitively, **tasks having strong relationships should exhibit similar dominant sub-network structures within the shared encoder**. We use a SNIP-like criterion (no training required!) to get importance scores that give us dominant subnetworks and compute the similarity between these task-specific subnetworks to quantitatively estimate the similarity between tasks.

We propose multiple metrics to compute sub-network similarity and evaluate our method on the NYUv2 dataset. To validate the usefulness of our similarity measure, we train multi-task models on pairs of tasks with high similarity scores and track whether we obtain improvement in task-specific performance and computational tractability. Our results show that using MCSS (Masked Cosine Similarity) score as a measure of similarity between subnetworks allows us to predict task relationships accurately with no training or computationally expensive procedures. Our code is available at this URL.



	seg/sn	seg/depth	sn/depth
EPI-Sim	0.876	0.787	0.855
IOU	0.783	0.712	0.751
CSS	0.970	0.976	0.964
MCSS	**0.865**	**0.838**	**0.835**

Figure 1: Left: An overview for our approach to calculate Task Similarity. Right: MCSS captures task-relationships for different tasks on NYUv2 Dataset

1 INTRODUCTION

Multi-task learning aims to exploit shared structures and commonalities between different tasks to improve the training efficiency and/or performance on individual tasks vis-à-vis training models for each task separately. An important paradigm in this approach is the notion of task "similarity": naive joint training on all the tasks can lead to significant performance degradation while brute-force searches for optimal task grouping can have extremely high computational complexity. Existing methods approach measuring task relationships as an optimization problem: jointly train over all tasks $\rightarrow$ measure (pairwise) task similarities $\rightarrow$ identify optimal task groupings $\rightarrow$ evaluate groupings on test dataset. However, such approaches can be too slow and inefficient for practical purposes, especially with ever-increasing dataset and model sizes. To overcome this, we draw inspiration from model pruning methods. Intuitively, **tasks having strong relationships should exhibit similar dominant sub-network structures within the shared encoder**. We use existing pruning-based techniques to uncover task-specific subnetwork structures efficiently (with no training, and introduce metrics to evaluate the similarity of these subnetwork structures. To validate the usefulness of our similarity measures, we also train multi-task models on pairs of tasks, and report whether we obtain improvement in task-specific performance when the corresponding task similarity is high.

2 RELATED WORK

Task Relationships The work in Taskonomy Zamir et al. (2018) presents an approach to compute task relationships. They provide a computational approach to map the space of visual tasks- in other words, they attempt to answer the question: which visual tasks supply useful information to other tasks? They provide a computational method to produce a directed hypergraph that captures task similarity. They show that the total number of labeled data points needed for solving a set of 10 tasks can be reduced by roughly $\frac{2}{3}$ (compared to training independently) while keeping the performance nearly the same. This work, however, uses transfer learning to estimate task relationships, and Standley et al. (2020) gives experimental evidence that transfer relationships are not highly predictive of multi-task relationships. In addition, it requires training multiple task-specific models, followed by transfer networks, which is very expensive. We would like a method that requires little to no training, such that a user is able to determine task relationships efficiently before deciding which tasks to co-train.

Task-affinity groupings (TAG) Fifty et al. (2021) provides an approach for computing task affinity scores by training all tasks together using a shared network, computing (asymmetric) inter-task affinity scores based on the change in loss function values, which are then used to identify efficient task groupings. Other methods such as Peng et al. (2023) use directional similarities between the gradient vectors for different tasks to employ appropriate curriculum learning strategy-based task weights for the multi-task learning objective. Such methods result in computationally expensive optimization and training-based procedures, due to the (at least) two forward and and one backward pass for each model training step.

We aim to resolve these issues by drawing from model pruning literature (thereby making our approach more computationally efficient), while still using a multi-task training framework.

Model Pruning Although deep learning has become the preferred AI approach in many fields due to its remarkable success, the size and efficiency of models continue to be significant challenges in deploying these trained models. Model pruning has emerged as an effective compression technique for neural networks. Pruning aims to remove weights from a model, such that the model does not deteriorate in performance. In other words, given a neural network parameterized by $\mathbf{\Theta} \in \mathbb{R}^k$ we want to find a mask $\mathbf{B} \in \{0,1\}^k$ such that $\mathbf{\Theta} \odot \mathbf{B}$ retains the performance of $\mathbf{\Theta}$. This field has a rich history LeCun et al. (1989) Hassibi et al. (1993) Cheng et al. (2023), with several types of methods proposed. One commonly used approach is to simply mask out weights below a certain magnitude threshold. Other methods rely on gradient-based importance scores, or saliency scores. SNIP Lee et al. (2019) (Single-Shot Network Pruning) is an approach

that prunes parameters at initialization (prior to training) by identifying structurally important connections via a gradient-based saliency criterion. DiSparse Sun et al. (2022) extends this work to a multi-task setting, introducing a method to uncover dominant subnetwork structures corresponding to each task. We leverage these dominant substructures in our project.

Subnetwork similarity As mentioned earlier, our aim is to quantitatively estimate the similarity between tasks using sub-network similarity. Early Pruning Indicators Shen et al. (2022) provides a quantitative technique to compare these dominant sub-networks. One disadvantage of this metric is that it uses only the number of non-masked parameters in its calculation; it does not take into account their positions. We therefore propose new metrics to compute sub-network similarity that take the structure of the subnetwork into account.

3 Problem Statement and Datasets

3.1 Problem Description and Technical Approach

Notations: For a matrix $\mathbf{A}$, $\mathbf{A}_i$ denotes i^{th} row of $\mathbf{A}$. For a vector $\mathbf{x}$, x_i denotes i^{th} element of $\mathbf{x}$. $\|\cdot\|_2$ denotes euclidean norm of a vector and $\|\cdot\|_0$ will denote the ℓ_0 norms of a vector respectively. For a sparse vector $\mathbf{v} \in \mathbb{R}^d$, we define the support $\mathsf{supp}(\mathbf{v}) \subseteq [d]$ to be the set of indices s.t. $v_i \neq 0 \; \forall \; i \in \mathsf{supp}(\mathbf{v})$ and $v_i = 0$ otherwise. Similarly, we also use $\mathsf{supp}(\cdot)$ to denote compute the non-zero index (mask) of a matrix. Lastly, we use the notation $\mathsf{dim}(\mathbf{A})$ to denote the dimensionality of a matrix. For example, $\mathbf{A} \in \mathbb{R}^{m \times n} \implies \mathsf{dim}(\mathbf{A}) = m \times n$

Given tasks $T = \{T_1, T_2, \cdots T_k\}$ we are interested in computing pairwise similarities for every pair of tasks T_i, T_j. In this project, we use a standard multi-task architecture consisting of a shared backbone and task-specific heads; the architecture is described in 3.3. We will produce a sub-network structure in the shared backbone $\mathbf{B}_i$ (i.e a mask or saliency scores) corresponding to each task T_i. Then, we compute $\binom{k}{2}$ similarity scores (one for each pair of subnetworks).

Let $\text{Sim}(\cdot, \cdot)$ denote the similarity score function (over pairs of tasks) as determined by common dominant sub-networks, and Acc(A | B) is the accuracy obtained on task A when we co-train on tasks A and B. For any two tasks A and B, if Sim(A, B) $>$ Sim(A, C) then we expect that Acc(A | B) $>$ Acc(A | C). Our goal is to validate this assertion/hypothesis. To this end, we also run $\binom{k}{2}$ training runs; in each of these runs, we co-train 2 tasks at a time, and store the accuracy corresponding to each trained network- this is the desired output from our work. The novelty in our approach stems from the fact that we are using model pruning literature to attempt to identify task similarity/groupings- this means our similarity measurement requires no training at all, and to the best of our knowledge this has not been done before.

3.2 Dataset and Tasks

We use the popular multi-task dataset NYU-V2 Nathan Silberman & Fergus (2012), comprised of video sequences from a variety of indoor scenes as recorded by both the RGB and Depth cameras.

We consider three tasks- Semantic Segmentation (`sn`), Surface Normalization (`sn`) and Depth Prediction (`depth`). We essentially use the same data configurations, tasks and task-specific accuracy metrics (tabulated in section 4) as in Sun et al. (2022). For an explanation of these metrics, please refer to Sun et al. (2022).

3.3 Multi-Task Architecture and Training

We trained a multi-task model on pairs of these tasks- that is, one (co-trained) multi-task model for each of the following pairs: (*sn*, *seg*), (*sn*, *depth*), and (*seg*, *depth*), resulting in three trained models in total. We use the architecture used in DiSparse Sun et al. (2022); this architecture uses DeepLab-ResNet as

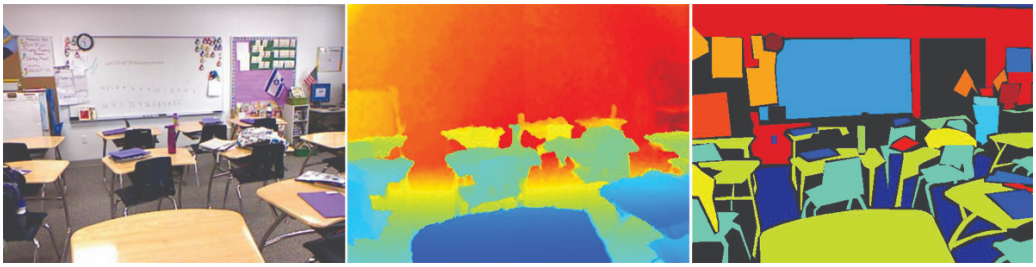


Figure 2: Sample from the NYUv2 dataset. The images show the RGB image (left), preprocessed depth (center), and a set of labels (right) for the image. Image and caption are taken from the NYUv2 website

a shared backbone, and ASPP for task-specific heads. We used the code available at `https://github.com/SHI-Labs/DiSparse-Multitask-Model-Compression/` to run our experiments. Each of our 3 models were trained for 5000 iterations, with task weightings of 1, 20 and 3 for *seg*, *sn* and *depth* respectively. Each of the models was trained on a single Azure GPU. Loss curves are attached in section 4.

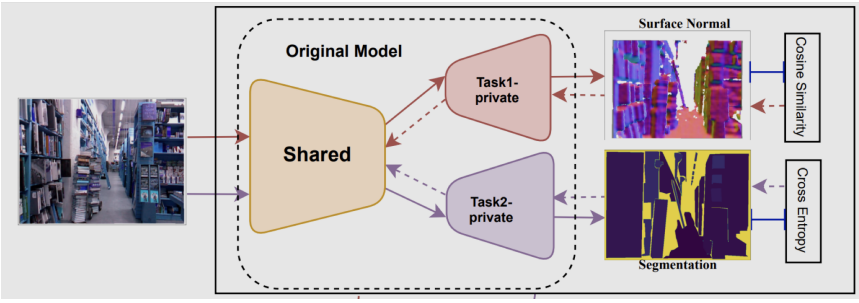


Figure 3: Multi-task Architecture (we additionally included depth prediction as a third task). Image taken from Sun et al. (2022)

3.4 Identifying Dominant Sub-Networks

A popular method to measure the importance of a neuron (entry in a weight matrix) is the magnitude based criterion: higher the magnitude, more the contribution of the neuron towards the layer activation and the final model output. Therefore, one method to prune a network is to sort the model weight entries in descending order of their magnitude and pick the top-k entries while zeroing out the remaining entries to induce sparsity. However, such a pruning method work well when the model pruning and optimization steps are alternated many times till convergence during the training procedure. The highly expensive prune - retrain cycles for large-scale models make the training intractable. A *relatively* less expensive approach is to fully fine-tune the model and then do a 1-shot pruning at the end of the training. However, such an approach is still impractical for our purposes as this would entail fully fine-tuning over all-possible pairs of tasks, and then computing task similarities using the dominant sub-networks for each task.

Since our objective is to measure task-similarity *before* executing any training procedure, we cannot use magnitude based relevance scores as a measure of the dominant sub-structure. Instead, we will use gradient based criterion from SNIP to compute the relevance scores for each neuron. First, we will introduce auxiliary indicator variables (masks) $\mathbf{B} \in \{0,1\}_{m \times n}$ representing the connectivity of the parameters in the l^{th} layer. Therefore, we now have twice the number of actual model parameters with $\mathbf{B}_{(l)} = \mathsf{supp}(\mathbf{\Theta}_{(l)})$ for each layer.

Therefore, for each layer, we have $\mathbf{\Theta}_{(l)} = \mathbf{W}_{(l)} \odot \mathbf{B}_{(l)}$. Next, we compute the gradient of the Loss function $\mathcal{L}(\cdot)$ w.r.t the mask variables by relaxing the binary constraints on the masks:

$$g^k_{(l)} = \frac{\partial \mathcal{L}(\mathbf{W}_{(l)} \odot \mathbf{B}_{(l)}; D)}{\partial \mathbf{B}^k_{(l)}}$$

where k iterates over individual neurons of the l^{th} layer weight and D is the Dataset. Now, note that if $|g^k_{(l)}|$ is high, then it essentially means that the connection $\mathbf{B}^k_{(l)}$ has a considerable effect on the loss $\mathcal{L}$, and it has to be preserved to allow learning on $\mathbf{W}^k_{(l)}$ (and by implication $\mathbf{\Theta}^k_{(l)}$). Based on this, SNIP defines the connection sensitivity/saliency score matrix as the normalized magnitude of the derivatives:

$$\text{SAL}(\mathbf{\Theta}_{(l)})^k = \frac{|g^k_{(l)}|}{\sum_k^{\mathsf{dim}(\mathbf{\Theta}_{(l)})} |g^k_{(l)}|}$$

Once the saliency scores are computed for each neuron, we pick only the top-K connections (and zero out the mask values for all other neurons), where K denotes the desired number of non-zero weights. Therefore, we are able to eliminate the need for any kind of training procedure and make the task similarity measure significantly faster.

3.5 Similarity Metrics

For our metrics, we start off with the EPI score as introduced in the DiSparse Paper. Since we want to limit the output range of the metric to $[0, 1]$, with 0 denoting no similarity while 1 denoting very high similarity between the two tasks, we use a modified version of EPI which we call EPI-Sim:

$$\text{EPI-Sim}(T_1, T_2) = 1 - \frac{1}{L} \sum_{l=1}^{L} w_l \frac{|n_{(1,l)} - n_{(2,l)}|}{n_{(1,l)} + n_{(2,l)}}, \tag{1}$$

where $n_{(1,l)} = \mathbf{1}^\mathsf{T} \mathbf{B}_{(1,l)} \mathbf{1}$ and $n_{(2,l)} = \mathbf{1}^\mathsf{T} \mathbf{B}_{(2,l)} \mathbf{1}$ are the number of activated (non-zero) connections of l^{th} layer in the common backbone of Tasks T_1 and T_2 respectively at a certain sparsity level, and w_l is a scalar which denotes how much weight is to be given to the l^{th} layer s.t. $\sum_l w_l = 1$.

EPI-Sim defines subnetwork similarity in terms of the "quantity" of activated connections across different layers. Therefore, a lower value of EPI-sim indicates high variations between the two sub-networks, while a high value indicates stability. However, note that EPI-Sim discards any information about how those activated connections (and neurons) are distributed in the layer weights and does not leverage the saliency scores of the neurons. We find that the inability of EPI-Sim to capture the positions of masked weights (in addition to the count) does not allow it to correlate with task-similarity as presented in section 4.

To account for these, we experiment with three different metrics:

1. Intersection Over Union:

$$\text{IOU}(T_1, T_2) = \frac{1}{L} \sum_{l=1}^{L} w_l \frac{|\mathbf{B}_{(1,l)} \cap \mathbf{B}_{(2,l)}|}{|\mathbf{B}_{(1,l)} \cup \mathbf{B}_{(2,l)}|}, \tag{2}$$

   where, $\mathbf{B}_{(\cdot)}$ stores the indices with non-zero entries of a weight matrix (essentially, $\mathsf{supp}(\mathbf{\Theta}_{(\cdot)})$) and we use $|\cdot|$ to count the number of non-zero entries.
   Note that IOU is more powerful than EPI-Sim since it captures the similarity in the distribution of activated neurons in the layer-specific weight matrices. If two sparse weight matrices have the very similar positional distribution of the non-zero scalars then $\mathbf{B}_{(1,l)}) \sim \mathbf{B}_{(2,l)} \implies \text{IOU}(T_1, T_2) \approx 1$.

On the other hand, if the distribution of the non-zero entries is different for the two layer-specific weight matrices then $\{\mathbf{B}_{(1,l)} \cap \mathbf{B}_{(2,l)}\} \sim \phi \implies \text{IOU}(T_1, T_2) \approx 0$.

2. Cosine Similarity:

$$\text{CSS}(T_1, T_2) = \frac{1}{L}\sum_{l=1}^{L} w_l \text{cosine}\Big(\text{SAL}(\mathbf{\Theta}_{(1,l)}), \text{SAL}(\mathbf{\Theta}_{(2,l)})\Big), \tag{3}$$

where, cosine$(\cdot, \cdot)$ is the cosine function between two vectors and $\text{SAL}(\mathbf{A}) \in \mathbb{R}^{\mathsf{dim}(\mathbf{A})}$ is the saliency vector created by stretching the saliency scores of each entry in the weight matrix $\mathbf{A}$ along a single dimension and $|\mathbf{A}|$ being being the total number of entries in $\mathbf{A}$. Firstly, note that CSS's output range is [-1, 1] and not [0, 1]. Next, CSS focuses on the raw importance scores of the layer-specific weights instead of focusing only on the binarized sparsity masks. Further, the cosine function measures to what extent are the distribution patterns of the salience scores across the weight matrices common between the two tasks, i.e. if the sub-networks for both tasks T_1 and T_2 have a similar distribution for saliency scores over each entry of their respective layer-specific weights, then $\text{CSS}(T_1, T_2) \approx 1$. Therefore, CSS overcomes the shortcoming of both EPI-Sim and IOU. However, from our experiments(see section 4) we found that CSS fails to capture task similarities in some cases. After iterating over different variations, we arrived at our final metric.

3. Masked Cosine Similarity:

$$\text{MCSS}(T_1, T_2) = \frac{1}{L}\sum_{l=1}^{L} w_l \text{cosine}\Big(\mathbf{B}_{(1,l)} \odot \text{Sal}(\mathbf{\Theta}_{(1,l)}), \mathbf{B}_{(2,l)} \odot \text{Sal}(\mathbf{\Theta}_{(2,l)})\Big). \tag{4}$$

The major difference between MCSS and CSS is that we are multiplying the saliency scores with the corresponding sparse mask value. Our hypothesis is that this gets rid of the noisy weights (neurons with low saliency scores with 0 as mask) which are not discriminatory. Instead, we focus MCSS draws comparisons between discriminatory neurons (high saliency scores and 1 as mask) which should enable it to perform better than CSS.

4 Experiments and Results

We start by co-training the model on each pair of tasks: *(seg,sn)*, *(seg,depth)* and *(sn,depth)*. The accuracies we obtained after co-training are tabulated in Appendix A.

From the tables, it is clear that:

$$\begin{aligned} \mathsf{Acc}(seg|sn) &> \mathsf{Acc}(seg|depth) \\ \mathsf{Acc}(sn|seg) &> \mathsf{Acc}(sn|depth) \\ \mathsf{Acc}(depth|seg) &> \mathsf{Acc}(depth|sn) \end{aligned}$$

Based on the accuracy scores (See Appendix A) and our hypothesis, we expect to see

$$\mathsf{Sim}(seg, sn) > \mathsf{Sim}(seg, depth) > \mathsf{Sim}(sn, depth).$$

We experimented with multiple similarity metrics and averaging methods. For every layer, we computed 4 similarity metrics: EPI-Sim, IOU, CSS, MCSS. From our experiments, we found that the computed pruned masks w.r.t the saliency scores can vary quite a lot across the layers of the backbone: the initial layers are not pruned and their corresponding masks remain almost intact while the sparsity of the later layers is quite high. To compensate for this in our metric computations so that each layer is given equal weightage, we experimented with multiple weighting methods($\{w_l\}_{l=1}^{L}$):

1. Equal: All layers have equal weights
2. Linear: All layers given linearly increasing weights with the first layer having zero weight.
3. Step like: First 5 layers are given zero weights and the rest have equal weights
4. Sparsity: Each layer has weight proportional to the average sparsity of that layer ($n_{(l)}$) across the mask for both tasks.

Each experiment was repeated over 4 seed values and the similarity scores were uniformly averaged over all seeds. Results are presented in Tables 1 and 2. Table 1 presents the similarity scores for the sparsity values at which MCSS metric matches the expected trend. Table 2 presents the other cases we experimented with.

Trends across pairs of task are maintained across most sparsity levels as well as layer weighting methods. For example EPI-Sim(seg, sn) > EPI-Sim$(sn, depth)$ > EPI-Sim$(seg, depth)$ holds across all weighting methods. Similarly, MCSS(seg, sn) > MCSS$(seg, depth)$ > MCSS$(sn, depth)$ holds across all weighting methods. Also, MCSS similarity matches the trend predicted by the hypothesis. Thus we can conclude that MCSS is a good similarity metric when it comes to identifying similar pairs of tasks which might be benefical to be co-trained. This can be explained by the fact that EPI and IOU both rely only on the binary masks and CSS relies on all saliency scores. MCSS on the other hand only considers the cosine similarity of dominant saliency scores.

We also note that trends are best observed when the target sparsity levels are closer to 50%. The similarity metrics that we tested with did not seem to follow the expected trends for sparsity ratios of $\leq 30\%$ and $\geq 90\%$. Averaging over multiple sparsity ratios also works well, though is not needed given that a sparsity ratio of 50% works just as well.

5 Conclusion

We hypothesized that $\mathsf{Sim}(A, B) > \mathsf{Sim}(A, C) \implies \mathsf{Acc}(A|B) > \mathsf{Acc}(A|C)$. We tested our hypothesis by computing various similarity metrics over pairs of tasks in the NYU V2 Nathan Silberman & Fergus (2012) and comparing them to obtained accuracies. We conclude that the MCSS 4 metric computed with mid sparsity values is a reliable similarity metric and can be efficiently used to compute similar pairs of tasks. Co-training on similar pairs of tasks thus found will provide improved accuracies.

5.1 Future work

While the work presented in this report covers multiple similarity metrics over multiple averaging methods, there is still much work that can be done to improve confidence in our hypothesis.

- Run similar experiments over other datasets such as Taskonomy Zamir et al. (2018).
- We used co-training accuracy obtained after 5000 training iterations. Since co-training took more than a day for each pair of tasks, we were unable to train for more iterations during the course of the project. We plan to train the model for longer to get better accuracy metrics.

6 Team Contributions

Due to the nature of the project, we all worked together on some common aspects as well as the overall conceptual understanding of the work. We started with the codebase at `https://github.com/SHI-Labs/DiSparse-Multitask-Model-Compression/` and decided to use the NYU-V2 Nathan Silberman & Fergus (2012) dataset. This dataset contains 3 tasks: segmentation(*seg*), surface normal(*sn*) and depth detection(*depth*). After some minor modifications to the code completed together, we co-trained all pairs of tasks, distributing equally among ourselves:

	Sparsities	50			70			30,50,70,90		
		seg/sn	seg/depth	sn/depth	seg/sn	seg/depth	sn/depth	seg/sn	seg/depth	sn/depth
Equal	EPI-Sim	0.967	0.926	0.958	0.888	0.779	0.877	0.876	0.787	0.855
	IOU	0.902	0.854	0.872	0.772	0.667	0.727	0.783	0.712	0.751
	CSS	0.970	0.976	0.964	0.970	0.976	0.964	0.970	0.976	0.964
	MCSS	**0.950**	**0.947**	**0.932**	**0.885**	**0.854**	**0.845**	**0.865**	**0.838**	**0.835**
Linear	EPI-Sim	0.951	0.892	0.940	0.840	0.692	0.831	0.836	0.717	0.808
	IOU	0.851	0.787	0.807	0.668	0.539	0.607	0.698	0.617	0.657
	CSS	0.959	0.966	0.948	0.959	0.966	0.948	0.959	0.966	0.948
	MCSS	**0.925**	**0.919**	**0.896**	**0.824**	**0.780**	**0.760**	**0.804**	**0.770**	**0.760**
Step like	EPI-Sim	0.962	0.914	0.951	0.871	0.745	0.858	0.857	0.754	0.833
	IOU	0.887	0.832	0.852	0.737	0.616	0.684	0.750	0.669	0.713
	CSS	0.967	0.973	0.960	0.967	0.973	0.960	0.967	0.973	0.960
	MCSS	**0.944**	**0.940**	**0.924**	**0.869**	**0.833**	**0.822**	**0.846**	**0.814**	**0.811**
Sparsity	EPI-Sim	0.929	0.849	0.918	0.792	0.614	0.791	0.799	0.668	0.782
	IOU	0.779	0.700	0.724	0.571	0.431	0.518	0.630	0.544	0.595
	CSS	0.948	0.956	0.932	0.954	0.962	0.943	0.952	0.960	0.939
	MCSS	**0.893**	**0.883**	**0.851**	**0.771**	**0.721**	**0.707**	**0.768**	**0.731**	**0.726**

Table 1: Experimental results for sparsities equal to 50%, 70% and average of multiple sparsities. The MCSS similarity metric follows the expected trend for multiple sparsity levels as well as layer weighting methods. Expected trends are marked in bold.

- Onkar: Segmentation and Depth detection
- Nikil: Segmentation and Surface Normal
- Prateek: Surface Normal and Depth Detection

We each co-trained our assigned pairs of tasks for 5000 iterations and noted the obtained accuracies on each of the tasks.

We together wrote the code to compute similarity metrics for dominant subnetwork structures, distributing among us methods to compute the 4 similarity metrics and averaging methods. We analyzed the results together and brainstormed on various similarity metrics as well as their intuition. This culminated in a script that can run all required experiments and generate a JSON with all required similarity scores, while caching intermediate results for faster prototyping. The results have been compiled in Section 4.

REFERENCES

Hongrong Cheng, Miao Zhang, and Javen Qinfeng Shi. A survey on deep neural network pruning-taxonomy, comparison, analysis, and recommendations, 2023.

Chris Fifty, Ehsan Amid, Zhe Zhao, Tianhe Yu, Rohan Anil, and Chelsea Finn. Efficiently identifying task groupings for multi-task learning. In M. Ranzato, A. Beygelzimer, Y. Dauphin, P.S. Liang, and J. Wortman

	Sparsities	30			90		
		seg/sn	seg/depth	sn/depth	seg/sn	seg/depth	sn/depth
Equal	EPI-Sim	0.992	0.977	0.985	0.658	0.465	0.602
	IOU	0.962	0.945	0.948	0.495	0.380	0.458
	CSS	0.970	0.976	0.964	0.970	0.976	0.964
	MCSS	0.967	0.971	0.959	0.660	0.579	0.605
Linear	EPI-Sim	0.987	0.966	0.979	0.566	0.318	0.482
	IOU	0.941	0.917	0.920	0.333	0.226	0.295
	CSS	0.959	0.966	0.948	0.959	0.966	0.948
	MCSS	0.954	0.959	0.940	0.514	0.420	0.444
Step like	EPI-Sim	0.990	0.973	0.983	0.605	0.385	0.541
	IOU	0.956	0.936	0.940	0.419	0.293	0.378
	CSS	0.967	0.973	0.960	0.967	0.973	0.960
	MCSS	0.964	0.969	0.955	0.607	0.513	0.544
Sparsity	EPI-Sim	0.980	0.949	0.969	0.496	0.258	0.448
	IOU	0.907	0.874	0.875	0.264	0.172	0.263
	CSS	0.945	0.953	0.926	0.961	0.969	0.954
	MCSS	0.936	0.941	0.912	0.471	0.378	0.432

Table 2: Experimental results for sparsities equal to 30% and 90%. At these sparsities, no similarity metric follows the trend.

Vaughan (eds.), *Advances in Neural Information Processing Systems*, volume 34, pp. 27503–27516. Curran Associates, Inc., 2021. URL https://proceedings.neurips.cc/paper_files/paper/2021/file/e77910ebb93b511588557806310f78f1-Paper.pdf.

Babak Hassibi, David G. Stork, Gregory Wolff, and Takahiro Watanabe. Optimal brain surgeon: Extensions and performance comparisons. In *Proceedings of the 6th International Conference on Neural Information Processing Systems*, NIPS'93, pp. 263–270, San Francisco, CA, USA, 1993. Morgan Kaufmann Publishers Inc.

Yann LeCun, John S. Denker, and Sara A. Solla. Optimal brain damage. In *Neural Information Processing Systems*, 1989. URL https://api.semanticscholar.org/CorpusID:7785881.

Namhoon Lee, Thalaiyasingam Ajanthan, and Philip H. S. Torr. Snip: Single-shot network pruning based on connection sensitivity, 2019.

Pushmeet Kohli Nathan Silberman, Derek Hoiem and Rob Fergus. Indoor segmentation and support inference from rgbd images. In *ECCV*, 2012.

Xinyu Peng, Cheng Chang, Fei-Yue Wang, and Li Li. Robust multitask learning with sample gradient similarity. *IEEE Transactions on Systems, Man, and Cybernetics: Systems*, pp. 1–10, 2023. doi: 10.1109/TSMC.2023.3315541.

Maying Shen, Pavlo Molchanov, Hongxu Yin, and Jose M Alvarez. When to prune? a policy towards early structural pruning. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2022.

Trevor Standley, Amir R. Zamir, Dawn Chen, Leonidas Guibas, Jitendra Malik, and Silvio Savarese. Which tasks should be learned together in multi-task learning?, 2020.

Xinglong Sun, Ali Hassani, Zhangyang Wang, Gao Huang, and Humphrey Shi. Disparse: Disentangled sparsification for multitask model compression. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 12382–12392, 2022.

Amir R. Zamir, Alexander Sax, William B. Shen, Leonidas J. Guibas, Jitendra Malik, and Silvio Savarese. Taskonomy: Disentangling task transfer learning. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. IEEE, 2018.

A CO-TRAINING PLOTS AND ACCURACIES

Attached are plots of the models' loss curves for 5000 iterations. Note that we have skipped plotting the first few iterations to exclude outliers.

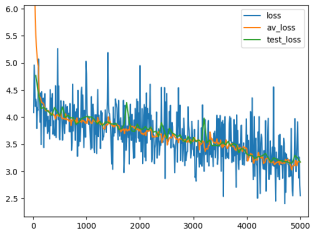


Figure 4: Loss curve for seg, sn

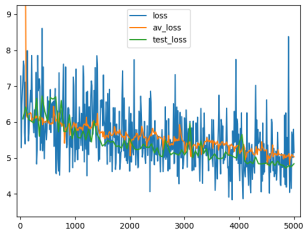


Figure 5: Loss curve for seg, depth.

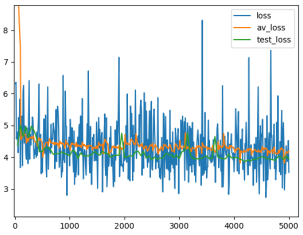


Figure 6: Loss curve for sn, depth

The accuracies we obtained are tabulated below:

	Angle Mean	Angle Median	11.25	22.5	30	45
Acc(*sn*|*seg*)	17.49	15.69	29.65	75.96	87.16	96.97
Acc(*sn*|*depth*)	18.86	16.71	24.50	70.08	87.19	97.13

Table 3: Accuracies on `sn` when co-trained with `seg` and `depth`

	mIoU	Pixel Acc.	Error
Acc(*seg*|*sn*)	0.16	0.47	1.90
Acc(*seg*|*depth*)	0.10	0.37	2.31

Table 4: Accuracies on `seg` when co-trained with `sn` and `depth`

	Abs. Err.	Rel. Err.	$\delta 1.25$	$\delta 1.25^2$	$\delta 1.25^3$
Acc(*depth*|*sn*)	0.82	0.35	45.3	74.88	89.87
Acc(*depth*|*seg*)	0.80	0.34	46.85	76.26	90.63

Table 5: Accuracies on `depth` when co-trained with `seg` and `sn`