Which Tasks Should Be Learned Together in Multi-task Learning?

Trevor Standley[1] Amir R. Zamir[1,2] Dawn Chen[3] Leonidas Guibas[1] Jitendra Malik[2] Silvio Savarese[1]

[1]Stanford University [2]The University of California, Berkeley [3]Google Inc.

http://taskgrouping.stanford.edu/

Abstract

Many computer vision applications require solving multiple tasks in real-time. A neural network can be trained to solve multiple tasks simultaneously using 'multi-task learning'. This saves computation at inference time as only a single network needs to be evaluated. Unfortunately, this often leads to inferior overall performance as task objectives compete, which consequently poses the question: ***which tasks should and should not be learned together in one network when employing multi-task learning?*** *We systematically study task cooperation and competition and propose a framework for assigning tasks to a few neural networks such that cooperating tasks are computed by the same neural network, while competing tasks are computed by different networks. Our framework offers a time-accuracy trade-off and can produce better accuracy using less inference time than not only a single large multi-task neural network but also many single-task networks.*

1. Introduction

Many applications, especially robotics and autonomous vehicles, are chiefly interested in using multi-task learning to reduce the inference time required to estimate many characteristics of visual input. These estimates must be produced quickly because the reaction time of such a robot is of utmost importance. For example, an autonomous vehicle may need to detect the location of pedestrians, determine a per-pixel depth, and predict objects' trajectories, all within 15 milliseconds. In multi-task learning, multiple learning tasks are solved at the same time, typically with a single neural network. In addition to reduced inference time, solving a set of tasks jointly rather than independently can, in theory, have other benefits such as improved prediction accuracy, increased data efficiency, and reduced training time.

Unfortunately, the quality of predictions often suffers when a network is tasked with making multiple predictions. This is because learning objectives can have complex and unknown dynamics and may compete. In fact, multi-task performance can suffer so much that smaller independent networks are often superior (as we will see in the experiments section). We refer to any situation in which the competing priorities of the network cause poor task performance as *crosstalk*.

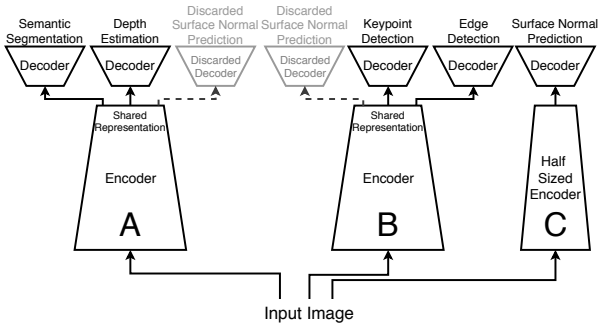

Figure 1. **Given five tasks to solve, there are many ways that they can be split into task groups for multi-task learning. How do we find the best one?** We propose a computational framework that, for instance, suggests the following grouping to achieve the lowest total loss, using a computational budget of 2.5 units: train network A to solve Semantic Segmentation, Depth Estimation, and Surface Normal Prediction; train network B to solve Keypoint Detection, Edge Detection, and Surface Normal Prediction; train network C with a less computationally expensive encoder to solve Surface Normal Prediction alone; including Surface Normals as an output in the first two networks were found advantageous for improving the other outputs, while the best Normals were predicted by the third network. This task grouping outperforms all other feasible ones, including learning all five tasks in one large network or using five dedicated smaller networks.

On the other hand, when task objectives do not interfere much with each other, performance on both tasks can be maintained or even improved when jointly trained. Intuitively, this loss or gain of quality seems to depend on *the relationship between the jointly trained tasks*.

Prior work has studied the relationship between tasks for transfer learning[44]. However, we find that multi-task relationships are not closely related to transfer relationships. In addition to studying multi-task relationships, we attempt to determine how to produce good prediction accuracy under a limited inference time budget by assigning competing tasks to separate networks and cooperating tasks to the same

network.

This leads to the following problem: Given a set of tasks, $\mathcal{T}$, and a computational budget b (e.g., maximum allowable inference time), what is the optimal way to assign tasks to networks with combined cost $\leq b$ such that a combined measure of task performance is maximized?

To this end, we develop a computational framework for choosing the best tasks to group together in order to have a small number of separate deep neural networks that completely cover the task set and that maximize task performance under a given computational budget. Note that the inclusion of an additional task in a network can potentially improve the accuracy that can be achieved on the existing tasks, even though the performance of the added task might be poor. This can be viewed as regularizing or guiding the loss of one task by adding an additional loss. Our system can take advantage of this phenomenon, as shown in Figure 1.

In this work, we analyze the compatibility of various tasks for multi-task learning, and compare that to the transfer learning task affinity in [44]. We also contribute a framework for systematically assigning tasks to networks to achieve the best total prediction accuracy with a limited budget. We then show how one can reduce the training-time burden of our framework. Finally, we analyze the resulting prediction accuracy and show the importance of selecting the best assignment of tasks to groups.

2. Prior Work

2.1. Multi-Task Learning

The authors in [37] provide a good overview of techniques for multi-task learning. They identify two clusters of techniques that we believe cover the space well, hard parameter sharing and soft parameter sharing. The primary difference between the majority of the existing works and our study is that we wish to understand the relationships between tasks and find compatible groupings of tasks for any given set of tasks, rather than designing a neural network architecture to solve a particular set of tasks well. For another good survey of multi-task learning, see [46].

2.1.1 Hard Parameter Sharing

In hard parameter sharing, hidden layers are shared between all tasks to produce representations and task-specific layers are used to produce a prediction for each task. Hard parameter sharing requires specifying a set of relative weights (or learning rates), one for each task's loss. If the weight for a task is too high, it can starve the other tasks of the network capacity they need to perform well. These extra hyperparameters are sometimes tricky to choose, and the best set of parameters can potentially change throughout training. Having equal task weights can sometimes be a good default, but often the different task's losses can create gradients that differ in magnitude by a large factor.

The best contemporary example of hard parameter sharing in computer vision is UberNet[18]. The authors tackle 7 computer vision problems using hard parameter sharing. The authors focus on reducing the computational cost of training for hard parameter sharing, but experience a rapid degradation in performance as more tasks are added to the network. Hard parameter sharing is also used in many other works such as [42, 4, 27, 12, 17, 2, 33, 10, 45, 19, 23, 8, 47, 36].

Other works, such as [39] and [6], aim to dynamically re-weight each task's loss during training. The former work finds weights that provably lead to a Pareto-optimal solution, while the latter attempts to find weights that balance the influence of each task on network weights.

Finally, [3] studies task interaction for natural language processing.

2.1.2 Partial or Soft Parameter Sharing

In soft or partial parameter sharing, either there is a separate set of parameters per task, or a significant fraction of the parameters are unshared. The models are tied together either by information sharing or by requiring parameters to be similar. Examples include [9, 11, 26, 41, 43, 20]

The canonical example of soft parameter sharing can be seen in [11]. The authors are interested in designing a deep dependency parser for languages such as Irish that do not have much treebank data available. They tie the weights of two networsk together by adding an L2 distance penalty between corresponding weights and show substantial improvement.

Another example of soft parameter sharing is Cross-stitch Networks [26]. Starting with separate networks for Semantic Segmentation and Surface Normal Prediction, the authors add 'cross-stitch units' between them, which allow each network to peek at the other network's hiddden layers. This approach reduces but does not eliminate crosstalk, and the overall performance is less sensitive to the relative loss weights.

None of the aforementioned works attempt to discover good sets of tasks to train together. Moreover, soft parameter sharing does not reduce inference time, a major goal of our work.

2.2. Transfer Learning and Task Relationships

Transfer learning, [34, 16, 40, 15, 25, 28, 21, 35, 31], is similar to multi-task learning in that solutions are learned for multiple tasks. Unlike multi-task learning, however, transfer learning methods first learn a model for a source task and then adapt that model to a target task. Trans-

fer learning methods generally do not seek any benefit for source tasks.

Piggyback [22] first learns a general source task and then tries to adapt the learned network to more specific transfer tasks by learning a separate binary mask on its weights for each specific task. Unfortunately, each task requires a separate pass through the network during inference, so this method does not reduce inference time, unlike in our work.

PathNet [14] focuses on transfer learning. It uses alternating gradient and evolution optimization methods to choose the best paths through the network for a particular task. Layers chosen for the source task that are also useful for the transfer task are reused in the latter's paths. This work is optimized for sample efficiency and doesn't attempt to discover good pairs of source and target tasks. It also does not try to meet an inference time budget and does not achieve much inference-time speedup from module reuse.

Progressive Neural Networks [38] achieve good transfer learning results by adding a new network onto an existing source network. They avoid catastrophic forgetting by keeping source parameters fixed, while enabling transfer by giving target network neurons access to the hidden states of the source network.

Our work is most related to *Taskonomy*[44], where the authors studied the relationships between visual tasks for *transfer learning* in depth and introduced a dataset with more than 4 million images and corresponding labels for 26 tasks[1]. This was followed by a number of recent works, which further analyzed task relationships [30, 13, 1] for transfer learning. While they extract relationships between these tasks for transfer learning, we are interested in the multi-task learning setting. We find notable differences between transfer task affinity and multi-task affinity. Their method also differs in that they are interested in labeled-data efficiency and not inference-time efficiency. Finally, we believe that the transfer learning approach taken by Taskonomy is only capable of finding relationships between the high-level bottleneck representations developed for each task, whereas structural similarities between tasks at all levels are potentially relevant for multi-task learning.

3. Task Grouping Framework

We define the problem as follows: We want to minimize the overall loss on a set of tasks $\mathcal{T} = \{t_1, t_2, ..., t_k\}$ given a limited inference time budget, b, which is the total amount of time we have to complete all tasks. Each neural network that solves some subset of $\mathcal{T}$ and that could potentially be a part of the final solution is denoted by n. It has an associated inference time cost, c_n, and a loss for each task, $\mathcal{L}(n, t_i)$ (which is ∞ for each task the network does not attempt to solve). A solution $\boldsymbol{S}$ is a set of networks that together solve all tasks. The computational cost of a solution is $\text{cost}(\boldsymbol{S}) = \sum_{n \in \boldsymbol{S}} c_n$. The loss of a solution on a task, $\mathcal{L}(\boldsymbol{S}, t_i)$, is the lowest loss on that task among the solution's networks[2], $\mathcal{L}(\boldsymbol{S}, t_i) = \min_{n \in \boldsymbol{S}} \mathcal{L}(n, t_i)$. The overall performance for a solution is $\mathcal{L}(\boldsymbol{S}) = \sum_{t_i \in \mathcal{T}} \mathcal{L}(\boldsymbol{S}, t_i)$.

We want to find the solution with the lowest overall loss and a cost that is under our budget, $\boldsymbol{S}_b = \text{argmin}_{\boldsymbol{S}:\text{cost}(\boldsymbol{S}) \leq b} \mathcal{L}(\boldsymbol{S})$. Though our goal is to discover the best task grouping from scratch, for practical reasons, we start by choosing the best candidate networks from a set that we have trained a priori.

3.1. Network Training

For a given task set $\mathcal{T}$, we wish to determine not just how well each *pair* of tasks performs when trained together, but also how well each *combination* of tasks performs together so that we can capture higher-order task relationships. To that end, we've trained networks for all $2^{|\mathcal{T}|} - 1$ possible groupings: $\binom{|\mathcal{T}|}{1}$ networks with one task, $\binom{|\mathcal{T}|}{2}$ networks with two tasks, $\binom{|\mathcal{T}|}{3}$ networks with three tasks, etc. For the five tasks we use in our experiments, this is 31 networks, of which five are single-task networks.

Single-task networks are a useful baseline for comparing against a multi-task network. Multi-task learning is only useful for a set of tasks if it performs better on at least one task than a collection of single-task networks when given the same amount of total inference time. Thus, for comparison, we've also trained 25 smaller single-task networks with sizes $\frac{1}{5}$, $\frac{3}{10}$, $\frac{2}{5}$, $\frac{1}{2}$, and $\frac{4}{5}$ of the "standard" network size. Training all 56 networks required approximately 3,500 GPU hours with our setup. In Sec. 3.3 we discuss methods for reducing the training time.

We use these trained networks to analyze the relationships between tasks in Section 5. Some or all of these networks can also be used as input to our framework for finding the best-performing combination of networks in the section below.

3.2. Network Selection

Consider the situation in which we have an initial candidate set $\boldsymbol{C}_0 = \{n_1, n_2, ..., n_m\}$ of fully-trained networks that each solve some subset of our task set $\mathcal{T}$. Our goal is to choose a subset of $\boldsymbol{C}_0$ that solve all the tasks with total inference time under budget b and the lowest overall loss. More formally, we want to find a solution $\boldsymbol{S}_b = \text{argmin}_{\boldsymbol{S} \subseteq \boldsymbol{C}_0 : \text{cost}(\boldsymbol{S}) \leq b} \mathcal{L}(\boldsymbol{S})$.

It can be shown that solving this problem is NP-hard in general (reduction from SET-COVER). A brute force approach would take $O(|\mathcal{T}| \cdot |\boldsymbol{C}_0|^{\frac{b}{\min_{n \in \boldsymbol{C}_0} c_n}})$, which is exponential in the maximum number of networks that fit in our

[1] We choose 5 tasks from this set and use their data for our experiments.

[2] In principle, it may be possible to create an even better-performing ensemble when multiple networks solve the same task, though we do not explore this.

budget. This would be computationally challenging even for small problems.

However, many techniques exist that can optimally solve *most* instances of problems like these in reasonable amounts of time. All of these techniques produce solutions that perform equally well. We chose to use a branch-and-bound-like algorithm for finding this optimal solution (shown in Algorithm 1), but in principle the same solution could be achieved by other optimization methods, such as encoding the problem as a binary integer program (BIP) and solving it in a way similar to Taskonomy[44].

Algorithm 1 Get Best Networks

Input: $\boldsymbol{C}_r$, a running set of candidate networks, each with an associated cost $c \in \mathbb{R}$ and a performance score for each task the network solves. Initially, $\boldsymbol{C}_r = \boldsymbol{C}_0$
Input: $\boldsymbol{S}_r \subseteq \boldsymbol{C}_0$, a running solution, initially $\emptyset$
Input: $b_r \in \mathbb{R}$, the remaining time budget, initially b

1: **function** GETBESTNETWORKS($\boldsymbol{C}_r, \boldsymbol{S}_r, b_r$)
2: $\quad\boldsymbol{C}_r \leftarrow$ FILTER($\boldsymbol{C}_r, \boldsymbol{S}_r, b_r$)
3: $\quad\boldsymbol{C}_r \leftarrow$ SORT($\boldsymbol{C}_r$) ▷ Most promising networks first
4: $\quad\boldsymbol{Best} \leftarrow \boldsymbol{S}_r$
5: $\quad$**for** $n \in \boldsymbol{C}_r$ **do**
6: $\quad\quad\boldsymbol{C}_r \leftarrow \boldsymbol{C}_r \setminus n$ ▷ $\setminus$ is set subtraction.
7: $\quad\quad\boldsymbol{S}_i \leftarrow \boldsymbol{S}_r \cup \{n\}$
8: $\quad\quad b_i \leftarrow b_r - c_n$
9: $\quad\quad\boldsymbol{Child} \leftarrow$ GETBESTNETWORKS($\boldsymbol{C}_r, \boldsymbol{S}_i, b_i$)
10: $\quad\quad\boldsymbol{Best} \leftarrow$ BETTER($\boldsymbol{Best}, \boldsymbol{Child}$)
11: $\quad$**return** $\boldsymbol{Best}$

12: **function** FILTER($\boldsymbol{C}_r, \boldsymbol{S}_r, b_r$)
13: $\quad$Remove networks from $\boldsymbol{C}_r$ with $c_n > b_r$.
14: $\quad$Remove networks from $\boldsymbol{C}_r$ that cannot improve $\boldsymbol{S}_r$'s performance on any task.
15: $\quad$**return** $\boldsymbol{C}_r$

16: **function** BETTER($\boldsymbol{S}_1, \boldsymbol{S}_2$)
17: $\quad$**if** $C(\boldsymbol{S}_1) < C(\boldsymbol{S}_2)$ **then**
18: $\quad\quad$**return** $\boldsymbol{S}_1$
19: $\quad$**else**
20: $\quad\quad$**return** $\boldsymbol{S}_2$

Algorithm 1 chooses the best subset of networks in our collection, subject to the inference time budget constraint. The algorithm recursively explores the space of solutions and prunes branches that cannot lead to optimal solutions. The recursion terminates when the budget is exhausted, at which point $\boldsymbol{C}_r$ becomes empty and the loop body does not execute.

The sorting step on line 3 requires a heuristic upon which to sort. We found that ranking models based on how much they improve the current solution, $\boldsymbol{S}$, works well. It should be noted that this algorithm always produces an optimal solution, regardless of which sorting heuristic is used. However, better sorting heuristics reduce the running time because subsequent iterations will more readily detect and prune portions of the search space that cannot contain an optimal solution. In our setup, we tried variants of problems with 5 tasks and 36 networks, and all of them took less than a second to solve.

The definition of the BETTER() function is application-specific. For our experiments, we prefer networks that have the lowest total loss across all five tasks. Other applications may have hard performance requirements for some of the tasks, and performance on one of these tasks cannot be sacrificed in order to achieve better performance on another task. Such application-specific constraints can be encoded in BETTER().

3.3. Reducing the Training Time Burden

Conceptually, good multi-task performance for a set of tasks can be obtained by fully training a large number of candidate networks and selecting the best subset using Algorithm 1 or an equivalent selection process. Using all combinations of networks as we outlined in Section 3.1 is a good place to start. For some applications, those in which only a relatively small number of tasks are necessary, or those in which prediction accuracy is of the utmost importance, training an exponential number of networks may be acceptable. But if the number of tasks is too large or access to training hardware is too scarce, we need to find a way to reduce the training cost.

This section describes two techniques for reducing the training time required to obtain a collection of networks as input to the network selection algorithm. Our goal is to produce task groupings with results similar to the ones produced by the complete search, but with less training time burden. Both techniques involve predicting the performance of a network without actually training it to convergence. The first technique involves training each of the networks for a short amount of time, and the second involves inferring how networks trained on more than two tasks will perform based on how networks trained on two tasks perform.

3.3.1 Early Stopping Prior to Convergence

We found a moderately high correlation (Pearson's $r = 0.49$) between the validation loss of our neural networks after a pass through just 20% of our data and the final test loss of the fully trained networks. This implies that the task relationship trends stabilize early and we can get decent results by running network selection on the lightly trained networks, and then simply training the chosen networks to convergence. Of course, the exact percentage of data to use can

be chosen according to the particular setup and time-savings and accuracy goals.

For our setup, this technique reduces the training time burden by about **20x** and would require fewer than 150 GPU hours to execute. Obviously, this technique does come with a prediction accuracy penalty. Because the correlation between early network performance and final network performance is not perfect, the decisions made by network selection are no longer guaranteed to be optimal once networks are trained to convergence. We call this approximation the Early Stopping Approximation (ESA) and present the results of using this technique in Section 6.

3.3.2 Predict Higher-Order From Lower-Order

Do the performances of a network trained with tasks A and B, another trained with tasks A and C, and a third trained with tasks B and C tell us anything about the performance of a network trained on tasks A, B, and C? As it turns out, the answer is yes. Although this ignores complex task interactions and nonlinearities, a simple average of the first-order networks' accuracies was a good indicator of the accuracy of a higher-order network.

Using this strategy, we can predict the performance of all networks with three or more tasks using the performance of all of the fully trained two task networks. First, simply train all networks with two or fewer tasks to convergence. Then predict the performance of higher-order networks. Finally, run network selection on both groups.

With our setup (see Section 4), this strategy saves training time by only about 50%, compared with 95% for the early stopping approximation, and it still comes with a prediction quality penalty. However, this technique requires only a quadratic number of networks to be trained rather than an exponential number, and would therefore win out when the number of tasks is large.

We call this strategy the Higher Order Approximaiton (HOA), and present its results for our setup in Section 6.

4. Experimental Setup

4.1. Task Setup

Although our techniques are task agnostic, we have chosen to study multi-task relationships between five tasks in detail: *Semantic Segmentation, Depth Estimation, Surface Normal Prediction, Keypoint Detection*, and *Edge Detection*. The first one is a semantic task, the next two are 3D tasks, and the last two are 2D tasks. These tasks were chosen to be representative of major task categories, but also to have enough overlap in order to test the hypothesis that similar tasks will train well together. Cross-entropy loss was used for Semantic Segmentation, while an $L1$ loss was used for all other tasks.

The data for these tasks comes from the Taskonomy dataset [44]. The data was obtained from 3D scans of about 600 buildings. There are 4,076,375 examples, which we divided into 3,974,199 training instances, 52,000 validation instances, and 50,176 test instances[3]. There was no overlap in the buildings that appeared in the training and test sets. All data labels were normalized to have zero mean and unit standard deviation.

4.2. Training Details

All networks used a standard encoder-decoder architecture with a modified Xception[7] encoder. Our choice of architecture is not critical and was chosen for reasonably fast inference time performance. The Xception network encoder was simplified to have 17 layers and the middle flow layers were reduced to having 512 rather than 728 channels. Furthermore, all max-pooling layers were replaced by 2×2 convolution layers with a stride of 2 (similar to [5]). The full-sized encoder had about 4 million parameters. All networks had an input image size of 256x256. For simplicity, we measure inference time in units of the time taken to do inference for one of our full-sized encoders. We call this a standard network time (SNT). This corresponds to 2.28 billion multiply-adds and about 4 ms/image on a single Nvidia RTX 2080 Ti.

Our decoders were designed to be lightweight and have four transposed convolutional layers[29] and four separable convolutional layers[7]. Every decoder has about 116,000 parameters. All training was done using PyTorch[32] with Apex for fp16 acceleration[24].

As described in Section 3.1, we trained 31 networks with full sized encoders and standard decoders. 26 were multi-task networks and 5 were single task networks. Another five single task networks were trained, each having a half-sized encoder and a standard decoder. These 36 networks were included in network optimization as $\boldsymbol{C}_0$. 20 smaller single task networks of various sizes were also trained as a baseline, but not used for network selection (See Section 3.1). In order to produce our smaller models, we shrunk the number of channels in every layer of the encoder such that it had the appropriate number of parameters and flops.

The training loss we used was the unweighted mean of the losses for the included tasks. Networks were trained with an initial learning rate of 0.2, which was reduced by half every time the training loss stopped decreasing. Networks were trained until their validation loss stopped improving, typically requiring only 4-8 passes through the dataset. The network with the highest validation loss (checked after each epoch of 20% of our data) was saved.

The performance scores given to network selection were calculated on the validation set. We computed solutions for

[3]We found a few images that were corrupted and discarded the data from those buildings.

inference time budgets from 1 to 5 at increments of 0.5. Each solution chosen was evaluated on the test set.

4.3. Baselines

We compare our results with conventional methods, such as five single-task networks and a single network with all tasks trained jointly.

We also compare with [39] using the authors' code. We found that the algorithm under-weighted the Semantic Segmentation task too aggressively, leading to poor performance on the task and poor performance overall compared to a simple sum of task losses. We speculate that this is because SemSeg's loss is very different from the other losses. In any event, these techniques are orthogonal to ours and can be used in conjunction for situations in which they might lead to better solutions than simply summing losses.

We also provide comparason to soft parameter sharing[11] in the supplemental materials.

Finally, we compare our results to two control baselines illustrative of the importance of making good choices about which tasks to train together, 'Random' and 'Pessimal.' 'Random' is a solution consisting of valid random task groupings that solve our five tasks. The reported values are the average of a thousand random trials. 'Pessimal' is a solution in which we choose the networks that lead to the worst overall performance, though the solution's performance on each task is still the best among its networks.

Each baseline was evaluated with multiple encoder sizes so that all models' results could be compared at many inference time budgets.

5. Task Relationships

First we provide the multi-task learning relationships between our five tasks and compare those to the relationships found in Taskonomy[44].

		Relative Performance On					
		SemSeg	Depth	Normals	Keypoints	Edges	Average
Trained With	SemSeg	–	-5.41%	-11.29%	-4.32%	-34.64%	-13.92%
	Depth	4.17%	–	-3.55%	3.49%	3.76%	1.97%
	Normals	8.50%	2.48%	–	1.37%	12.33%	6.17%
	Keypoints	4.82%	1.38%	-0.02%	–	-5.26%	0.23%
	Edges	3.07%	-0.92%	-4.42%	1.37%	–	-0.23%
	Average	5.14%	-0.62%	-4.82%	0.48%	-5.95%	-1.15%

Table 1. **The first-order multi-task learning relationships between tasks.** The table lists the performance of every task when trained as a pair with every other task. For instance, when Depth is trained with SemSeg, SemSeg performs 4.17% better than when SemSeg is trained alone on a half-sized network.

We see in Table 1 that SemSeg improves no matter which of the other four tasks it is paired with. It improves the most when paired with Normals. We also see that every task improves when paired with Normals, but Edge Detection improves the most. This suggests that it can be advantageous to train tasks together even if predictions are needed for only one task. On the other hand, as the large drop in performance of Edge Detection when trained with SemSeg shows, multi-task learning with two tasks can also be deleterious and practitioners need to be careful about which tasks they combine if they hope to see any benefit.

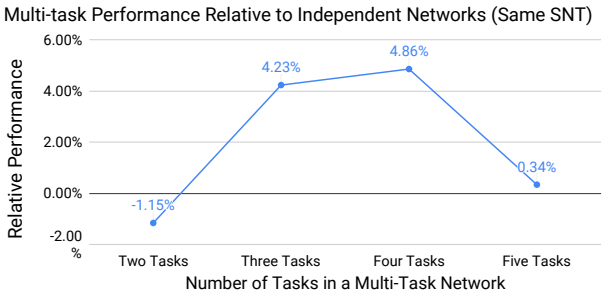

Figure 2. The average performance of multi-task networks relative to the performance of 2, 3, 4, or 5 single-task networks. Single-task networks are $\frac{1}{2}$, $\frac{1}{3}$, $\frac{1}{4}$, or $\frac{1}{5}$-sized so that comparisons are between solutions with a total evaluation time of 1 SNT.

Figure 2 shows that networks smaller than $\frac{1}{2}$ SNT seem to be starved of resources, but multi-task learning effectively shares these resources. This resource sharing is only beneficial when resources are scarce. When there are more resources (in our case, when the total size of all networks is above 1.5 SNT), single-task networks outperform multi-task networks that solve all five tasks.

Section 3.3.2 suggests predicting higher-order results using an average of first-order results. This strategy has an average max ratio error of only 5.2%, and overestimates performance by about 1.4% on average. The results in Section 6 show that this was enough for network selection to do a reasonably good job of choosing higher-order networks for training to convergence.

	Depth	Normals	Keypoints	Edges
SemSeg	-0.62%	-1.39%	0.25%	-15.78%
Depth		-0.54%	2.43%	1.42%
Normals			0.67%	3.95%
Keypoints				-1.95%

Table 2. **The first-order multi-task learning affinity between tasks.** This is the mean of Table 1 and its transpose. These values show the average change in the performance of two tasks when trained as a pair, relative to when they are trained separately using half-sized networks.

	Depth	Normals	Keypoints	Edges
SemSeg	1.740%	1.828%	0.723%	0.700%
Depth		1.915%	0.406%	0.468%
Normals			0.089%	0.118%
Keypoints				0.232%

Table 3. The averaged forward and backward transfer learning affinity between tasks from Taskonomy[44].

5.1. Multi-Task vs Transfer Relationships

In order to determine the multi-task affinity between tasks, we took the average of our first-order relationships

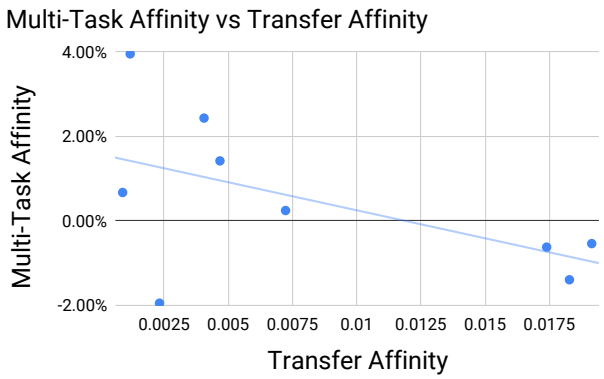

Figure 3. Task affinities for multi-task learning vs. transfer learning. The correlation (Pearson's r) is -0.54, $p = 0.13$. One outlier is removed.

matrix (Table 1) and its transpose. The result is shown in Table 2. The tasks with the highest affinity by this metric are Surface Normal Prediction and 2D Edge Detection. Our two 3D tasks, Depth Estimation and Surface Normal Prediction, do not score highly on this similarity metric, unlike for Taskonomy (Table 3), where they have the highest affinity. Our two 2D tasks also do not score highly. We speculate that the Normals task naturally preserves edges, while Depth and Normals (for example) don't add much training signal to each other.

Figure 3 depicts the relationship between first-order transfer learning affinities and first-order multi-task affinities, which surprisingly seem to be negatively correlated. This suggests that it is better to train dissimilar tasks together. This could be because dissimilar tasks are able to provide stronger and more meaningful regularization. More research is necessary to discover if this correlation and explanation hold in general.

6. Task Grouping Evaluation

Time Budget	1	1.5	2	2.5	3	3.5	4	4.5	5
Sener et al.	0.562		0.556	0.551			0.547		
Pessimal Grouping	0.503	0.503	0.503	0.503	0.503	0.502	0.499	0.496	0.495
Traditional MTL	0.503		0.492	0.487			0.488		
Random Groupings	0.503	0.483	0.475	0.471	0.467	0.464	0.462	0.460	0.459
Independent	0.515	0.501	0.477	0.465			0.454		0.448
Ours (ESA) 3.3.1	0.503	0.487	0.467	0.461	0.457	0.451	0.451	0.447	0.447
Ours (HOA) 3.3.2	0.503	0.461	0.455	0.451	0.449	0.445	0.444	0.445	0.442
Ours Optimal	0.503	0.461	**0.452**	**0.446**	**0.442**	**0.439**	**0.436**	**0.436**	**0.435**

Table 4. The total test set loss on all five tasks for each method under each inference time budget. Lower is better. The data is the same as in Figures 4 and 5.

Figure 5 shows the task groups that were chosen for each technique, and Figure 4 shows the performance of these groupings along with those of our baselines, with the tabular data provided in Table 4. We can see that each of our methods outperforms our traditional baselines for every computational budget.

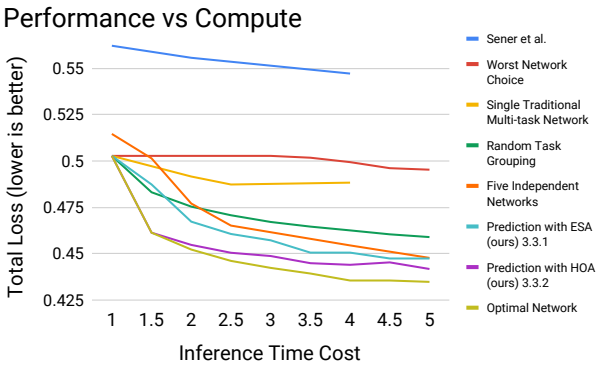

Figure 4. The performance/inference time trade-off for various methods. Raw data in Table 4.

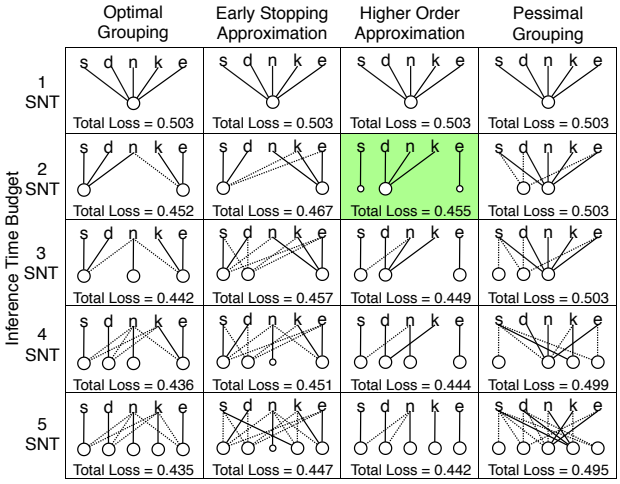

Figure 5. **The task groups picked by each of our techniques for integer budgets between 1 and 5.** Networks are shown as ○ (full-sized) or ○ (half sized). Networks are connected to the tasks for which they compute predictions. *s: Semantic Segmentation, d: Depth Estimation, n: Surface Normal Prediction, k: Keypoint Detection, e: Edge Detection*. Gray edges represent unused decoders. For example, the highlighted solution consists of two half-sized networks and a full-sized network. The full-sized network solves Depth Estimation, Surface Normal Prediction, and Keypoint Detection. One half-sized network solves Semantic Segmentation and the other solves Edge Detection. The total loss for all five tasks is 0.455. The groupings for fractional budgets are shown in the Supplemental Material.

When the computational budget is only 1 SNT, all of our methods select the same model—a traditional multi-task network with a single unit-sized encoder and five decoders. This model performs better than both a similar model trained with the algorithm in [39] and our baseline model with five encoders, each taking $\frac{1}{5}$ of an SNT. When the budget is 1.5 SNT, our optimal method chooses a half-sized network for SemSeg with a full-sized network for the remaining four tasks. The Higher Order Approximation chooses the same two networks, while the Early Stopping approximation chooses a half-sized network for Edges and a full-sized network for the remaining four tasks. This choice results in inferior performance compared to our other meth-

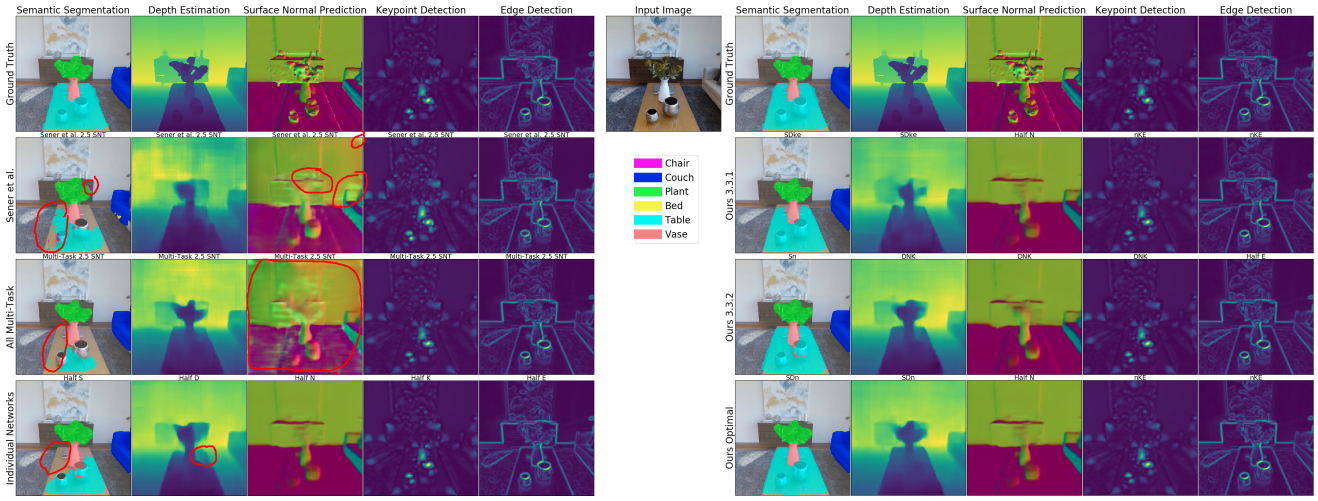

Figure 6. Qualitative results for our techniques (right) and our baselines (left). All solutions have an inference time cost of 2.5 SNT.

ods, but still better than all methods with a budget of only 1 SNT, as well as all baselines except 'Random.' For higher inference time budgets, our optimal method chooses better networks than our approximate methods, but all of our methods have superior performance compared to the traditional multi-task learning baseline, as well as the method in [39].

As shown in Figure 5, the Early Stopping Approximation selects a network that solves the four tasks s, d, k, and e for solutions with more than 2.5 SNT. This network does not perform well enough to be the best to use for any task, though this was not discovered by the algorithm until it was fully trained. In this case, that network performed abnormally well after one pass through 20% of the data. In cases such as these, the network could be removed from $\boldsymbol{C}_0$ and network selection could be run again to improve the results.

When the computational budget is effectively unlimited (5 SNT), our optimal method picks five networks, each of which is used to make predictions for a separate task. However, three of the networks are trained with three tasks, while two are each trained with one task. This shows that the representations learned through multi-task learning were found to be best for three of our tasks (s, d, and e), whereas two of our tasks (n and k) are best solved individually.

We also see that our optimal technique can perform better at 2.5 SNT than five individual networks can perform using 5 SNT total. Our Higher-Order Approximation can do better than 5 individual full-sized networks using only 3.5 SNT.

Figure 6 allows qualitative comparison between our methods (architecture shown in Figure 1) and our baselines. We can see clear visual issues (circled) with each of our baselines that are not present in our methods. Both of our approximate methods produces similar predictions to the optimal task grouping.

7. Conclusion

We describe the problem of task compatibility as it pertains to multi-task learning. We analyze how tasks interact in a multi-task setting and compare that with previous results on transfer learning task interactions. We provide an algorithm and framework for determining which tasks should be trained jointly and which tasks should trained separately. Our solution can take advantage of situations in which multi-task learning is beneficial to one task, but not the other. For many use cases, this framework is sufficient, but it can be costly at training time. Hence, we offer two strategies for coping with this issue and evaluate their performance. Our methods outperform single-task networks, a multi-task network with all tasks trained jointly, as well as other baselines.

Acknowledgements We gratefully acknowledge the support of TRI, NSF grant DMS-1546206, a Vannevar Bush Faculty Fellowship, and ONR MURI (N00014-14-1-0671). Toyota Research Institute ("TRI") provided funds to assist the authors with their research but this article solely reflects the opinions and conclusions of its authors and not TRI or any other Toyota entity.

References

[1] A. Achille, M. Lam, R. Tewari, A. Ravichandran, S. Maji, C. Fowlkes, S. Soatto, and P. Perona. Task2Vec: Task Embedding for Meta-Learning. *arXiv e-prints*, page arXiv:1902.03545, Feb. 2019. 3

[2] H. Bilen and A. Vedaldi. Integrated perception with recurrent multi-task neural networks. In D. D. Lee, M. Sugiyama, U. V. Luxburg, I. Guyon, and R. Garnett, editors, *Advances*

in Neural Information Processing Systems 29, pages 235–243. Curran Associates, Inc., 2016. 2

[3] J. Bingel and A. Søgaard. Identifying beneficial task relations for multi-task learning in deep neural networks. In *Proceedings of the 15th Conference of the European Chapter of the Association for Computational Linguistics: Volume 2, Short Papers*, pages 164–169, Valencia, Spain, Apr. 2017. Association for Computational Linguistics. 2

[4] R. Caruana. Multitask learning. *Machine Learning*, 28(1):41–75, Jul 1997. 2

[5] L. Chen, Y. Zhu, G. Papandreou, F. Schroff, and H. Adam. Encoder-decoder with atrous separable convolution for semantic image segmentation. In *Computer Vision - ECCV 2018 - 15th European Conference, Munich, Germany, September 8-14, 2018, Proceedings, Part VII*, pages 833–851, 2018. 5

[6] Z. Chen, V. Badrinarayanan, C. Lee, and A. Rabinovich. Gradnorm: Gradient normalization for adaptive loss balancing in deep multitask networks. In *Proceedings of the 35th International Conference on Machine Learning, ICML 2018, Stockholmsmässan, Stockholm, Sweden, July 10-15, 2018*, pages 793–802, 2018. 2

[7] F. Chollet. Xception: Deep learning with depthwise separable convolutions. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 1251–1258, 2017. 5

[8] P. B. C. d. Miranda, R. B. C. Prudłncio, A. C. P. L. F. d. Carvalho, and C. Soares. Combining a multi-objective optimization approach with meta-learning for svm parameter selection. In *2012 IEEE International Conference on Systems, Man, and Cybernetics (SMC)*, pages 2909–2914, Oct 2012. 2

[9] J. Dai, K. He, and J. Sun. Instance-aware semantic segmentation via multi-task network cascades. *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 3150–3158, 2016. 2

[10] C. Doersch and A. Zisserman. Multi-task self-supervised visual learning. In *International Conference on Computer Vision*, 2017. 2

[11] L. Duong, T. Cohn, S. Bird, and P. Cook. Low resource dependency parsing: Cross-lingual parameter sharing in a neural network parser. In *Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing (Volume 2: Short Papers)*, pages 845–850, July 2015. 2, 6

[12] N. Dvornik, K. Shmelkov, J. Mairal, and C. Schmid. BlitzNet: A real-time deep network for scene understanding. In *IEEE International Conference on Computer Vision (ICCV)*, 2017. 2

[13] K. Dwivedi and G. Roig. Representation similarity analysis for efficient task taxonomy and transfer learning. In *CVPR*. IEEE Computer Society, 2019. 3

[14] C. Fernando, D. Banarse, C. Blundell, Y. Zwols, D. Ha, A. A. Rusu, A. Pritzel, and D. Wierstra. Pathnet: Evolution channels gradient descent in super neural networks. *CoRR*, abs/1701.08734, 2017. 3

[15] C. Finn, X. Y. Tan, Y. Duan, T. Darrell, S. Levine, and P. Abbeel. Deep spatial autoencoders for visuomotor learning. In *Robotics and Automation (ICRA), 2016 IEEE International Conference on*, pages 512–519. IEEE, 2016. 2

[16] T. Helleputte and P. Dupont. Feature selection by transfer learning with linear regularized models. In W. Buntine, M. Grobelnik, D. Mladenić, and J. Shawe-Taylor, editors, *Machine Learning and Knowledge Discovery in Databases*, pages 533–547, Berlin, Heidelberg, 2009. Springer Berlin Heidelberg. 2

[17] 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 (CVPR)*, 2018. 2

[18] I. Kokkinos. Ubernet: Training a universal convolutional neural network for low-, mid-, and high-level vision using diverse datasets and limited memory. In *2017 IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2017, Honolulu, HI, USA, July 21-26, 2017*, pages 5454–5463, 2017. 2

[19] M. Long, Z. CAO, J. Wang, and P. S. Yu. Learning multiple tasks with multilinear relationship networks. In I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, editors, *Advances in Neural Information Processing Systems 30*, pages 1594–1603. Curran Associates, Inc., 2017. 2

[20] Y. Lu, A. Kumar, S. Zhai, Y. Cheng, T. Javidi, and R. S. Feris. Fully-adaptive feature sharing in multi-task networks with applications in person attribute classification. In *CVPR*, pages 1131–1140. IEEE Computer Society, 2017. 2

[21] Z. Luo, Y. Zou, J. Hoffman, and L. F. Fei-Fei. Label efficient learning of transferable representations acrosss domains and tasks. In *Advances in Neural Information Processing Systems*, pages 164–176, 2017. 2

[22] A. Mallya and S. Lazebnik. Piggyback: Adding multiple tasks to a single, fixed network by learning to mask. *CoRR*, abs/1801.06519, 2018. 3

[23] Q. Mercier, F. Poirion, and J.-A. Dsidri. A stochastic multiple gradient descent algorithm. *European Journal of Operational Research*, 271(3):808–817, 2018. 2

[24] P. Micikevicius, S. Narang, J. Alben, G. Diamos, E. Elsen, D. Garcia, B. Ginsburg, M. Houston, O. Kuchaiev, G. Venkatesh, et al. Mixed precision training. *arXiv preprint arXiv:1710.03740*, 2017. 5

[25] L. Mihalkova, T. Huynh, and R. J. Mooney. Mapping and revising markov logic networks for transfer learning. In *AAAI*, volume 7, pages 608–614, 2007. 2

[26] I. Misra, A. Shrivastava, A. Gupta, and M. Hebert. Cross-stitch networks for multi-task learning. *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 3994–4003, 2016. 2

[27] V. Nekrasov, T. Dharmasiri, A. Spek, T. Drummond, C. Shen, and I. D. Reid. Real-time joint semantic segmentation and depth estimation using asymmetric annotations. *CoRR*, abs/1809.04766, 2018. 2

[28] A. Niculescu-Mizil and R. Caruana. Inductive transfer for bayesian network structure learning. In *Artificial Intelligence and Statistics*, pages 339–346, 2007. 2

[29] H. Noh, S. Hong, and B. Han. Learning deconvolution network for semantic segmentation. In *2015 IEEE International Conference on Computer Vision (ICCV)*, pages 1520–1528, Dec 2015. 5

[30] A. Pal and V. N. Balasubramanian. Zero-shot task transfer, 2019. 3

[31] S. J. Pan and Q. Yang. A survey on transfer learning. *IEEE Transactions on Knowledge and Data Engineering*, 22(10):1345–1359, Oct 2010. 2

[32] A. Paszke, S. Gross, S. Chintala, G. Chanan, E. Yang, Z. DeVito, Z. Lin, A. Desmaison, L. Antiga, and A. Lerer. Automatic differentiation in pytorch. In *NIPS-W*, 2017. 5

[33] A. Pentina and C. H. Lampert. Multi-task learning with labeled and unlabeled tasks. In D. Precup and Y. W. Teh, editors, *Proceedings of the 34th International Conference on Machine Learning*, volume 70 of *Proceedings of Machine Learning Research*, pages 2807–2816, International Convention Centre, Sydney, Australia, 06–11 Aug 2017. PMLR. 2

[34] L. Y. Pratt. Discriminability-based transfer between neural networks. In S. J. Hanson, J. D. Cowan, and C. L. Giles, editors, *Advances in Neural Information Processing Systems 5*, pages 204–211. Morgan-Kaufmann, 1993. 2

[35] A. S. Razavian, H. Azizpour, J. Sullivan, and S. Carlsson. Cnn features off-the-shelf: An astounding baseline for recognition. In *Proceedings of the 2014 IEEE Conference on Computer Vision and Pattern Recognition Workshops*, CVPRW '14, pages 512–519, Washington, DC, USA, 2014. IEEE Computer Society. 2

[36] E. M. Rudd, M. Günther, and T. E. Boult. MOON: A mixed objective optimization network for the recognition of facial attributes. In *ECCV (5)*, volume 9909 of *Lecture Notes in Computer Science*, pages 19–35. Springer, 2016. 2

[37] S. Ruder. An overview of multi-task learning in deep neural networks. *CoRR*, abs/1706.05098, 2017. 2

[38] A. A. Rusu, N. C. Rabinowitz, G. Desjardins, H. Soyer, J. Kirkpatrick, K. Kavukcuoglu, R. Pascanu, and R. Hadsell. Progressive neural networks. *CoRR*, abs/1606.04671, 2016. 3

[39] O. Sener and V. Koltun. Multi-task learning as multi-objective optimization. In S. Bengio, H. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi, and R. Garnett, editors, *Advances in Neural Information Processing Systems 31*, pages 525–536. Curran Associates, Inc., 2018. 2, 6, 7, 8

[40] D. L. Silver and K. P. Bennett. Guest editors introduction: special issue on inductive transfer learning. *Machine Learning*, 73(3):215–220, 2008. 2

[41] C. Tessler, S. Givony, T. Zahavy, D. J. Mankowitz, and S. Mannor. A deep hierarchical approach to lifelong learning in minecraft. In *Proceedings of the Thirty-First AAAI Conference on Artificial Intelligence*, AAAI'17, pages 1553–1561. AAAI Press, 2017. 2

[42] S. Thrun. Is learning the n-th thing any easier than learning the first? In *Advances in Neural Information Processing Systems*, pages 640–646. The MIT Press, 1996. 2

[43] Y. Yang and T. Hospedales. Trace norm regularised deep multi-task learning. In *5th International Conference on Learning Representations Workshop*, 2017. 2

[44] A. R. Zamir, A. Sax, W. B. Shen, L. J. Guibas, J. Malik, and S. Savarese. Taskonomy: Disentangling task transfer learning. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. IEEE, 2018. 1, 2, 3, 4, 5, 6

[45] A. R. Zamir, T. Wekel, P. Agrawal, C. Wei, J. Malik, and S. Savarese. Generic 3d representation via pose estimation and matching. In B. Leibe, J. Matas, N. Sebe, and M. Welling, editors, *Computer Vision – ECCV 2016*, pages 535–553, Cham, 2016. Springer International Publishing. 2

[46] Y. Zhang and Q. Yang. A survey on multi-task learning. *CoRR*, abs/1707.08114, 2017. 2

[47] D. Zhou, J. Wang, B. Jiang, H. Guo, and Y. Li. Multi-task multi-view learning based on cooperative multi-objective optimization. *IEEE Access*, 6:19465–19477, 2018. 2