

# Dynamic Task Prioritization for Multitask Learning

Michelle Guo<sup>[0000-0002-6574-6669]</sup>, Albert Haque<sup>[0000-0001-6769-6370]</sup>, De-An Huang<sup>[0000-0002-6945-7768]</sup>, Serena Yeung<sup>[0000-0003-0529-0628]</sup>, and Li Fei-Fei<sup>[0000-0002-7481-0810]</sup>

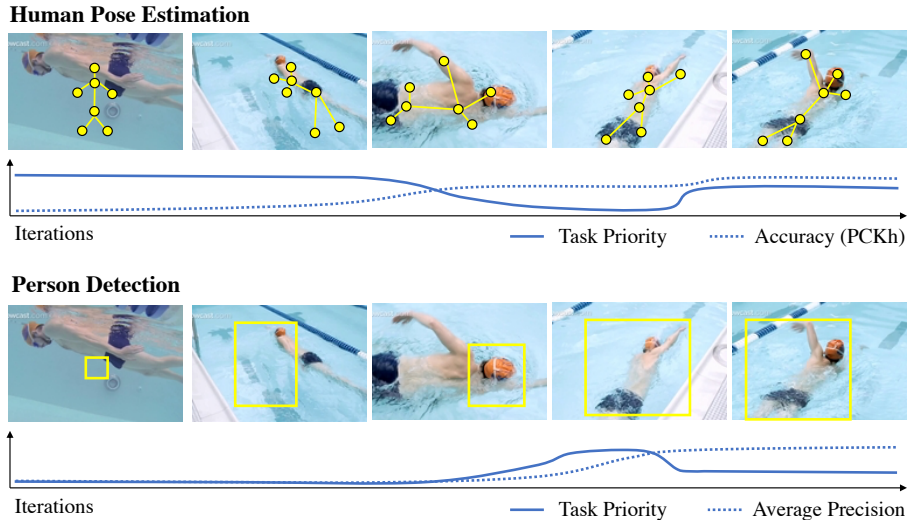
Department of Computer Science, Stanford University, USA

**Abstract.** We propose *dynamic task prioritization* for multitask learning. This allows a model to dynamically prioritize difficult tasks during training, where difficulty is inversely proportional to performance, and where difficulty changes over time. In contrast to curriculum learning, where easy tasks are prioritized above difficult tasks, we present several studies showing the importance of prioritizing difficult tasks first. We observe that imbalances in task difficulty can lead to unnecessary emphasis on easier tasks, thus neglecting and slowing progress on difficult tasks. Motivated by this finding, we introduce a notion of *dynamic task prioritization* to automatically prioritize more difficult tasks by adaptively adjusting the mixing weight of each task’s loss objective. Additional ablation studies show the impact of the task hierarchy, or the task ordering, when explicitly encoded in the network architecture. Our method outperforms existing multitask methods and demonstrates competitive results with modern single-task models on the COCO and MPII datasets.

## 1 Introduction

Children can efficiently manage multiple subjects in school. This multitasking capability is generally possible because one spends more time and effort on the subjects they find more challenging, rather than the subjects they find easy [1]. By allocating mental resources proportional to the complexity and difficulty of each subject, humans can increase the effectiveness and efficiency at which learning occurs [2, 3]. This idea is supported by the task management and cognitive workload literature [4, 5].

Like humans, computational models can also perform multitask learning by jointly training on multiple tasks. Multitask learning is prevalent in several applications, including computer vision [6–8], natural language processing [9–13], speech processing [14–16], and reinforcement learning [17–20]. Some works [21] train a single model across multiple input domain modalities. However, when multiple tasks are presented to a model, it is possible for easy tasks to dominate learning, while progress is stunted on harder ones. We pose the following question: As we train a multitask model, should we adjust the amount of learning from easy versus difficult tasks?



**Fig. 1: Dynamic task prioritization.** Example of a single model trained on two simultaneous tasks: (top) pose estimation and (bottom) person detection. For each task: (Images) Input images with corresponding task-specific labels. (Line Plot) Dynamic task priority and performance over time. The  $x$  axis denotes training iteration number;  $y$  axis denotes task priority and model performance.

A key challenge towards machine multitasking is *task prioritization*: deciding which resources to allocate to which tasks. These resources can take the form of gradient magnitudes, parameter count, or update frequencies. Task prioritization is especially challenging when tasks vary in their degrees of difficulty. In traditional multitask learning [22], a model continues to invest the same level of detail on easy tasks, even after mastering them. Perfecting these simple tasks wastes valuable resources. As a result, challenging tasks, which may require additional learning, learn less quickly and perform poorly, compared to easier tasks.

Curriculum learning attempts to treat easy and hard tasks differently by learning easy tasks before harder ones [23]. Defined by Bengio et al. [24], curriculum learning divides a single task into simpler subtasks which are presented to a model in increasing difficulty. A critical assumption of curriculum learning is that the underlying distribution across all tasks is the same but the entropy increases over time [24]. However, this assumption is broken when defining the multitask problem over disparate tasks (i.e., tasks do not share the same distribution, such as pose estimation versus classification). Since curriculum learning holds this assumption, conclusions from curriculum learning cannot be applied in the general, and arguably more common, multitask setting where tasks are not subsets of a single task.

**Contributions.** In this paper, we propose *dynamic task prioritization* for multitask learning. Inspired by human learning [1–3], our model is encouraged

to prioritize difficult tasks and examples. We liken this to the problem of class imbalance, which is commonly remedied by hard negative mining [25, 26]. Our contributions are two-fold:

1. We present a comprehensive analysis to better understand the task prioritization problem at both an example-level and task-level. The results of our analysis indicate that more learning resources should be allocated to difficult tasks rather than easier tasks.
2. We propose a unified framework that operationalizes the above insight: Our method dynamically adjusts task-level loss coefficients to continually prioritize difficult tasks. This uses learning progress signals to automatically compute a time-varying distribution of task weights.

Empirically, we evaluate our method on classification, segmentation, detection, and pose estimation using the COCO [27] and MPII Human Pose datasets [28].

## 2 Related Work

Our work on multitask learning is related to curriculum learning, which was proposed by Elman [29] to improve the training of multiple task subsets with a constant underlying distribution, starting with smaller and simpler tasks first. This has been demonstrated in many works [30, 31, 23]. For example, in [32], Zaremba and Sutskever propose two criteria for self-pacing through the curriculum. However, once learning occurs from diverse tasks (i.e., data or labels from different distributions), as in our setting, the assumptions of curriculum learning no longer hold [24] and it can be difficult for these pre-selected progress criteria to continue to hold. In our case, the underlying distribution across tasks can be significantly different (e.g., domain adaptation [33–35]).

To address diverse tasks, there are two approaches: (i) assign different priorities to tasks by using task-level weights or (ii) structure the network architecture to take advantage of inter-task relationships, as is common in task hierarchies.

### 2.1 Task Weighting

**Task Weighting.** Multitask learning models are sensitive to task weights [36]. A task weight is commonly defined as the mixing or scaling coefficient used to combine multiple loss objectives. Task weights are typically selected through extensive hyperparameter tuning (e.g., UberNet [37], Overfeat [38]). Additionally, task weights are often static throughout the course of training, potentially diverting training resources to unnecessary tasks or examples [39]. In [36], the authors automatically derive the weights based on the uncertainty of each task, but they do not consider task difficulty. Recent methods attempt to dynamically adjust or normalize the task weights according to prescribed criteria or normalization requirements, such as GradNorm [40]. These dynamic techniques are sometimes referred to as self-paced learning methods.

**Self-Paced Learning.** Self-paced learning [41] is an automated approach to curriculum learning where the curriculum is determined by the model’s abilities rather than being fixed via external human supervision [42]. In [43], the authors proposed automatically selecting task-specific loss weights via a regularizer on task weights. However, the tasks were subsets of a larger task and thus do not represent a diverse set of tasks. In [44], the authors alternate between learning the task-ordering and instance-level ordering. It is similar to our work but assumes the task-specific model can be trained in a single iteration (i.e., no gradient descent), so its effectiveness for deeper neural networks is unclear. We believe automatic weighting is the correct research direction, but task weights must be selected to better suit the multitask setting.

**Learning From Progress Signals.** In [31], Graves et al. use an accuracy metric as a *learning progress* signal to find a stochastic policy for task curriculum learning [45]. This learning progress signal is used to actively select the syllabus through a curriculum such that it maximizes overall progress. Learning from progress signals is commonplace in reinforcement learning tasks, serving as indicators of reward signals to encourage exploration [46–49]. Routing Networks [50] takes a multi-agent approach to dynamically select different network submodules, depending on the task and rewards. Neural architecture search [51] takes this a step further and trains an agent with the goal of designing entire network architectures, using accuracy as the progress (reward) signal. In this work, we use a variant of prediction gain [52], reformulated for supervised learning tasks, to dynamically compute task weights/priority during training.

## 2.2 Inter-Task Relationships

In this work, we jointly predict classification, person segmentation, person detection, and human pose labels. These tasks are important for understanding humans in images. Mask R-CNN [53] is a popular method which is capable of predicting segmentation, detection, and human pose labels. Our work differs in that we predict all tasks simultaneously by leveraging inter-task difficulty levels.

**Hard Parameter Sharing.** Hard parameter sharing shares the hidden layers across all tasks, but maintains separate task-specific output modules (e.g., a single fully-connected layer before the loss). It is one of the most commonly used approaches for multitask learning. The motivation is that one can improve generalization by using domain information contained in related tasks [22]. Hard parameter sharing has been successful in image classification [54], object detection [55, 39], semantic segmentation [53], and facial analysis [56]. In [57], the authors use hard sharing with sequence-to-sequence models. Compared to a single model per task, hard parameter sharing can reduce the risk of overfitting [58] occasionally leading to performance improvements [37, 59].

However, hard parameter sharing has two major drawbacks. First, task-specific loss objectives must be combined, requiring task-specific weights. Selecting these weights can be difficult and expensive [60]. Second, at some point in the network architecture, hard sharing methods use a single shared representation which is then fed into multiple task submodules [39, 61, 53, 62, 8, 57]. This

leads to a *critical layer*: a layer responsible for learning representations that must satisfy all downstream objectives. The burden on this layer can make it difficult to optimize [22].

**Task Hierarchy.** Multitask learning benefits from multiple related tasks [63, 64] as they can reinforce one another and improve overall performance [23, 65]. One method of exploiting inter-task relationships is to formulate a task hierarchy [66]. In these hierarchical multitask models, increasingly complex tasks are predicted at successively deeper layers. This has yielded promising results in the natural language processing community [10]. In the work by Søgaard and Goldberg [67], they developed a model with part-of-speech tags supervised at lower layers while higher-level language tasks such as language inference [68] and machine translation [69] were supervised at later layers. Feedback Networks [70] show the efficacy of learning an implicit hierarchy by learning a different function at different depths of a network unrolled in time. While task hierarchy is not our primary contribution in this paper, we examine the applicability of an explicit task hierarchy embedded in the network architecture. We arrange multiple computer vision tasks in a hierarchy, ordered by difficulty.

### 3 Method

We introduce *dynamic task prioritization* for multitask learning. In contrast to the self-paced multitask loss proposed in [43], which assigns more weight to easier tasks, our method prioritizes difficult tasks instead. Also different from [43], our method does not use task losses to determine relative task difficulties. Instead, we use more intuitive and realistic metrics for dynamically prioritizing tasks: progress signals – also known as key performance metrics (KPIs). This is an idea commonly explored in reinforcement learning literature [52, 31], which we adapt for the multitask setting.

#### 3.1 Priority Based on Difficulty

In this subsection, we define the notion of priority and discuss how we dynamically adjust it, based on difficulty. There are two use cases: (i) example-level priority and (ii) task-level priority.

**Preliminaries.** We define our algorithm over an ordered set of tasks  $T = \{T_1, \dots, T_{|T|}\}$ . We define difficulty  $\mathcal{D} \propto \kappa^{-1}$  where  $\kappa$  is a performance metric such as accuracy. Let  $t$  denote the current task index being considered from the task set in  $T$ . Tasks  $T_1, \dots, T_{|T|}$  are ordered according to their difficulty  $\mathcal{D}(T_t)$ . Without loss of generality,  $\forall t \in |T|$  we have  $\mathcal{D}(T_t) \geq \mathcal{D}(T_{t+1})$ .

The task-specific loss (e.g., cross-entropy) for task  $T_t$  is denoted by  $L_t(\cdot)$ . Since some examples may not contain ground truth labels for all possible tasks in  $T$ , we use  $\delta_{t,i} \in \{0, 1\}$  to denote the availability of ground truth data for example  $i$ , task  $T_t$ . Then the masked task loss  $\mathcal{L}_t(\cdot)$  is defined in (0), where  $i$  is the index of the training example,  $p_t^i$  is the model’s post-softmax output for

example  $i$  for task  $T_t$ , and  $y_t^i$  is the ground truth for example  $i$  for task  $T_t$ .

$$\mathcal{L}_t(\cdot) = \frac{1}{N} \sum_{i=1}^N \delta_{t,i} L_t(p_t^i, y_t^i) \quad (0) \quad \mathcal{L}_{\text{Total}} = \sum_{t=1}^{|T|} \lambda_t \mathcal{L}_t \quad (1)$$

In the standard multitask learning setup, multiple losses are combined using mixing parameters  $\lambda_t$  as shown in (1). Intuitively,  $\lambda_t$  denotes the task weight (i.e., relative importance/scaling).

**Key Performance Indicators.** For each task  $T_t$ , we select a key performance indicator (KPI) denoted by  $\kappa_t \in [0, 1]$ . The KPI  $\kappa_t$  should be a meaningful metric such as accuracy or average precision (AP), including for regression tasks (e.g., where success is defined by some error threshold). We compute  $\kappa_t$  to be an exponential moving average  $\bar{\kappa}_t^{(\tau)} = \alpha \kappa_t^{(\tau)} + (1 - \alpha) \bar{\kappa}_t^{(\tau-1)}$  where  $\tau$  is the training iteration number and  $\alpha \in [0, 1]$  is the discount factor. Larger values of  $\alpha$  prioritize more recent examples. We discuss later that  $\kappa_t$  need not be differentiable.

Finally, let  $\gamma_0 \geq 0$  denote the example-level focusing parameter and  $\gamma_1, \dots, \gamma_t \geq 0$  denote the task-level focusing parameters. These focusing parameters  $\gamma_0, \dots, \gamma_t$  are *not* the actual weights applied to the loss (i.e., not the mixing parameters) but rather adjust the *rate* at which easy examples and tasks are down-weighted.

**Example-Level Prioritization.** We now describe how difficult examples are identified. Consider binary classification with cross entropy (CE):

$$\text{CE}(p_c) = -\log(p_c) \quad \text{where} \quad p_c = \begin{cases} p, & \text{if } y = 1 \\ 1 - p, & \text{otherwise} \end{cases} \quad (2)$$

where  $y \in \{-1, +1\}$  denotes the true class label and  $p \in [0, 1]$  is the model’s post-softmax output (i.e., probability) for the class  $y = 1$ . One notable property of CE is that easily classified examples will have  $p_c \gg 0.5$ .

In [39], the authors proposed the Focal Loss as a way to down-weight easier examples and focus on harder examples during training. It is defined as:

$$\text{FL}(p_c; \gamma_0) = -(1 - p_c)^{\gamma_0} \log(p_c) \quad (3)$$

where  $\gamma_0$  is the example-level focusing parameter, as defined above. While  $\text{FL}(\cdot)$  is defined for classification, we can extend this to regression tasks. Consider a real-valued error metric  $e_i$  for some example  $i$ . We can use  $\text{FL}(e_i; \gamma_0)$  if  $e_i \in [0, 1]$ . One normalization scheme is to scale  $e_i$  by a constant such as the image size.

We define the task-specific loss function as  $\mathcal{L}_t^*(\cdot) = \text{FL}(p_c; \gamma_0)$ , where each example is weighted by its difficulty. The loss  $\mathcal{L}_t^*(\cdot)$  effectively scales the example-level weight because difficult examples now contribute more to the overall loss. As a result they are given more “weight” during backpropagation. This is in line with our overall motivation: we wish to dynamically adjust the training procedure such that learning resources are not constantly allocated to easy examples.

**Task-Level Prioritization.** Similar to example-level prioritization, if the KPI  $\bar{\kappa}_t \gg 0.5$ , we can assume that task  $T_t$  is easy for the model. If accuracy or precision on a given task may be 99%, this should be taken into consideration

when combining the loss with a more difficult task. To balance easy and difficult tasks, we propose to scale each task-specific loss  $\mathcal{L}_t^*(\cdot)$  by computing the task difficulty  $\mathcal{D}(T_t) = \text{FL}(\bar{\kappa}_t; \gamma_t)$ . Our *dynamic task prioritization* loss ( $\mathcal{L}_{\text{DTP}}$ ) is:

$$\mathcal{L}_{\text{DTP}}(\cdot) = \mathcal{L}_{\text{Total}}^*(\cdot) = \sum_{t=1}^{|T|} \text{FL}(\bar{\kappa}_t; \gamma_t) \mathcal{L}_t^*(\cdot) \quad (4)$$

To summarize thus far, our loss  $\mathcal{L}_{\text{DTP}}$  uses learning progress signals (i.e.,  $\hat{\kappa}_t$ ) to automatically compute a priority level at both a task-level and example-level. These priority levels vary throughout the training procedure.

**Gradients.** In the case where the KPI  $\kappa_t$  is differentiable, such as the intersection-over-union loss layer [71] or KPI approximations [72], the gradient can be computed as normal. In the case where the KPI  $\kappa_t$  may not be differentiable, the derivative of  $\mathcal{L}_{\text{DTP}}(\cdot)$  with respect to  $x$  is:

$$\frac{\partial}{\partial x} \mathcal{L}_{\text{DTP}}(\cdot) = \sum_{t=1}^{|T|} \left[ \frac{\partial}{\partial x} \mathcal{L}_t^*(\cdot) \right] \text{FL}(\bar{\kappa}_t; \gamma_t) + \left[ \frac{\partial}{\partial x} \text{FL}(\bar{\kappa}_t; \gamma_t) \right] \mathcal{L}_t^*(\cdot) \quad (5)$$

Treating  $\text{FL}(\kappa_t; \gamma_t)$  as a constant causes the second term to evaluate to zero. As a result,  $\mathcal{L}_{\text{DTP}}(\cdot)$  reduces to the standard multitask learning loss with per-task weights as shown in Eq. 1. The final  $\mathcal{L}_{\text{DTP}}(\cdot)$  can be minimized with first-order optimization methods [73].

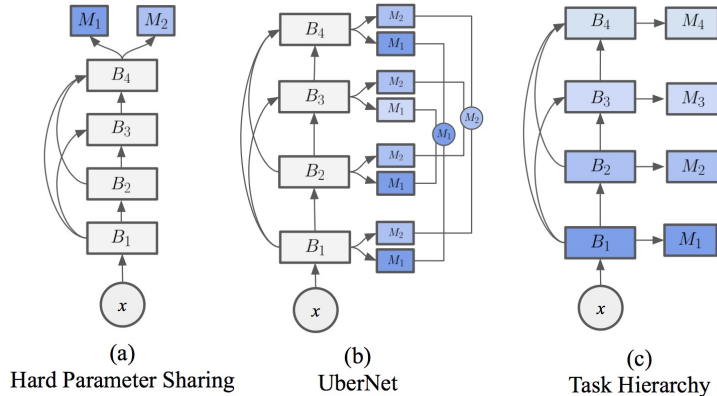
### 3.2 Implicit Priority from the Network Architecture

The central theme of this work is to prioritize learning from difficult examples and tasks, where difficulty is measured by some progress signal. Our proposed loss in §3.1 handles prioritization during the training phase. However, the network architecture may also indirectly affect task prioritization. To better understand this effect (if any), we perform a series of ablation studies to measure the influence of a task hierarchy.

**Task Hierarchy.** A *task hierarchy* refers to some arbitrary ordering of tasks, usually motivated by the inter-task relationships. This ordering may manifest itself through the underlying network architecture. In this work, we experiment with a task hierarchy based on the relative difficulties between different tasks.

Consider a task set  $T$  with the task ordering  $T_1, T_2, \dots, T_{|T|}$ . When placed in a hierarchy, the task  $T_t$  is processed before being fed into the next task  $T_{t+1}$ . In contrast, a multitask model not arranged in a task hierarchy, such as hard parameter sharing, would process the tasks in parallel, where all tasks  $T_1, T_2, \dots, T_{|T|}$  consume the same learned representation  $\phi(x)$  where  $x$  is the input and  $\phi$  is an arbitrary function (e.g., neural network). Typically, there are no cross-task dependencies after  $\phi(x)$ . Different from [23], our task hierarchy is not multi-stage; all tasks are computed in a single pass.

**Network Architecture.** To encode a task hierarchy at the architectural level, we enforce unidirectional feedforward connections between different layers



**Fig. 2: Comparison of multitask learning architectures.** Shades of blue denote different task-specific layers. Gray rectangles with  $B_t$  denote a backbone block,  $M_t$  denotes a task-specific submodule,  $x$  is the input. (a) Hard parameter sharing: the standard approach to multitask learning,  $T = 2$ . (b) UberNet [37],  $T = 2$ . (c) Task hierarchy used in our ablation studies,  $T = 4$ .

of a neural network (see Figure 2). Given an input  $x$  and a task hierarchy  $T$ , the input  $x$  is fed into a neural network module which we refer to as the *backbone*. The backbone consists of  $|T|$  submodules, which we call *backbone blocks*, denoted as  $B_1, B_2, \dots, B_{|T|}$ . For each task  $T_i$  in the curriculum, a backbone block  $B_i$  feeds a task-specific module  $M_i$  (e.g., deconvolution [74], pointwise convolution). Additionally, the backbone block  $B_i$  feeds into the next tasks’s backbone block,  $B_{i+1}$ . To pass features between backbone blocks, transition layers are inserted between blocks. This structure is stacked  $|T|$  times in the network architecture to create a model that encodes the task ordering specified by the task hierarchy  $T$  (see Figure 2c).

For any task  $T_t$  and any input  $x$ , the progression through the task hierarchy is defined by the following recurrence relation, where  $\phi$  is a learned representation:

$$\phi_0 = x \quad \text{and} \quad \phi_t = (M_t \circ B_t)(x) \quad (6)$$

where  $B_t(x) = (B_{t-1} \circ B_{t-2} \dots \circ B_1)(x)$  and  $\circ$  denotes function composition. One such hierarchy is to order tasks such that  $\mathcal{D}(T_t) \geq \mathcal{D}(T_{t+1}), \forall t \in |T|$ , where the task difficulty  $\mathcal{D}(t) = \text{FL}(\bar{\kappa}_t; \gamma_t)$ , is defined in §2.1. The result is a hierarchy where more difficult tasks are processed before easier tasks.

To summarize: In a task hierarchy, the output from a lower-level task is provided as input to a higher-level task. This is in contrast to hard parameter sharing (Figure 2a), where there is no concept of “lower-level” task in the architecture. UberNet (Figure 2b) consists of a hierarchy, but the task-specific submodules still share a critical layer.



## 4 Experiments

The goal of this work is to dynamically prioritize difficult tasks during multitask learning. Our experiments are three-fold:

1. We perform an analysis to show the importance of task-level prioritization.
2. We present two ablation studies to measure: (i) explicit priority from our *dynamic task prioritization* and (ii) implicit priority from a task hierarchy.
3. We compare our proposed method with existing single-task methods on standard computer vision tasks and datasets.

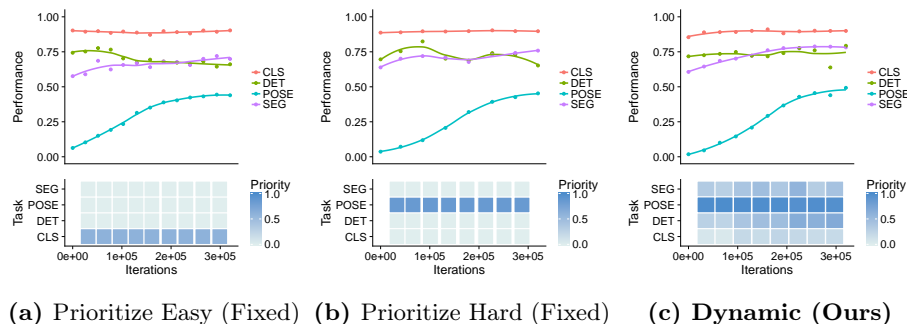
**Datasets.** We evaluate our approach on four core computer vision tasks: classification, segmentation, detection and pose estimation. We use the COCO 2017 dataset [27] and the MPII Human Pose dataset [28]. To use the full set of labels provided by these datasets, we focus on human understanding tasks where at most one person is present in the image. The reason for limiting images to zero or one person is to enable simpler flow of information between tasks. Scaling to multiple people is a matter of using more sophisticated task-specific decoder modules. Regardless, our method can be extended to multiple people through iterative application as has been done in prior work [75–77].

**Evaluation Metrics.** Classification is evaluated using the top-1 classification accuracy. For segmentation and detection we use the standard COCO metrics [27], primarily average precision (AP). We use: AP, AP<sub>50</sub>, AP<sub>75</sub>, AP<sub>S</sub>, AP<sub>M</sub>, and AP<sub>L</sub>, where the subscript refers to the minimum overlap threshold for a positive detection. Intersection-over-union (IoU) is the metric thresholded at 50% to 95%, in 5% increments, for small [S] (area < 32<sup>2</sup>), medium [M] (32<sup>2</sup> ≥ area < 96<sup>2</sup>), and large [L] (area ≤ 96<sup>2</sup>) objects [27]. For pose estimation, we use the standard MPII metric: percentage of correct keypoints (PCKh) metric [78]. The PCKh metric accounts for a person’s size in the image based on the length of the head-neck segment. If the predicted two-dimensional pose coordinate is within  $\epsilon$  pixels of the ground truth pose coordinate, the prediction is considered correct. The tolerance  $\epsilon$  is proportional to the ground truth head-neck length. Implementation details and hyperparameters can be found in Appendix B.

### 4.1 Task-Level Prioritization

Our first experiment is to evaluate different task-weighting schemes, including our *dynamic task prioritization* method. We trained a single hard parameter sharing model (Figure 2) to simultaneously predict classification, segmentation, detection, and pose estimation labels but used different weighting/prioritization mechanisms. The only difference between the weighting schemes is the task weights – some are fixed and some are dynamic. The network architecture consists of a single shared backbone (i.e., DenseNet [79]) with the final layer fed into multiple task-specific layers.

**Baselines.** Table 1 shows that our weighting scheme can outperform other multitask learning weighting schemes. We evaluated the following:

(a) Prioritize Easy (Fixed) (b) Prioritize Hard (Fixed) (c) **Dynamic (Ours)**

**Fig. 3: Comparison of task-level prioritization schemes.** (Top; line plot) Performance for each task on the validation set. Higher is better. The  $x$  axis denotes the number of training steps. (Bottom; square tiles) Priority level of each task during training. Darker colors denote to higher priority.

- *Uniform*: Each task is added together to produce a single scalar loss value.
- *Prioritize Easy*: Classification has weight 0.97, all other have weight 0.01.
- *Prioritize Hard*: Pose estimation has weight 0.97, all others have weight 0.01.
- *Hand-Crafted*: Pose estimation, detection, segmentation, and classification are given weights of 0.4, 0.3, 0.2, and 0.1 respectively (Selected by grid-search).
- *Loss Exponentiation*: Un-weighted loss outputs are raised to the power 1.2. This assumes larger loss magnitudes indicate more difficult tasks. The power of 1.2 was selected by grid-search.
- *Homoscedastic Uncertainty* [36]: Uses uncertainty, which is related to loss magnitude, to automatically weight different tasks.
- *Self-Paced* [43]: Task weights are learnable parameters and are regularized to encourage selecting the easy tasks in the earlier iterations of training.

Our method, *dynamic priority*, adaptively adjusts task-level priorities throughout the training procedure. This is apparent in Figure 3c. Initially, pose is given the highest priority, and over time, the model slowly increases the priority of detection and segmentation. Note that this is slightly different from our final proposed method, evaluated in §4.3. Our final model combines task-level priority with example-level priority, whereas the model in Table 1 and Figure 3c only applies a task-level priority.

## 4.2 Ablation Studies

Our ablation studies consist of two components: (i) analyzing our proposed *dynamic task prioritization* method and (ii) analyzing the effect of a task hierarchy.

**Dynamic Task Prioritization: Focusing Parameter  $\gamma$ .** To better understand the interaction between the task- and example-level focusing parameters  $\gamma_0, \dots, \gamma_t$ , we provide an ablation study in Table 2. In this experiment, we trained a hard-parameter sharing model on all four tasks. The difference between each

Method	C Top-1	S AP <sub>50</sub>	D AP <sub>50</sub>	P AP <sub>50</sub>
Uniform	90.14	78.57	75.29	47.72
Prioritize Easy	88.69	62.45	67.36	45.48
Prioritize Hard	89.92	75.80	65.86	55.96
Prioritize Easy (LR Schedule)	88.32	61.29	64.68	47.72
Prioritize Hard (LR Schedule)	89.44	72.84	<b>82.64</b>	56.09
Hand-Crafted	90.62	78.06	76.80	50.28
Loss Exponentiation	88.85	73.99	67.68	42.52
Homoscedastic Uncertainty [36]	88.59	73.56	65.01	48.72
Focal Loss ( $\gamma = 1$ )	88.80	70.77	65.05	54.04
Self-Paced [43]	89.76	74.46	78.18	49.34
Dynamic Priority - Easy	90.20	68.00	76.20	32.90
Dynamic Priority - Hard (Ours)	<b>91.05</b>	<b>78.83</b>	79.31	<b>56.11</b>

**Table 1: Comparison of task-level prioritization schemes.** Letters denote classification (C), segmentation (S), detection (D), and pose estimation (P) tasks. LR Schedule refers to a learning rate schedule (see Appendix B).

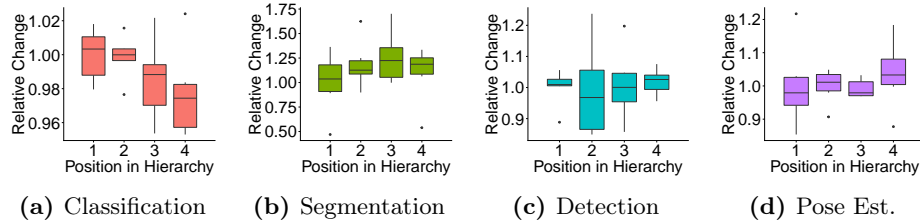
Method	C Top-1	S AP <sub>50</sub>	D AP <sub>50</sub>	P AP <sub>50</sub>
FL ( $\gamma_0=1$ )	74.0	74.0	82.0	49.1
FL ( $\gamma_0=2$ )	71.8	71.8	69.9	47.4
DTP ( $\gamma_1, \dots, \gamma_t=1$ )	<b>91.1</b>	<b>78.8</b>	79.3	<b>56.1</b>
DTP ( $\gamma_1, \dots, \gamma_t=2$ )	88.8	76.5	68.4	54.3
FL+DTP ( $\gamma_0, \dots, \gamma_t=1$ )	88.9	77.8	<b>85.9</b>	52.7
FL+DTP ( $\gamma_0, \dots, \gamma_t=2$ )	88.4	76.7	82.4	54.9

**Table 2: Effect of example- and task-level focusing parameters.** FL denotes Focal Loss [39], applied to example-level weighting. DTP denotes our dynamic task prioritization loss on task-level weighting.  $\gamma_0$  denotes the example-level focusing parameter and  $\gamma_1, \dots, \gamma_t$  denotes the task-level focusing parameter.

run was the inclusion or exclusion of example- or task-level weighting. We also varied the focusing parameter value.

Increasing the focusing parameter exponentially marks easier examples and tasks as unimportant. By increasing  $\gamma_0$ , performance for classification and segmentation decrease. Surprisingly, detection and pose estimation AP improves on FL and FL+DTP when  $\gamma_0$  increases from 1.0 to 2.0. Intuitively, this makes sense pose estimation is more difficult than classification and segmentation (i.e., pose estimation is a multi-regression task). A larger  $\gamma_0$  forces the model to focus on detection and pose estimation, but unfortunately at the cost of classification and segmentation performance.

**Task Hierarchy: Effect of the Task Ordering.** This paper focuses on the tasks of classification, person segmentation, person detection and human pose estimation. Enumerating the possible task orderings results in  $4! = 24$



**Fig. 4: Effect of task ordering on performance.** The  $x$  axis denotes position in the hierarchy (e.g., 1: first task, 4: last task). Each bar/box denotes the middle 50%. The  $y$  axis denotes the relative performance of a model if you place a task  $T_i$  at a specified position in the hierarchy, compared to a model trained with  $T_i$  at the first position. If  $y \geq 1.0$ , this means task  $T_i$  performs better at that position than a model with task  $T_i$  positioned at the beginning of the curriculum. For example, in (a), the classification task tends to perform better when placed before other tasks. Black dots represent outliers, defined as more than  $1.5\times$  away from the inter-quartile range. A table of full results is given in Appendix A.

permutations. We conducted an experiment where we train and evaluate 24 models, each with a different task permutation. While this is an exhaustive search, the goal of this experiment is not to find the optimal ordering but rather determine if such an ordering has an effect on performance.

We use a densely connected convolutional network [79] as the backbone of the task hierarchy (see Figure 2c). The classification module consists of a linear layer to generate classification predictions. The segmentation module consists of a small fully convolutional network [80] and outputs a segmentation mask. For the detection and pose modules, we used point-wise convolutions to regress vectors which parameterize bounding boxes and 2D body part positions.

An analysis of the ordering experiment is shown in Figure 4. It is clear that some tasks perform better at different positions in the hierarchy. Figure 4a shows that classification performs better when placed at the beginning of the hierarchy (i.e., the first layer, see Figure 2c). Segmentation demonstrates significantly improved performance when placed in later layers of the hierarchy (see Figure 4b). When placed in the center of the task hierarchy network at position 3, segmentation performance is boosted by  $1.25\times$ . Detection (Figure 4c) is fairly robust to its position in the hierarchy. Pose estimation (Figure 4d) also appears to be robust to its position, but the high variances may prove to be inconclusive.

**Task Hierarchy for Multitask Learning.** Our experiments thus far suggest that a task hierarchy does impact performance – especially for the case of classification and segmentation (see Figure 4). We now pose the following question: How does a task hierarchy compare to existing multitask methods, such as hard parameter sharing?

*Baselines.* We evaluate the commonly used hard parameter sharing model [22], where multiple task heads branch out from a single *critical* layer near the end of the network. Additionally, we evaluate the UberNet [37] architecture for

Models	C Top-1	S AP <sub>50</sub>	D AP <sub>50</sub>	P AP <sub>50</sub>
Hard Sharing [22]	91.05	78.83	79.31	56.11
UberNet [37]	89.76	77.81	80.06	55.47
Task Hierarchy	<b>91.16</b>	<b>80.26</b>	<b>80.92</b>	<b>62.40</b>

**Table 3: Comparison with task hierarchy on COCO.** Letters denote classification (C), segmentation (S), detection (D), and pose estimation (P) tasks.

multitask learning. Visually, these baselines are illustrated in Figure 2. We briefly discuss the experimental configuration of each multitask method:

- *Hard Parameter Sharing* [22]. A DenseNet [79] was used as the shared model. The output feature map of the shared model is fed into individual task modules (i.e., readout functions or decoders).
- *UberNet* [37]. We similarly use a DenseNet as the “trunk”. Each dense block outputs to a batch normalization [81] layer, which branches out into task modules. Each layer outputs task-specific features.
- *Task Hierarchy*. We also use a DenseNet as the backbone. Each dense block outputs to a different task module. The ordering selected for this experiment is the best ordering, as discovered from Figure 4: classification, segmentation, detection, and pose estimation.

UberNet [37], a variant of hard parameter sharing, is a unified architecture for jointly training multiple tasks in parallel. They demonstrate competitive performance with state-of-the-art single-task models when training on one or two tasks. However, when scaling to several tasks, performance deteriorates [37]. We believe their observation can be attributed to task difficulty. This leads to a key difference between our work and UberNet: Our method learns representations in a task hierarchy ordered by task difficulty, whereas UberNet learns a standard deep learning feature hierarchy [82].

*Results.* Table 3 compares hard parameter sharing, UberNet, and our task hierarchy. Each baseline contained identical backbone and decoder modules. As was apparent in 1, in this task hierarchy study, we also observe the effects of transfer learning. Our task hierarchy outperforms hard sharing and UberNet with a wide margin for pose estimation – our most difficult task. Classification and detection demonstrate comparable performance, with a slight improvement in segmentation accuracy by our task hierarchy.

As a reminder, from our definition of task difficulty in Section 3, task performance serves as a proxy for difficulty. The results in Figure 4 suggest that pose estimation and detection are significantly more difficult than classification and segmentation. This is evident from the quantitative results in Table 2, which analyzed our *dynamic task prioritization*, and also from Figure 4 and Table 3, which suggest that a task hierarchy does impose a notion of priority – with pose estimation being the most difficult task and classification being the easiest. Therefore, we adopt the following hierarchy: classification first, detection second, segmentation third, and pose estimation last.

		COCO				MPII
Task	Method	AP	AP <sub>50</sub>	AP <sub>75</sub>	Accuracy	AP <sub>50</sub>
Classification	DenseNet [79]	—	—	—	89.8	—
Classification	Our Method	—	—	—	<b>91.5</b>	—
Segmentation	FCN [80]	<b>51.9</b>	<b>83.6</b>	<b>55.5</b>	—	—
Segmentation	Our Method	49.4	83.2	50.7	—	—
Detection	RetinaNet [39]	49.4	<b>83.7</b>	<b>57.0</b>	—	—
Detection	Our Method	<b>52.5</b>	80.2	53.1	—	—
Pose Estimation	Hourglass [83]	59.9	—	—	—	43.9
Pose Estimation	Our Method	<b>62.4</b>	—	—	—	<b>45.8</b>

**Table 4: Single-task versus our multitask, dynamic-task priority model.** We compare a single instance of our dynamic-task priority model (trained on all four tasks simultaneously) with single-task methods.

### 4.3 Comparison to Single-Task Models

Having analyzed the independent effect of our proposed *dynamic task prioritization* scheme and the indirect effects of a task hierarchy, we now combine these two technical insights into a single, unified model. In this experiment, we train a single model equipped with dynamic task prioritization. It is trained jointly on classification, segmentation, detection and pose estimation. We compare our model to existing state-of-the-art single-task models such as RetinaNet [39], FCN [80], and stacked hourglass networks [83]. To keep our model’s parameter count as close as possible to each single-task model, we use identical task-specific modules. Table 4 shows the results.

For the detection task, RetinaNet [39] demonstrated an  $AP_S$ ,  $AP_M$ , and  $AP_L$  of 11.8, 45.6, and 70.8, respectively. Our method demonstrated an  $AP_S$ ,  $AP_M$ , and  $AP_L$  of 12.78, 40.6, and 70.5. While our method performs better on smaller objects, RetinaNet outperforms our method on medium and large – indicating comparable overall performance. We can see that our method, which is simultaneously trained on the classification, segmentation, detection, and pose tasks, is capable of competitive results with state-of-the-art models.

## 5 Conclusion

In this work, we proposed *dynamic task prioritization* for multitask learning. Our method encourages a model to learn from difficult examples and difficult tasks. Ablation studies analyzed the effect of explicit priority generated by our proposed method and the implicit priority generated by a task hierarchy, embedded in the network architecture. In conclusion, we showed that training a single multitask model with *dynamic task prioritization* can achieve competitive performance with existing single-task models. We believe our results provide useful insights for both the application and research of single-task and multitask learning methods.

## References

1. Coviello, D., Ichino, A., Persico, N.: Time allocation and task juggling. *The American Economic Review* (2014)
2. Kenny, J., Fluck, A., Jetson, T., et al.: Placing a value on academic work: The development and implementation of a time-based academic workload model. *Australian Universities' Review*, The (2012)
3. Kenny, J.D., Fluck, A.E.: The effectiveness of academic workload models in an institution: a staff perspective. *Journal of Higher Education Policy and Management* (2014)
4. Bellotti, V., Dalal, B., Good, N., Flynn, P., Bobrow, D.G., Ducheneaut, N.: What a to-do: studies of task management towards the design of a personal task list manager. In: *Conference on Human factors in computing systems*. (2004)
5. Kember, D.: Interpreting student workload and the factors which shape students' perceptions of their workload. *Studies in higher education* (2004)
6. Yang, Y., Hospedales, T.: Deep multi-task representation learning: A tensor factorisation approach. *arXiv* (2016)
7. Jou, B., Chang, S.F.: Deep cross residual learning for multitask visual recognition. In: *Multimedia Conference*. (2016)
8. Misra, I., Shrivastava, A., Gupta, A., Hebert, M.: Cross-stitch networks for multi-task learning. In: *CVPR*. (2016)
9. Luong, M.T., Le, Q.V., Sutskever, I., Vinyals, O., Kaiser, L.: Multi-task sequence to sequence learning. *arXiv* (2015)
10. Hashimoto, K., Xiong, C., Tsuruoka, Y., Socher, R.: A joint many-task model: Growing a neural network for multiple nlp tasks. *arXiv* (2016)
11. Dong, D., Wu, H., He, W., Yu, D., Wang, H.: Multi-task learning for multiple language translation. In: *ACL*. (2015)
12. Collobert, R., Weston, J.: A unified architecture for natural language processing: Deep neural networks with multitask learning. In: *ICML*. (2008) 160–167
13. Augenstein, I., Ruder, S., Søgaard, A.: Multi-task learning of pairwise sequence classification tasks over disparate label spaces. *arXiv* (2018)
14. Wu, Z., Valentini-Botinhao, C., Watts, O., King, S.: Deep neural networks employing multi-task learning and stacked bottleneck features for speech synthesis. In: *ICASSP*. (2015)
15. Seltzer, M.L., Droppo, J.: Multi-task learning in deep neural networks for improved phoneme recognition. In: *ICASSP*. (2013)
16. Huang, J.T., Li, J., Yu, D., Deng, L., Gong, Y.: Cross-language knowledge transfer using multilingual deep neural network with shared hidden layers. In: *ICASSP*. (2013) 7304–7308
17. Jaderberg, M., Mnih, V., Czarnecki, W.M., Schaul, T., Leibo, J.Z., Silver, D., Kavukcuoglu, K.: Reinforcement learning with unsupervised auxiliary tasks. *arXiv* (2016)
18. Rusu, A.A., Rabinowitz, N.C., Desjardins, G., Soyer, H., Kirkpatrick, J., Kavukcuoglu, K., Pascanu, R., Hadsell, R.: Progressive neural networks. *arXiv* (2016)
19. Devin, C., Gupta, A., Darrell, T., Abbeel, P., Levine, S.: Learning modular neural network policies for multi-task and multi-robot transfer. In: *ICRA*. (2017)
20. Fernando, C., Banarse, D., Blundell, C., Zwols, Y., Ha, D., Rusu, A.A., Pritzel, A., Wierstra, D.: Pathnet: Evolution channels gradient descent in super neural networks. *arXiv* (2017)

21. Kaiser, L., Gomez, A.N., Shazeer, N., Vaswani, A., Parmar, N., Jones, L., Uszkoreit, J.: One model to learn them all. *arXiv* (2017)
22. Caruna, R.: Multitask learning: A knowledge-based source of inductive bias. In: *ICML*. (1993)
23. Pentina, A., Sharmanska, V., Lampert, C.H.: Curriculum learning of multiple tasks. In: *CVPR*. (June 2015)
24. Bengio, Y., Louradour, J., Collobert, R., Weston, J.: Curriculum learning. In: *ICML*. (2009)
25. Sung, K.K., Poggio, T.: Example-based learning for view-based human face detection. *T-PAMI* (1998)
26. Felzenszwalb, P.F., Girshick, R.B., McAllester, D., Ramanan, D.: Object detection with discriminatively trained part-based models. *T-PAMI* (2010)
27. Lin, T.Y., Maire, M., Belongie, S., Hays, J., Perona, P., Ramanan, D., Dollár, P., Zitnick, C.L.: Microsoft coco: Common objects in context. In: *ECCV*, Springer (2014)
28. Andriluka, M., Pishchulin, L., Gehler, P., Schiele, B.: 2d human pose estimation: New benchmark and state of the art analysis. In: *CVPR*. (June 2014)
29. Elman, J.L.: Learning and development in neural networks: The importance of starting small. *Cognition* (1993)
30. Pentina, A., Sharmanska, V., Lampert, C.H.: Curriculum learning of multiple tasks. In: *CVPR*. (2015)
31. Graves, A., Bellemare, M.G., Menick, J., Munos, R., Kavukcuoglu, K.: Automated curriculum learning for neural networks. *arXiv* (2017)
32. Zaremba, W., Sutskever, I.: Learning to execute. *arXiv* (2014)
33. Luo, Z., Zou, Y., Hoffman, J., Fei-Fei, L.: Label efficient learning of transferable representations across domains and tasks. In: *NIPS*. (2017)
34. Glorot, X., Bordes, A., Bengio, Y.: Domain adaptation for large-scale sentiment classification: A deep learning approach. In: *ICML*. (2011)
35. Tzeng, E., Hoffman, J., Darrell, T., Saenko, K.: Simultaneous deep transfer across domains and tasks. In: *ICCV*. (2015)
36. Kendall, A., Gal, Y., Cipolla, R.: Multi-task learning using uncertainty to weigh losses for scene geometry and semantics. *CVPR* (2018)
37. Kokkinos, I.: Ubernet: Training a universal convolutional neural network for low-, mid-, and high-level vision using diverse datasets and limited memory. *CVPR* (2017)
38. Sermanet, P., Eigen, D., Zhang, X., Mathieu, M., Fergus, R., LeCun, Y.: Overfeat: Integrated recognition, localization and detection using convolutional networks. *arXiv* (2013)
39. Lin, T.Y., Goyal, P., Girshick, R., He, K., Dollár, P.: Focal loss for dense object detection. *arXiv* (2017)
40. Chen, Z., Badrinarayanan, V., Lee, C.Y., Rabinovich, A.: Gradnorm: Gradient normalization for adaptive loss balancing in deep multitask networks. *arXiv* (2017)
41. Kumar, M.P., Packer, B., Koller, D.: Self-paced learning for latent variable models. In: *NIPS*. (2010)
42. Xu, D., Alameda-Pineda, X., Song, J., Ricci, E., Sebe, N.: Cross-paced representation learning with partial curricula for sketch-based image retrieval. *arXiv* (2018)
43. Li, C., Yan, J., Wei, F., Dong, W., Liu, Q., Zha, H.: Self-paced multi-task learning. In: *AAAI*. (2017)
44. Xu, W., Liu, W., Chi, H., Huang, X., Yang, J.: Multi-task classification with sequential instances and tasks. *Signal Processing: Image Communication* (2018)



45. Oudeyer, P.Y., Kaplan, F., Hafner, V.V.: Intrinsic motivation systems for autonomous mental development. *Transactions on Evolutionary Computation* (2007)
46. urgen Schmidhuber, J.: A possibility for implementing curiosity and boredom in model-building neural controllers. In: *From animals to animats: proceedings of the first international conference on simulation of adaptive behavior*. (1991)
47. Storck, J., Hochreiter, S., Schmidhuber, J.: Reinforcement driven information acquisition in non-deterministic environments. In: *International Conference on Artificial Neural Networks*. (1995)
48. Itti, L., Baldi, P.: Bayesian surprise attracts human attention. *Vision research* (2009)
49. Houthoofd, R., Chen, X., Duan, Y., Schulman, J., De Turck, F., Abbeel, P.: Vime: Variational information maximizing exploration. In: *NIPS*. (2016)
50. Rosenbaum, C., Klinger, T., Riemer, M.: Routing networks: Adaptive selection of non-linear functions for multi-task learning. *arXiv* (2017)
51. Zoph, B., Le, Q.V.: Neural architecture search with reinforcement learning. *arXiv* (2016)
52. Bellemare, M., Srinivasan, S., Ostrovski, G., Schaul, T., Saxton, D., Munos, R.: Unifying count-based exploration and intrinsic motivation. In: *NIPS*. (2016)
53. He, K., Gkioxari, G., Dollár, P., Girshick, R.: Mask r-cnn. *ICCV* (2017)
54. Bilen, H., Vedaldi, A.: Integrated perception with recurrent multi-task neural networks. In: *NIPS*. (2016)
55. Redmon, J., Farhadi, A.: Yolo9000: better, faster, stronger. In: *CVPR*. (2017)
56. Ranjan, R., Patel, V.M., Chellappa, R.: Hyperface: A deep multi-task learning framework for face detection, landmark localization, pose estimation, and gender recognition. *T-PAMI* (2017)
57. Anastasopoulos, A., Chiang, D.: Tied multitask learning for neural speech translation. *arXiv* (2018)
58. Baxter, J.: A bayesian/information theoretic model of learning to learn via multiple task sampling. *Machine learning* (1997)
59. Meyerson, E., Miikkulainen, R.: Pseudo-task augmentation: From deep multitask learning to intratask sharing-and back. *arXiv* (2018)
60. Ruder, S.: An overview of multi-task learning in deep neural networks. *arXiv* (2017)
61. Dai, J., He, K., Sun, J.: Instance-aware semantic segmentation via multi-task network cascades. In: *CVPR*. (2016) 3150–3158
62. Teichmann, M., Weber, M., Zoellner, M., Cipolla, R., Urtasun, R.: Multinet: Real-time joint semantic reasoning for autonomous driving. *arXiv* (2016)
63. Ben-David, S., Borbely, R.S.: A notion of task relatedness yielding provable multiple-task learning guarantees. *Machine learning* (2008)
64. Meyerson, E., Miikkulainen, R.: Beyond shared hierarchies: Deep multitask learning through soft layer ordering. *ICLR* (2018)
65. Kang, Z., Grauman, K., Sha, F.: Learning with whom to share in multi-task feature learning. In: *ICML*. (2011)
66. Collobert, R., Weston, J., Bottou, L., Karlen, M., Kavukcuoglu, K., Kuksa, P.: Natural language processing (almost) from scratch. *JMLR* (2011)
67. Søgaard, A., Goldberg, Y.: Deep multi-task learning with low level tasks supervised at lower layers. In: *Association for Computational Linguistics*. (2016)
68. Chen, Q., Zhu, X., Ling, Z., Wei, S., Jiang, H.: Enhancing and combining sequential and tree lstm for natural language inference. *arXiv* (2016)
69. Eriguchi, A., Hashimoto, K., Tsuruoka, Y.: Tree-to-sequence attentional neural machine translation. *arXiv* (2016)

70. Zamir, A.R., Wu, T.L., Sun, L., Shen, W.B., Shi, B.E., Malik, J., Savarese, S.: Feedback networks. In: CVPR. (2017)
71. Yu, J., Jiang, Y., Wang, Z., Cao, Z., Huang, T.: Unitbox: An advanced object detection network. In: Multimedia Conference. (2016)
72. Rahman, M.A., Wang, Y.: Optimizing intersection-over-union in deep neural networks for image segmentation. In: International Symposium on Visual Computing. (2016)
73. Kingma, D., Ba, J.: Adam: A method for stochastic optimization. arXiv (2014)
74. Zeiler, M.D., Krishnan, D., Taylor, G.W., Fergus, R.: Deconvolutional networks. In: CVPR. (2010)
75. Pishchulin, L., Jain, A., Andriluka, M., Thormählen, T., Schiele, B.: Articulated people detection and pose estimation: Reshaping the future. In: CVPR. (2012)
76. Gkioxari, G., Hariharan, B., Girshick, R., Malik, J.: Using k-poselets for detecting people and localizing their keypoints. In: CVPR. (2014)
77. Iqbal, U., Gall, J.: Multi-person pose estimation with local joint-to-person associations. In: ECCV, Springer (2016)
78. Andriluka, M., Pishchulin, L., Gehler, P., Schiele, B.: 2d human pose estimation: New benchmark and state of the art analysis. In: CVPR. (2014)
79. Huang, G., Liu, Z., Weinberger, K.Q., van der Maaten, L.: Densely connected convolutional networks. CVPR (2017)
80. Long, J., Shelhamer, E., Darrell, T.: Fully convolutional networks for semantic segmentation. In: CVPR. (2015)
81. Ioffe, S., Szegedy, C.: Batch normalization: Accelerating deep network training by reducing internal covariate shift. In: ICML. (2015) 448–456
82. LeCun, Y., Bengio, Y., Hinton, G.: Deep learning. Nature (2015)
83. Newell, A., Yang, K., Deng, J.: Stacked hourglass networks for human pose estimation. In: ECCV, Springer (2016)
84. Paszke, A., Chintala, S., Collobert, R., Kavukcuoglu, K., Farabet, C., Bengio, S., Melvin, I., Weston, J., Mariethoz, J.: Pytorch: Tensors and dynamic neural networks in python with strong gpu acceleration (2017)
85. Abadi, M., Agarwal, A., Barham, P., Brevdo, E., Chen, Z., Citro, C., Corrado, G.S., Davis, A., Dean, J., Devin, M., et al.: Tensorflow: Large-scale machine learning on heterogeneous distributed systems. arXiv:1603.04467 (2016)
86. Ren, S., He, K., Girshick, R., Sun, J.: Faster r-cnn: Towards real-time object detection with region proposal networks. In: NIPS. (2015)

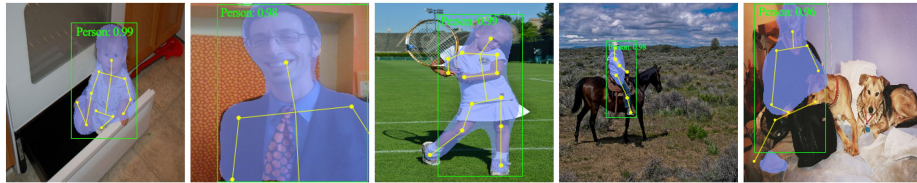
## Appendix A: Additional Results

### A.1 Task Hierarchy: Effect of the Task Order

In Figure 4, we evaluated the effect of task ordering (i.e., task hierarchy) on multitask performance. While the results demonstrate high variance for most tasks, when examining classification and segmentation, there does appear to be a trend: classification performs better when placed lower on the hierarchy (i.e., first position) and segmentation does better at position three. An exhaustive search was conducted for these tasks, but the goal of this experiment was not to find the optimal ordering but rather demonstrate that ordering is important. The results in Figure 4 mildly suggest this.

Ordering	C Top-1	S AP <sub>50</sub>	D AP <sub>50</sub>	P AP <sub>50</sub>
CDPS	1.02×	1.33×	1.06×	1.02×
CDSP	1.01×	1.28×	1.03×	1.09×
CPDS	0.98×	1.22×	0.86×	1.05×
CPSD	1.00×	1.17×	1.04×	0.98×
CSDP	1.00×	1.62×	1.20×	1.18×
CSPD	0.98×	1.11×	0.99×	0.97×
DCPS	1.02×	1.06×	1.01×	0.99×
DCSP	1.00×	1.38×	1.03×	1.02×
DPCS	0.99×	1.26×	1.06×	1.00×
DPSC	0.98×	1.01×	1.00×	1.02×
DSCP	1.00×	1.25×	1.01×	1.00×
DSPC	0.98×	0.90×	0.89×	0.97×
PCDS	1.00×	1.15×	0.96×	0.94×
PCSD	0.98×	1.00×	1.08×	1.02×
PDCS	0.97×	0.54×	0.85×	0.85×
PDSC	1.02×	1.70×	1.24×	1.22×
PSCD	0.99×	1.08×	1.02×	0.94×
PSDC	0.95×	1.14×	1.05×	1.03×
SCDP	1.00×	1.36×	1.04×	1.04×
SCPD	1.00×	1.11×	0.96×	1.03×
SDCP	0.95×	0.47×	0.85×	0.88×
SDPC	0.97×	0.96×	0.90×	0.97×
SPCD	1.02×	1.20×	1.04×	1.04×
SPDC	0.95×	0.89×	0.95×	0.91×

**Table 5: Relative performance of different task hierarchies.** Values denote the relative performance of a model if you place a task  $T_i$  at a specified position in the hierarchy, compared to a model trained with  $T_i$  at the first position. If the value  $\geq 1.0$ , this means task  $T_i$  performs better at that position than a model with task  $T_i$  positioned at the beginning of the curriculum. Models were trained and evaluated on COCO.



**Fig. 5: Qualitative results from our model.** Yellow dots denote body part predictions. Blue overlay denotes the segmentation result. Green rectangles denote the detection result. Classification probabilities are shown in green text.

Table 5 shows full results for the 24 experiments. Each cell in the table denotes the performance of a single model on a task  $T_i$  with the specified task hierarchy on the left. Values greater than 1.0 indicate that  $T_i$  performs better when placed at this position in the hierarchy compared to a model with  $T_i$  placed at the beginning of the hierarchy.

## A.2 Qualitative Results

Figure 5 shows qualitative results from our model. Specifically, the results were generated using a single model, equipped with dynamic task priority, trained simultaneously on the four tasks. Overall, the model does a good job at segmentation, as is shown in the middle image. Classification accuracy performs well, often scoring above 95% confident. Intuitively, the detection task should go hand-in-hand with the pose estimation result (i.e., the pose skeleton should be fully contained within the bounding box), but this is not the case for the middle and far right images.

## Appendix B: Implementation Details

In this section, we provide detailed information required for reproducing our results. All experiments were primarily implemented in PyTorch [84] with some components written in TensorFlow [85].

**Training Details.** For all experiments we used a learning rate of  $10^{-4}$  using the Adam optimizer [73]. To account for the different magnitudes in different task losses, we applied a magnitude normalization term to translate all losses into the same magnitude scaling. We used scaling terms  $10^5$ ,  $10^{-1}$ ,  $4 \times 10^2$ , and  $4 \times 10^0$  for the classification, segmentation, detection, and pose estimation losses, respectively.

**Optimization Objectives.** For our experiments, task-specific loss objectives follow standard evaluation protocols. Discrete labeling tasks (i.e., classification and segmentation) use the cross-entropy loss. For detection, we use the detection loss specified in Faster-RCNN [86], which is a classification plus regression loss. For pose estimation, the mean squared error was computed. To

conform with our proposed loss  $\mathcal{L}_{\text{DTP}}$ , the mean squared error was normalized to  $[0, 1]$  using the size of the image as the denominator.

### B.1 Comparison of Task-Weighting Methods

The goal of this experiment is to demonstrate the importance of prioritizing difficult tasks above easier tasks.

**Hard Parameter Sharing.** In this experiment we evaluate a number of task weighting methods using the hard-parameter sharing multitask learning method [22]. We used a single DenseNet-121 model [79] as the shared model base. For the pose estimation module, we utilized the stacked hourglass network with eight stacks and one block [83]. For the detection module, we used a RetinaFPN50 network [39]. For the segmentation module, we used a fully convolutional network [80] (see Table 6). For the classification module, we used a linear layer (see Table Table 7). To create an input of size  $256 \times 256 \times 3$  to the detection and pose estimation module, we upsampled the output of the DenseNet-121 from  $1024 \times 3 \times 3$  to  $256 \times 256 \times 3$ . See Table 10 for upsampler details.

**Self-Paced Multitask Loss.** For the self-paced multitask loss [43], we used  $\gamma = 0.5$  and a  $\mu_2 = 1.5$ . Since our goal of this experiment is purely to compare task weighting methods, we did not apply self-paced loss on individual examples. The learnable task weighting parameters we randomly initialized to  $[0, 1]$ .

**Difficulty-Weighted Multitask Loss.** The focusing parameters for the task-specific loss were defined as:  $\gamma_1, \gamma_2, \gamma_3, \gamma_4 = 1.0$ . Focal loss was not applied to individual examples: that is,  $\gamma_0 = 0$ . The key performance indicators were smoothed using an exponential moving average with a span of 2 and  $\alpha = 0.75$ .

Layer Name	Output Size	Filter Size	Layers
conv-1	$8 \times 8$	$1 \times 1$	512
deconv-1	$16 \times 16$	$5 \times 5$	256
deconv-2	$32 \times 32$	$5 \times 5$	128
deconv-3	$64 \times 64$	$5 \times 5$	64
deconv-4	$128 \times 128$	$5 \times 5$	32
deconv-5	$256 \times 256$	$5 \times 5$	2

**Table 6: Segmentation Module.** Not used for experiments in §4.3. Input to the segmentation module is  $1024 \times 8 \times 8$ . Produced output is a  $256 \times 256 \times 2$  segmentation map.

### B.2 Effect of Task Ordering on Performance

One of the goals in the paper is to evaluate the importance of focusing on difficult tasks in terms of a task hierarchy. To do this, we perform a comprehensive

Layer Name	Output Size	Filter Size	Layers
conv-1	$8 \times 8$	$1 \times 1$	512
conv-2	$8 \times 8$	$1 \times 1$	256
relu-2	$8 \times 8$	—	—
pool-2	$1 \times 1$	$7 \times 7$	stride 1
linear-3	2	1024	2

**Table 7: Classification Module.** Not used for experiments in §4.3. Input to the classification module is  $1024 \times 8 \times 8$ . The produced output is a vector of size 2.

Layer Name	Output Size	Filter Size	Layers
conv-1	$32 \times 32$	$3 \times 3$	32
conv-2	$16 \times 16$	$3 \times 3$	26
conv-3	$8 \times 8$	$3 \times 3$	26
conv-4	$4 \times 4$	$3 \times 3$	26
conv-5	$1 \times 1$	$4 \times 4$	26

**Table 8: Pose Estimation Module.** Not used in §4.3. Input is a tensor of size  $64 \times 64 \times 64$ , and a vector of size 26, corresponding to  $(x_i, y_i) \forall i \in J$  joints is predicted.

Layer Name	Output Size	Filter Size	Layers
conv-1	$32 \times 32$	$3 \times 3$	32
conv-2	$16 \times 16$	$3 \times 3$	16
conv-3	$8 \times 8$	$3 \times 3$	8
conv-4	$4 \times 4$	$3 \times 3$	4
conv-5	$1 \times 1$	$4 \times 4$	4

**Table 9: Detection Module.** Not used for experiments in §4.3. Input is a tensor of size  $64 \times 64 \times 64$ , and a vector of size 4 is predicted.

Layer Name	Output Size	Filter Size	Layers
conv-1	$8 \times 8$	$1 \times 1$	512
deconv-1	$16 \times 16$	$5 \times 5$	256
deconv-2	$32 \times 32$	$5 \times 5$	128
deconv-3	$64 \times 64$	$5 \times 5$	64
deconv-4	$128 \times 128$	$5 \times 5$	32
deconv-5	$256 \times 256$	$5 \times 5$	2

**Table 10: Upsampler.** Input to the upsampler module is  $1024 \times 8 \times 8$ , and the produced output is  $256 \times 256 \times 3$ . The upsampler was used in multiple experiments to align spatial dimensions.

analysis by enumerating all ordering possibilities on our four tasks in  $4! = 24$ . The results of the relative performance improvements are shown in Table 5.

**Task Hierarchy.** For our task hierarchy, we used a DenseNet-121 [79] as our convolutional backbone. We used the task module described in Table 6 for segmentation. The task module described in Table 7 was used for classification. For pose estimation and detection, we used the decoders described in Tables 8 and 9, respectively.

**Loss Weighting** For this experiment we applied the focal loss using  $\gamma = 2.0$ . Since our goal is to evaluate purely the effect of task ordering on model performance, we were not interested in applying our adaptive weighting. Thus for this experiment we did not apply any weighting to the task losses (i.e. uniform task weighting distribution).

### B.3 Comparison of Multitask Approaches

The goal of this experiment is to demonstrate that our proposed task hierarchy is able to outperform existing multitask learning approaches. We compare our multitask learning architecture to two existing approaches: hard parameter sharing [22] and UberNet [37]. For our method, we used the same network architecture and training settings described in Section B.2. For hard parameter sharing, we used the same architecture and training settings as described in Section B.1. For the UberNet architecture [37], we used a DenseNet-121 architecture as the convolutional trunk. We defined branching to occur at the end of every dense block in the DenseNet architecture, after the transition layers. We inserted batch normalization layers [81] at every dense block. We then combined the batch normalization outputs across all blocks using addition. Finally the combined batch normalization outputs were fed into task modules.

**Task Modules.** The task modules were controlled across all three multitask learning approaches (hard parameter sharing [22], UberNet [37], and our task hierarchy approach). For classification, we used the classification module described in Table 7. For segmentation, we used an FCN [80] described in Table 6. For detection, we used a RetinaFPN50 network [39]. For pose estimation, we

used a stacked hourglass network consisting of eight stacks with a single block [83].

#### B.4 Task Hierarchy with Dynamic Task Prioritization

Our goal is to develop a joint multitask learning framework that prioritizes the difficult tasks, capable of competing with existing single state-of-the-art models. In this experiment we compare our performance to modern approaches for predicting classification, segmentation, detection, and pose estimation.

**Our Approach** For our model we used a task hierarchy with the the most difficult tasks ordered before easier tasks. The difficulty is determined by our formula,  $\mathcal{D}(t) = \text{FL}(\bar{\kappa}_t; \gamma_t)$ . The resulting task ordering prioritizes more difficult above easier tasks, is the following: classification, segmentation, detection, and pose estimation. Thus we use this ordering to construct our task hierarchy for this experiment.

We additionally apply our dynamic task prioritization to our model. For this we used a  $\gamma_1, \gamma_2, \gamma_3, \gamma_4 = 1.0$ . We did not apply any example-based focal loss to this model. We computed the exponential moving average using a span of 2.

**Implementations of Modern Single-Task Approaches** For the classification model we used a DenseNet-121 [79] architecture. For the segmentation model we used a fully convolutional network (FCN) [80], as detailed in Table 6. For the detection model we used a RetinaFPN50 model [39]. For pose estimation we used the stacked hourglass network architecture using eight stacks and one block [83].