Controllable Dynamic Multi-Task Architectures

Dripta S. Raychaudhuri[1] Yumin Suh[2] Samuel Schulter[2] Xiang Yu[2] Masoud Faraki[2]
Amit K. Roy-Chowdhury[1] Manmohan Chandraker[2,3]
[1]University of California, Riverside [2]NEC Labs America [3]University of California, San Diego

Abstract

Multi-task learning commonly encounters competition for resources among tasks, specifically when model capacity is limited. This challenge motivates models which allow control over the relative importance of tasks and total compute cost during inference time. In this work, we propose such a controllable multi-task network that dynamically adjusts its architecture and weights to match the desired task preference as well as the resource constraints. In contrast to the existing dynamic multi-task approaches that adjust only the weights within a fixed architecture, our approach affords the flexibility to dynamically control the total computational cost and match the user-preferred task importance better. We propose a disentangled training of two hypernetworks, by exploiting task affinity and a novel branching regularized loss, to take input preferences and accordingly predict tree-structured models with adapted weights. Experiments on three multi-task benchmarks, namely PASCAL-Context, NYU-v2, and CIFAR-100, show the efficacy of our approach. Project page is available at https://www.nec-labs.com/~mas/DYMU.

1. Introduction

Multi-task learning [7, 40] (MTL) solves multiple tasks using a single model, with potential advantages of fast inference and improved generalization by sharing representations across related tasks. However, in practical scenarios, simultaneously optimizing all tasks is difficult due to task conflicts and limited model capacity [54]. Consequently, a trade-off between the competing tasks has to be found, necessitating precise balancing of the different task losses during optimization. In many applications, the desired trade-off can change over time, requiring a new model to be retrained from scratch. To overcome this lack of flexibility, recent methods propose dynamic networks for multi-task learning [26, 36]. These frameworks enable a single multi-task model to learn the entire trade-off curve, and allow users to control the desired trade-off during inference via task preferences denoting the relative task importance.

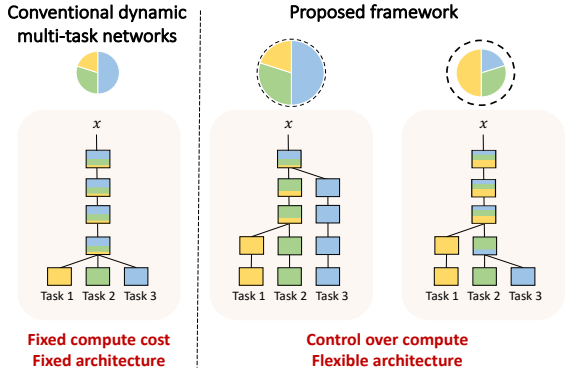

Figure 1. **Problem setup.** Our goal is to enable users to control resource allocation dynamically among multiple tasks at inference time. Conventional dynamic networks (PHN [36]) for MTL achieve this in terms of weight changes within a fixed model (color gradients indicate proportion of weights allocated for each task). In contrast, we perform resource allocation in terms of both architecture and weights. This enables us to control total compute cost in addition to task preference. Dashed circle represents maximum compute budget, while filled circle represents the desired budget. Portion of colors represents the user-defined task importance.

Conventional dynamic approaches for MTL assume a fixed model architecture, with all but the last prediction layers shared, and control trade-offs by changing the weights of this model. While such hard-parameter sharing is helpful in saving resources, the performance is inevitably lower than single task baselines when task conflicts exist due to over-sharing of parameters between tasks [40] . Furthermore, the fixed architecture suffers from a lack of flexibility, leading to a constant compute cost irrespective of the given task preference or compute budget changes. In many applications where the budget can change over time, these approaches may fail to take advantage of the increased resources in order to improve performance or accordingly lower the compute cost in order to satisfy stricter budget requirements.

To address the aforementioned issue and strike a balance between flexibility and performance, we propose a more expressive tree-structured [14] dynamic multi-task network which can adapt its *architecture* in addition to its weights at test-time, as illustrated in Figure 1. Specifically, we design a controller using two hypernetworks [16] that predict archi-

tectures and weights, respectively, given a user preference that specifies test-time trade-offs of relative *task importance* and *resource availability*. This increases flexibility by changing branching locations to re-allocate resources over tasks to match user-preferred task importance, and enhance or compromise task accuracy given computation budget requirements at any given moment. However, this comes at the cost of increase in complexity: 1) generalizing architecture prediction to unseen preferences, and 2) performing dynamic weight changes on potentially thousands of different models.

To tackle these challenges, we develop a two-stage training scheme that starts from an N-stream network, termed the anchor net, which is initialized using weights from N pre-trained single-task models. This guides the architecture search as a prior that is preference-agnostic yet captures *inter-task relations*. In the first stage, we exploit inter-task relations derived from the anchor net to train the first hypernetwork that predicts connections between the different streams. We introduce a branching regularized loss that encourages more *resource allocation* for dominant tasks while reducing the network cost from the less preferred ones. The predicted architectures contain edges that have not been observed during the anchor net initialization. These are denoted as cross-task edges since they connect nodes that belong to different streams. In the second stage, to improve the performance of the predicted architectures with cross-task edges, we train a secondary hypernetwork for *cross-task adaptation* via modulation of the normalization parameters.

Our framework is evaluated on three MTL datasets (PASCAL-Context, NYU-v2 and CIFAR-100) in terms of task performance, computational cost, and controllability (for both task importance and computational cost). Achieving performance comparable to state-of-the-art MTL architecture search methods under uniform task preference, our controller can further approximate efficient architectures for non-uniform preferences with provisions for reducing network size depending on computational constraints.

The primary contributions of our work are as follows:

- A controllable multi-task framework which allows users to assign task preference and the trade-off between task performance and network capacity via architectural changes.
- A controller, composed of two hypernetworks, to provide dynamic network structure and adapted network weights.
- A new joint learning objective including task-related losses and network complexity regularization to achieve the user defined trade-offs.
- Experiments on several MTL benchmarks (PASCAL-Context [35], NYU-v2 [45], CIFAR-100 [23]) demonstrate the efficacy of our framework.

2. Related Work

Multi-Task Learning. Multi-task learning seeks to learn a single model to simultaneously solve a variety of learning tasks by sharing information among the tasks [7]. In the context of deep learning, current works focus mostly on designing novel network architectures and constructing efficient shared representation among tasks [40, 55]. Typically, these works can be grouped into two classes - *hard-parameter sharing* and *soft-parameter sharing*. In the soft sharing setting [13, 34, 41], each task has its own set of backbone parameters with some sort of regularization mechanisms to enforce the distance between weights of the model to be close. In contrast, the hard sharing setting entails all the tasks sharing the same set of backbone parameters, with branches towards the outputs [21, 22, 31]. More recent works have attempted learning the optimal architectures via differentiable architecture search [5, 14, 46]. The overwhelming majority of these approaches are trained using a simple weighted sum of the individual task losses, where a proper set of weights is commonly selected using grid search or using techniques such as gradient balancing [8]. Other approaches [27, 33, 44] attempt to model multi-task learning as a multi-objective optimization problem and find Pareto stationary solutions among different tasks. Recently, optimization methods have also been proposed to manipulate gradients in order to avoid conflicts across tasks [9, 52]. None of these methods are suitable for dynamically modeling performance trade-offs, which is the focus of our work.

Hypernetworks. A hypernetwork is used to learn context dependent parameters for a dynamic network [16, 43], thus, obtaining multiple customizable models using a single network. Such hypernetworks have been successfully applied in different scenarios, *e.g.*, recurrent networks [16], 3D point cloud prediction [28], video frame prediction [20], neural architecture search [4] and reinforcement learning [38, 42]. Recent works [26, 36] propose using hypernetworks to model the Pareto front of competing multi-task objectives. Our approach is closely related to these works, however, these methods focus on generating weights for a fixed, handcrafted architecture, while we use hypernetworks to model the trade-offs in multi-task learning by varying the architecture. This allows us to take dynamic resource allocation into account, an aspect largely ignored in previous works.

Dynamic Networks. Dynamic neural networks, as opposed to usual static models, can adapt their structures during inference, leading to notable improvements in performance and computational efficiency [17]. Previous works focus on adjusting the network depth [3, 18, 48, 50], width [24, 53], or perform dynamic routing within a fixed supernet that includes multiple possible paths [25, 30, 37]. Dynamic depth is realised by either early exiting, *i.e.* allowing "easy" samples to be processed at shallow layers without executing the deeper layers [3, 18], or layer skipping, *i.e.* selectively skipping intermediate network layers conditioned on each sample [48, 50]. Dynamic width is an alternative to the dynamic depth where instead of layers, filters are selectively pruned

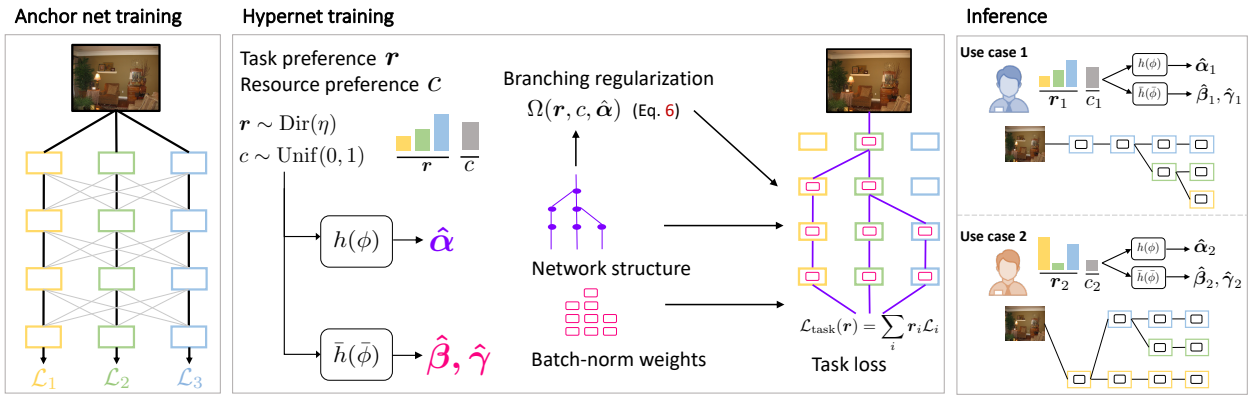

Figure 2. **Overview of framework.** We initialize our framework using an anchor net which consists of single-task networks and cross-task edges. During training, we first train the edge hypernet $h(\phi)$ using sampled preferences $(\boldsymbol{r}, c)$ to optimize the task loss and a branching regularizer, for preference aware branching. Next, we optimize the weight hypernet $\bar{h}(\bar{\phi})$ in a similar fashion by minimizing only the task loss. At inference, the hypernets jointly predict architecture and weights according to the user preferences.

conditioned on the input [24, 53]. Dynamic routing can be implemented by learning controllers to selectively execute one of multiple candidate modules at each layer [30, 37]. Due to the non-differentiable nature of the discrete choices, reinforcement learning is employed to learn these controllers. In [25], the routing modules utilize a differentiable activation function which conditionally outputs zero values, facilitating the end-to-end training of routing decisions. Recent works have also proposed learning dynamic weights for modeling different hyperparameter configurations [10] and domain adaptation [49]. In contrast to most of the existing works which intrinsically adapt network structures as a function of input, our method enables explicit control of the total computational cost as well as the task trade-offs.

Weight Sharing Neural Architecture Search. Weight sharing has evolved as a powerful tool to amortize computational cost across models for neural architecture search (NAS). These methods integrate the whole search space of architectures into a weight sharing supernet and optimize network architectures by pursuing the best performing sub-networks. Joint optimization methods [6, 29, 51] optimize the weights of the supernet and a differentiable routing policy simultaneously. In contrast, one-shot methods [1, 2, 4, 15] disentangle the training into two steps: first, the weights of the supernet are trained, after which the agent is trained with the fixed supernet. We utilize such a weight sharing strategy in our framework for dynamic resource allocation.

3. Method

Given a set of N tasks $\mathbf{T} = \{\mathcal{T}_1, \mathcal{T}_2, \ldots, \mathcal{T}_N\}$, conventional multi-task learning seeks to minimize a weighted sum of task-specific losses: $\mathcal{L}_{\text{task}}(\boldsymbol{r}) = \sum_i r_i \mathcal{L}_i$, where each $\mathcal{L}_i$ represents the loss associated with task $\mathcal{T}_i$, and $\boldsymbol{r}$ denotes a task preference vector. This vector signifies the desired performance trade-off across the different tasks, with larger values of r_i denoting higher importance to task $\mathcal{T}_i$. Here $\boldsymbol{r} \in \mathcal{S}_N$, where $\mathcal{S}_N = \{\boldsymbol{r} \in \mathbb{R}^N | \sum_i r_i = 1, r_i \geq 0\}$ represents the N-dimensional simplex [40]. We seek to approximate the trade-off curve defined by different values of $\boldsymbol{r}$ using tree-structured sub-networks [14] within a single multi-task model, given a total computational budget defined by a resource preference variable $c \in [0, 1]$, where larger c denotes more frugal resource usage. This is formulated as a minimization of the expected value of the task loss over the user preference distribution, with regularization Ω to control resource usage, *i.e.*, $\mathbb{E}_{(\boldsymbol{r},c)\sim P_{(\boldsymbol{r},c)}} \mathcal{L}_{\text{task}}(\boldsymbol{r}) + \Omega(\boldsymbol{r}, c)$. Optimizing this directly is equivalent to solving NAS [29] for every possible $(\boldsymbol{r}, c)$ simultaneously. Thus, instead of solving directly, we cast it as a search to find tree sub-structures and the corresponding modulation of features for every $(\boldsymbol{r}, c)$, within an N-stream anchor network with fixed weights.

Our framework consists of two *hypernets* (h and $\bar{h}$) [16] and an *anchor net* F, as shown in Figure 2. At test-time, given an input preference, we utilize the network connections and adapted weights predicted by the hypernets to modulate F, to obtain the final model. We propose a two-stage training scheme to train the framework. First, we initialize a preference agnostic anchor net, which provides the anchor weights at test time (Section 3.1). Based on this anchor net, the tree-structured architecture search space is then defined (Section 3.2). Next, we train the *edge hypernet* using prior task relations obtained from the anchor net by optimizing a novel branching regularized loss function derived by inducing a dichotomy over the tasks (Section 3.3.1). Finally, we train a *weight hypernet*, keeping the anchor net and edge hypernet fixed, to modulate the anchor net weights (Section 3.3.2).

3.1. Anchor Network

We introduce an anchor net F as an alternative approach to model weight generation in dynamic networks for MTL [26, 36]. Previous methods adopt chunking [16] to mitigate the large computation and memory required for gen-

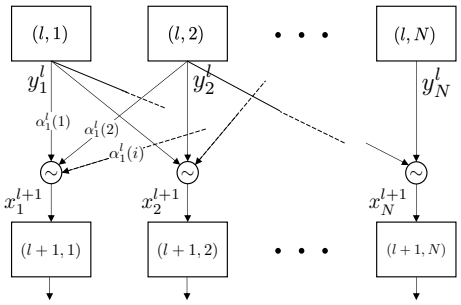

Figure 3. **Branching block.** Illustration of the parent sampling operation in Section 3.2. Nodes in layer l are sampled in accordance to a categorical distribution defined by α_j^l $\left(\sum_i \alpha_j^l(i) = 1\right)$ for each node $(l+1, j)$ in layer $l+1$.

erating entire network weights at the expense of limiting the hypernet capacity. The anchor net, consisting of N-stream backbones trained for N individual tasks (Figure 2), overcomes this bottleneck by providing the weights in the tree structures predicted by the edge hypernet. Our choice of the anchor net is motivated by the need for an initialization that reflects inter-task relations and is based on observations from [47], where branching in tree-structured MTL networks is shown to be contingent on how similar task features are at any layer. It can also be interpreted as a supernet used in one-shot NAS approaches [2], which is capable of emulating any architecture in the search space. Subsequently, the base weights of the anchor net are further modulated via the weight hypernet to address the cross-task connections unseen in the anchor net (Section 3.3.2).

3.2. Architecture Search Space

We utilize a tree-structured network topology which has been shown to be highly effective for multi-task learning in [14]. It shares common low-level features over tasks while extracting task-specific ones in the higher layers, enabling control of the trade-off between tasks by changing branching locations conditioned on the desired preference $(\boldsymbol{r}, c)$. The search space is represented as a *directed acyclic graph* (DAG), where vertices in the graph represent different operations and edges denote the data flow through the network. Figure 3 shows a block of such a graph, containing N *parent* and *child* nodes. In this work, we realize a tree-structure by stacking such blocks sequentially and allowing a child node to sample a path from the candidate paths between itself and all its parent nodes. Concretely, we formulate the stochastic branching operation at layer l as

$$x_j^{l+1} = d_j \cdot Y^l, \quad d_j \sim p_{\alpha_j^l}, \tag{1}$$

where x_j^{l+1} denotes the input to the j-th node in layer $l+1$, d_j is a one-hot vector indicating the parent node sampled from the *categorical* distribution parameterized by α_j^l and, $Y^l = [y_1^l, \ldots, y_N^l]$ concatenates outputs from all parent nodes at layer l. Note that selecting a parent from every node determines a unique tree structure. This suggests learning $\boldsymbol{\alpha} = \{\alpha_j^l\}_{0 \le j \le N, 0 \le l < L}$, conditioned on a preference $(\boldsymbol{r}, c)$, in a manner which satisfies the desired task trade-offs. Here, L denotes the total number of layers.

3.3. Preference Conditioned Hypernetworks

We use two hypernets [16] to construct our controller for architectural changes. The edge hypernet h, parameterized by ϕ, predicts the branching parameters $\hat{\boldsymbol{\alpha}} = h(\boldsymbol{r}, c; \phi)$ within the anchor net. Subsequently, the weight hypernet $\bar{h}$, parameterized by $\bar{\phi}$, predicts the normalization parameters $\{\hat{\boldsymbol{\beta}}, \hat{\boldsymbol{\gamma}}\} = \bar{h}(\boldsymbol{r}, c; \bar{\phi})$ to adapt the predicted network.

Optimizing the task loss $\mathcal{L}_{\text{task}}$ only takes into account the individual task performances without considering computational cost. Consequently, we introduce a branching regularizer $\Omega(\boldsymbol{r}, c, \hat{\boldsymbol{\alpha}})$ to encourage node sharing (or branching) based on the preference. This regularizer contains two terms, the *active* loss, which encourages limited sharing of features among the high preference tasks, and the *inactive* loss, which aims to reduce resource utilization for the less important ones. In particular, the active loss is additionally weighted by the cost preference c to enable the control of total computational cost. Formally, our objective is formulated as to find the controller (ϕ and $\bar{\phi}$) that minimizes the expectation of the branching regularized task loss over the distribution of user preferences $P_{(\boldsymbol{r},c)}$:

$$\min_{\phi, \bar{\phi}} \mathbb{E}_{(\boldsymbol{r},c)\sim P_{(\boldsymbol{r},c)}} \left[\mathcal{L}_{\text{task}}(\boldsymbol{r}, \hat{\boldsymbol{\alpha}}, \hat{\boldsymbol{\beta}}, \hat{\boldsymbol{\gamma}}) + \Omega(\boldsymbol{r}, c, \hat{\boldsymbol{\alpha}}) \right], \tag{2}$$

We disentangle the training of the hypernetworks for stability – the edge hypernet is trained first, followed by the weight hypernet. At test time, when a preference $(\boldsymbol{r}, c)$ is presented to the controller, the maximum likelihood architecture corresponding to the supplied preference is first sampled from the branching distribution parameterized by the predictions of h. The weights of this tree-structure are then inherited from the anchor net, supplemented via adapted normalization parameters predicted by $\bar{h}$.

3.3.1 Regularizing the Edge Hypernet

We illustrate the idea of branching regularization in Figure 4: tasks with higher preferences should have a greater influence on the branching structure while tasks with smaller preferences may be de-emphasized by encouraging them to follow existing branching choices. Specifically, we define two losses, active and inactive losses, based on the task division into two groups, active tasks $\mathcal{A} = \{\mathcal{T}_i \mid r_i \ge \tau, \ \forall i \in [N]\}$, and inactive tasks $\mathcal{I} = \{\mathcal{T}_i \mid r_i < \tau, \ \forall i \in [N]\}$ with some threshold τ. Although individual tasks are already weighted by $\boldsymbol{r}$ in task loss $\mathcal{L}_{\text{task}}$, this explicit emphasizing of certain tasks over others was found to be crucial to induce better

controllability, as shown in Section 4.6.

Active loss. The active loss $\mathcal{L}_{\text{active}}$ encourages nodes in the anchor net, corresponding to the active tasks, to be shared in order to avoid the whole network being split up by tasks with little knowledge shared among them. Specifically, we encourage any pair of nodes that are likely to be sampled in the final architecture (P) and are from two similar tasks (A) to take the same parent node. Formally, we define $\mathcal{L}_{\text{active}}$ as,

$$\mathcal{L}_{\text{active}} = \sum_{l=1}^{L} \sum_{\substack{i,j \in \mathcal{A} \\ i \neq j}} \frac{L-l}{L} \cdot A(i,j) \cdot P(l,i,j) \cdot \|\nu_i^l - \nu_j^l\|^2, \quad (3)$$

where $P(l,i,j) = P_{\text{use}}(l,i) \cdot P_{\text{use}}(l,j)$. $P_{use}(l,i) = 1 - \prod_k \{1 - P_{use}(l+1,k) \cdot \nu_k^l(i)\}$ denotes the probability that the nodes i in layer l are used in the sampled tree structure. $A(i,j)$ captures the *task affinity* between tasks $\mathcal{T}_i$ and $\mathcal{T}_j$, where we adopt *Representational Similarity Analysis* (RSA) [11] to compute the affinity. The factor $\frac{L-l}{L}$ encourages more sharing of nodes which contain low-level features. The full derivations of P_{use} and A are detailed in the supplementary document.

We use the Gumbel-Softmax reparameterization trick [19] to obtain the samples ν_i^l from the predicted logits $\hat{\boldsymbol{\alpha}}$,

$$\nu_i^l(k) = \frac{\exp\left((\log \alpha_i^l(k) + G_i^l(k))/\zeta\right)}{\sum_{m=1}^{N} \exp\left((\log \alpha_i^l(m) + G_i^l(m))/\zeta\right)}. \quad (4)$$

Here, $G_i^l = -\log(-\log U_i^l)$ is a standard Gumbel distribution with U_i^l sampled i.i.d. from the uniform distribution $\text{Unif}(0,1)$, and ζ denotes the temperature of the softmax.

Inactive loss. The inactive tasks should have minimal effect in terms of branching. Inactive loss, $\mathcal{L}_{\text{inactive}}$, encourages these tasks to mimic the most closely related branching pattern,

$$\mathcal{L}_{\text{inactive}} = \sum_{l=1}^{L} \sum_{j \in \mathcal{I}} \min_{i \in \mathcal{A}} \|\nu_i^l - \nu_j^l\|^2. \quad (5)$$

This ensures that the network branching is controlled by the active tasks, with the inactive tasks sharing nodes with the active tasks.

Thus, the branching regularizer is defined as follows,

$$\Omega(\boldsymbol{r}, c, \hat{\boldsymbol{\alpha}}) = c \cdot \lambda_{\mathcal{A}} \mathcal{L}_{\text{active}} + \lambda_{\mathcal{I}} \mathcal{L}_{\text{inactive}}, \quad (6)$$

where $\lambda_{\mathcal{A}}$, $\lambda_{\mathcal{I}}$ are hyperparameters to determine the weighting of the losses. Typically, we set $\lambda_{\mathcal{A}} = 1$ and $\lambda_{\mathcal{I}} = 0.1$. Here, the active loss is additionally weighted by the resource preference c, so that larger c encourages more feature sharing to reduce total computational cost.

3.3.2 Cross-task Adaptation

The architecture sampled by the edge hypernet h contains edges that have not been observed during the anchor net training. These are denoted as *cross-task* edges since they connect nodes that belong to different streams in F. Consequently, the performance of the sampled network is sub-optimal. To rectify this issue, we propose to modulate the weights of the anchor net to adaptively update the unseen edges using an additional weight hypernet $\bar{h}$. Inspired from the prior works [32, 49] that estimate normalization statistics and optimize channel-wise affine transformations, we modulate only the normalization parameters using a hypernetwork. Concretely, we modulate the original batch normalization operation at layer l, $\text{BN}_i^l(\boldsymbol{x}_i^l) = \gamma_i^l \frac{x_i^l - \mu_i^l}{\sigma_i^l} + \beta_i^l$, to $\text{BN}_i^l(\boldsymbol{x}_i^l) = (\gamma_i^l + \Delta\boldsymbol{\gamma_i^l}) \frac{x_i^l - \mu_i^l}{\sigma_i^l} + (\beta_i^l + \Delta\boldsymbol{\beta_i^l})$ by predicting the perturbations to the parameters: $\{\Delta\beta_i^l, \Delta\gamma_i^l\}_{0 \leq i \leq N, 0 \leq l < L} = \bar{h}(r, c; \theta)$, where γ_i^l and β_i^l are the original affine parameters, and μ_i^l and σ_i^l denote the batch statistics of the node input x_i^l. This modulation primarily affects the preferences with two or more dominant tasks, where cross-task connections occur.

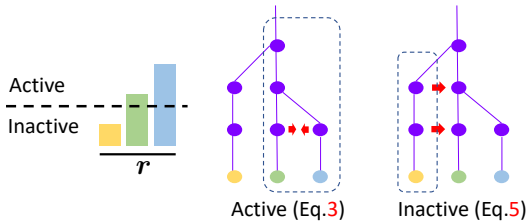

Figure 4. **Branching loss.** Illustration of the branching regularization, consisting of active and inactive losses. The active loss encourages limited sharing between high importance tasks, while the inactive loss tries to limit branching for less preferred tasks as much as possible.

4. Experiments

In this section, we demonstrate the ability of our framework to dynamically search for efficient architectures for multi-task learning. We show that our framework achieves flexibility between two extremes of the accuracy-efficiency trade-off, allowing a better control within a single model. Extensive experiments indicate that the predicted network structures match well with the input preferences, in terms of both resource usage and task performance.

4.1. Evaluation Criteria

Uniformity. To measure controllability with respect to task preferences, we utilize uniformity [33] which quantifies how well the vector of task losses $\boldsymbol{L} = [\mathcal{L}_1, \ldots, \mathcal{L}_N]$ is aligned with the given preference. Specifically, for the loss vector $\boldsymbol{L}$ corresponding to the architecture for task preference $\boldsymbol{r}$, uniformity is defined as $\mu_{\boldsymbol{r}} = 1 - D_{KL}(\hat{\boldsymbol{L}} \parallel \boldsymbol{1}/N)$, where $\hat{\boldsymbol{L}}(j) = \frac{r_j \mathcal{L}_j}{\sum_i r_i \mathcal{L}_i}$. This arises from the fact that, ideally, $r_j \propto 1/\mathcal{L}_j$, which in turn implies $r_1\mathcal{L}_1 = r_2\mathcal{L}_2 \cdots = r_N\mathcal{L}_N$.

Hypervolume. Using the trained controller, we are able to approximate the trade-off curve among the different tasks in the loss space. To evaluate the quality of this curve, we

utilize hypervolume (HV) [56] – a popular metric in the multi-objective optimization literature to compare different sets of solutions approximating the Pareto front [12]. It measures the volume in the loss space of points dominated by a solution in the evaluated set. Since this volume is unbounded, hypervolume measures the volume in a rectangle defined by the solutions and a selected *reference point*. More details can be found in the supplementary.

Computational Resource. We measure the computational cost using the memory of the activated nodes in the anchor net and the GFLOPs, which approximates the time spent in the forward pass. We also report the computational cost of the hypernets to take into account their overheads. We discuss more on the model size in the supplementary document.

4.2. Datasets

We evaluate the performance of our approach using three multi-task datasets, namely **PASCAL-Context** [35] and **NYU-v2** [45], and **CIFAR-100** [23]. The PASCAL-Context dataset is used for joint semantic segmentation, human parts segmentation and saliency estimation, as well as these three tasks together with surface normal estimation, and edge detection as in [5]. The NYU-v2 dataset comprises images of indoor scenes, fully labeled for semantic segmentation, depth estimation and surface normal estimation. For CIFAR-100, we split the dataset into 20 five-way classification tasks [39].

4.3. Baselines

We compare our framework with both *static* and *dynamic* networks. Static networks include **Single-task** networks, where we train each task separately using a task-specific backbone, and **Multi-task** networks, in which all tasks share the backbone but have separate task-specific heads at the end. These multi-task networks are trained separately for different preferences and thus, training time scales linearly with the number of preferences. We use this to contrast the training time of our framework. The single-task networks demonstrate the anchor net performance. We also compare our architectures with two multi-task NAS methods, **LTB** [25] and **BMTAS** [5], which use the same tree-structured search space to perform NAS, but are static. The dynamic networks include Pareto Hypernetworks (**PHN**) [36], which predicts only the weights of a shared backbone network conditioned on a task preference vector using hypernetworks, and **PHN-BN**, a variation of PHN which predicts only the normalization parameters similar to our weight hypernet. Implementation details are presented in the supplementary.

4.4. Comparison with Baselines

Controllable resource usage. We visualize the variation in computational cost with respect to different task and resource usage preferences in Figure 5. We adopt the ratio of the size of the predicted architecture to the size of the total anchor net as the criterion for evaluating computational cost. Compared to conventional dynamic networks that only adjust weights with a fixed computational cost (right), our framework (left) enables control over the total cost via a cost preference c. Resource usage peaks at the center of the contour, when more tasks are active, and falls down gradually as we move towards the corners, where task preferences are heavily skewed. Furthermore, the average resource usage decreases as c is increased, indicating the ability of the controller to incorporate resource constraints.

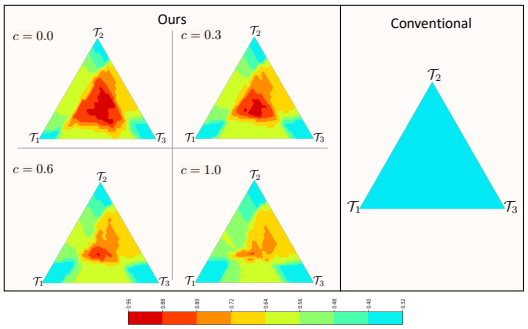

Figure 5. **Resource usage on NYU-v2.** We visualize resource usage by plotting the proportion of parameters active in the anchor net versus the task preference. The three vertices represent the task preferences with non-zero importance to only one task, while areas in the middle correspond to more dense preferences. As c increases, the predicted networks grow progressively smaller in the dense regions. On the other hand, conventional dynamic networks for MTL always have a constant resource usage ($\mathcal{T}_1$:semantic seg., $\mathcal{T}_2$:surface normal, $\mathcal{T}_3$:depth).

Multi-task performance. We demonstrate the overall multi-task performance in Tables 1-4 on four different settings (PASCAL-Context 5-task, PASCAL-Context 3-task, NYU-v2 3-task, CIFAR-100 20-task). In all cases, we report hypervolume (reference point mentioned below heading) and uniformity averaged across 20 task preference vectors $\boldsymbol{r}$, sampled uniformly from $\mathcal{S}_N$. Inference network cost is calculated similarly over 1000 preference vectors. These are shown for two choices of $c \in \{0, 1\}$ to highlight the two extreme cases of resource usage.

Our framework achieves higher values in both hypervolume and uniformity compared to the existing dynamic models (PHN and PHN-BN) in all four settings. While the high hypervolume reinforces the efficacy of tree-structured models in solving multi-task problems, the uniformity values consolidate architectural change as an effective approach towards modeling task trade-offs. This is accompanied by increased average computational cost, indicated by inference parameter count. As discussed above, this is due to the flexible architecture over preferences, where actual cost will differ for each preference, *e.g.*, reaching the cost of PHN-BN for extremely skewed preferences (Figure 5). Compared to Single-Task, the proposed controller is able to find effective architectures (as indicated by the hypervolume) which

Method	HV.↑ [3, 3, . . . , 3]	Unif.↑	Inference Params.↓	GFLOPs↓	Control Params.
Single-Task	81.56	-	9.84M	16.17	-
PHN	42.61	0.72	2.15M	6.28	21.50M
PHN-BN	72.27	0.69	2.15M	6.28	3.63M
Ours w/o adaptation, c=0.0	47.73	0.84	3.34M	7.21	0.06M
Ours w/o adaptation, c=1.0	30.91	**0.86**	2.75M	6.81	0.06M
Ours, c=0.0	**75.52**	0.76	3.34M	7.21	15.32M
Ours, c=1.0	73.20	0.79	2.75M	6.81	15.32M

Table 1. **Evaluation on PASCAL-Context (5 tasks).**

Method	HV.↑ [3, 3, 3]	Unif.↑	Inference Params.↓	GFLOPs↓	Control Params.
Single-Task	4.31	-	5.91M	9.75	-
PHN	1.97	0.74	2.06M	4.81	21.10M
PHN-BN	3.92	0.79	2.06M	4.81	3.32M
Ours w/o adaptation, c=0.0	3.56	**0.92**	3.15M	5.52	0.03M
Ours w/o adaptation, c=1.0	3.35	0.91	2.86M	5.07	0.03M
Ours, c=0.0	**4.26**	0.82	3.15M	5.52	9.25M
Ours, c=1.0	4.25	0.82	2.86M	5.07	9.25M

Table 2. **Evaluation on PASCAL-Context (3 tasks).**

Method	HV.↑ [4, 4, 4]	Unif.↑	Inference Params.↓	GFLOPs↓	Control Params.
Single-Task	12.83	-	64.47M	58.78	-
PHN	2.36	0.75	21.59M	21.02	21.04M
PHN-BN	11.72	0.73	21.59M	21.02	2.23M
Ours w/o adaptation, c=0.0	12.42	0.82	41.06M	29.04	0.03M
Ours w/o adaptation, c=1.0	9.53	**0.84**	34.68M	25.98	0.03M
Ours, c=0.0	**13.43**	0.76	41.06M	29.04	5.72M
Ours, c=1.0	13.08	0.78	34.68M	25.98	5.72M

Table 3. **Evaluation on NYU-v2**.

Method	HV.↑ [1, 1, . . . , 1]	Unif.↑	Inference Params.↓	GFLOPs↓	Control Params.
Single-Task	0.009	-	36.18M	348.79	-
PHN	0.002	0.54	16.35M	73.13	11.03M
PHN-BN	0.007	0.49	16.35M	73.13	0.31M
Ours w/o adaptation, c=0.0	0.003	**0.58**	31.86M	174.36	0.34M
Ours w/o adaptation, c=1.0	0.001	0.53	31.37M	129.23	0.34M
Ours, c=0.0	**0.010**	0.54	31.86M	174.36	3.10M
Ours, c=1.0	0.009	0.49	31.37M	129.23	3.10M

Table 4. **Evaluation on CIFAR-100**.

perform nearly at par with a smaller memory footprint (as indicated by the average inference network parameter count). Notably, in the NYU-v2 3-task and CIFAR-100 settings, the ability to find effective architectures enables the model to outperform single-task networks, demonstrating the benefit of sharing features among related tasks via architectural change. In addition, our framework enjoys flexibility between two extreme cases, *i.e*. Single-Task (highest accuracy with lowest inference efficiency) and dynamic models with shared backbone (lowest accuracy with highest inference efficiency), spanning a range of trade-offs for different c values. The range of HV is larger when task-specific features are useful, compared to when the compact architecture already achieves higher HV than the Single-Task (Tables 3,4). "Control Params." is the cost of the hypernets. Note that this overhead will materialize only when the preference changes and does not have any effect on the task inference time.

Effect of cross-task adaptation In Tables 1-4, "Ours w/o adaptation" denotes the model without weight hypernet. As indicated by larger HVs, cross-task adaptation improves the performance without affecting the inference time. A trend

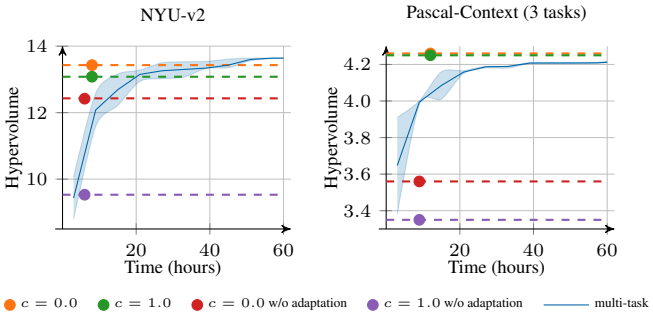

Figure 6. **Comparison with preference-specific multi-task networks.** For static multi-task models, each value is computed by evaluating a subset of preferences, with the shaded area marking the variance across selected subsets. Our framework achieves high hypervolume significantly faster with a single model.

that persists across all the settings is the slight drop in uniformity that accompanies the adapted models in comparison to the unadapted ones. This is due to the propensity of the weight hypernet to improve task performance as much as possible while keeping the preferences intact. This leads to improved performance even in the low-priority tasks at the expense of lower uniformity. Note that our primary factor of controllability is through architectural changes which remains unaffected by the weight hypernet.

Training efficiency. In contrast to dynamic networks, static multi-task networks require multiple models to be trained, corresponding to different task preferences, to approximate the trade-off curve. As a result, these methods have a clear trade-off between their performance and their training time. To analyze this trade-off, we plot hypervolume vs. training time for our framework when compared to training multiple static models in Figure 6. We trained 20 multi-task models with different preferences sampled uniformly, and at the inference time we selected subsets of various sizes and computed their hypervolume. The shaded area in Figure 6 reflects the variance over different selections of task preference subsets. This empirically shows that our approach requires shorter training time to achieve similar hypervolume compared to the static multi-task networks.

4.5. Analysis

Architecture evaluation. We study the effectiveness of the architectures predicted by the edge hypernet by comparing them with those predicted by LTB [14] and BMTAS [5]. We choose the architecture predicted for a uniform task preference and, similar to LTB, we retrain it for a fair comparison. We evaluate the performance in terms of relative drop in performance across tasks and number of parameters with respect to the single task baseline. Despite not being directly trained for NAS, our framework is able to output architectures which perform at par with LTB (Table 5). More results on the Pascal-Context dataset and visualization of dynamic architectures can be found in the supplementary document.

Method	$\mathcal{T}_1 \uparrow$	$\mathcal{T}_2 \uparrow$	$\mathcal{T}_3 \uparrow$	Avg $\Delta_{\mathbf{T}}(\%) \uparrow$	# Params (%) $\downarrow$
Single-Task	64.11	58.41	65.17	-	-
LTB	61.84	59.41	64.18	-1.12	-35.0
BMTAS	62.79	58.41	64.74	-0.93	**-48.9**
Ours, c=0.0	63.60	59.41	64.94	**+0.18**	-35.2
Ours†, c=0.0	62.34	58.60	65.17	-0.81	-35.2
Ours, c=1.0	63.12	58.93	64.93	-0.34	-40.8
Ours†, c=1.0	61.91	58.71	65.01	-1.05	-40.8

Table 5. **Architecture evaluation on PASCAL-Context (3 tasks).** We report the mean intersection over union for $\mathcal{T}_1$: Semantic seg., $\mathcal{T}_2$: Parts seg., and $\mathcal{T}_3$: Saliency. Presence of † indicates that we train the networks initialized from ImageNet weights, while its absence indicates training from anchor net weights.

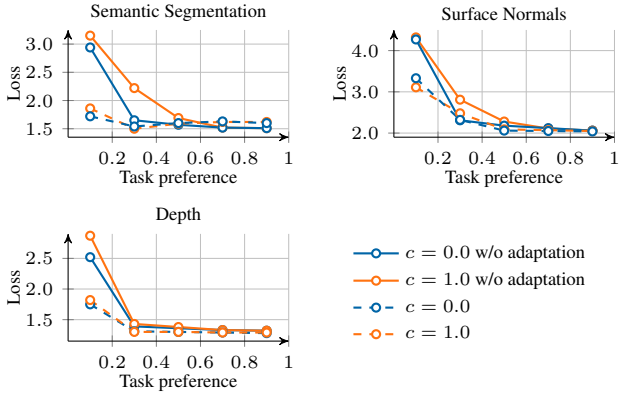

Figure 7. **Marginal evaluation tasks on NYU-v2**

Task controllability. In Figure 7 we visualize the task controllability for our framework by plotting the test loss at different values of task preference for that specific task, marginalized over preference values of the other tasks. As expected, increasing the preference for a task gradually leads to a decrease in the loss value. Furthermore, increasing c leads to higher loss values due to smaller predicted architectures. The effect of the weight hypernet is also evident, as shown by the lower loss values obtained on using it on top of the edge hypernet (w/o adaptation).

4.6. Ablation Study

Impact of inactive loss. Removing $\mathcal{L}_{\text{inact}}$ leads to loss of controllability with the edge hypernet predominantly predicting the full original anchor net, with minimal branching, leading to high resource usage and poor uniformity (Table 6).

Impact of weighting factors. Removing the two branching weights, $\frac{L-l}{L}$ and A, in the active loss, we make three key observations in Table 6: 1) average resource usage increases, 2) uniformity drops due to poor alignment between architectures and preferences, with larger architectures incorrectly predicted for skewed preferences, which ideally require less resources, 3) hypervolume remains almost constant across different c indicating poor cost control. Resource usage plots are presented in the supplementary.

Analysis of task threshold. We compare the effect of varying the threshold τ in Figure 8. Increasing the value beyond $1/N$ (~ 0.3) leads to loss of controllability as indicated by the constant hypervolume across different values of c. This is due to the inability to account for uniform preferences. On the other hand, choosing values below this threshold leads to comparable performance. Additional explanations are provided in the supplementary.

Method	HV.↑		Unif.↑		#Inference Params↓	
	$c = 0.0$	$c = 1.0$	$c = 0.0$	$c = 1.0$	$c = 0.0$	$c = 1.0$
Ours	13.43	13.08	0.76	0.78	41.06M	34.68M
no $\mathcal{L}_{\text{inactive}}$	12.81	12.73	0.49	0.51	61.75M	54.78M
no layer weighting	12.69	12.21	0.51	0.53	46.21M	41.33M
no task affinity A	12.53	12.35	0.51	0.52	45.73M	42.55M
no task dichotomy	12.57	11.20	0.60	0.63	56.73M	45.17M

Table 6. **Ablation study on NYU-v2**

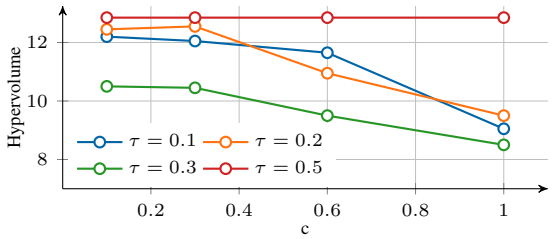

Figure 8. **Varying thresholds on NYU-v2**

Task classification. We analyse the importance of the induced task dichotomy by considering all tasks as active. This leads to: 1) high overall resource usage, and 2) poor controllability, especially at low values of c, as shown in Table 6. Resource usage plots are presented in the supplementary.

5. Conclusion

We present a new framework for dynamic resource allocation in multi-task networks. We design a controller using hypernets to dynamically predict both network architecture and weights to match user-defined task trade-offs and resource constraints. In contrast to current dynamic MTL methods which work with a fixed model, our formulation allows the flexibility in controlling the total compute cost and matches the task preference better. We show the effectiveness of our approach on four multi-task settings, attaining diverse and efficient architectures across a wide range of preferences.

Limitations and future work. Our framework searches solely over network width and thus, the compute cost is lower bounded by network depth. One possible solution is to extend the search space to allow skip connections within and across streams to allow variable depth. Also, scalability could be an issue as the required memory for the anchor net is proportional to the number of tasks. Our future work will address these issues by reducing the dependency on the anchor net initialization.

Acknowledgements. This work was a part of Dripta S. Raychaudhuri's internship at NEC Labs America. This work was also partially supported by the NRI grant 2021-67022-33453 and the NSF grant 1724341.

References

[1] Youhei Akimoto, Shinichi Shirakawa, Nozomu Yoshinari, Kento Uchida, Shota Saito, and Kouhei Nishida. Adaptive stochastic natural gradient method for one-shot neural architecture search. In *International Conference on Machine Learning*, pages 171–180. PMLR, 2019. 3

[2] Gabriel Bender, Pieter-Jan Kindermans, Barret Zoph, Vijay Vasudevan, and Quoc Le. Understanding and simplifying one-shot architecture search. In *International Conference on Machine Learning*, pages 550–559. PMLR, 2018. 3, 4

[3] Tolga Bolukbasi, Joseph Wang, Ofer Dekel, and Venkatesh Saligrama. Adaptive neural networks for efficient inference. In *International Conference on Machine Learning*, pages 527–536. PMLR, 2017. 2

[4] Andrew Brock, Theodore Lim, James M Ritchie, and Nick Weston. Smash: one-shot model architecture search through hypernetworks. *arXiv preprint arXiv:1708.05344*, 2017. 2, 3

[5] David Bruggemann, Menelaos Kanakis, Stamatios Georgoulis, and Luc Van Gool. Automated search for resource-efficient branched multi-task networks. *arXiv preprint arXiv:2008.10292*, 2020. 2, 6, 7

[6] Han Cai, Ligeng Zhu, and Song Han. Proxylessnas: Direct neural architecture search on target task and hardware. *arXiv preprint arXiv:1812.00332*, 2018. 3

[7] Rich Caruana. Multitask learning. *Machine learning*, 28(1):41–75, 1997. 1, 2

[8] Zhao Chen, Vijay Badrinarayanan, Chen-Yu Lee, and Andrew Rabinovich. Gradnorm: Gradient normalization for adaptive loss balancing in deep multitask networks. In *International Conference on Machine Learning*, pages 794–803. PMLR, 2018. 2

[9] Zhao Chen, Jiquan Ngiam, Yanping Huang, Thang Luong, Henrik Kretzschmar, Yuning Chai, and Dragomir Anguelov. Just pick a sign: Optimizing deep multitask models with gradient sign dropout. In *Advances in Neural Information Processing Systems*, 2020. 2

[10] Alexey Dosovitskiy and Josip Djolonga. You only train once: Loss-conditional training of deep networks. In *International Conference on Learning Representations*, 2019. 3

[11] Kshitij Dwivedi and Gemma Roig. Representation similarity analysis for efficient task taxonomy & transfer learning. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 12387–12396, 2019. 5

[12] Mark Fleischer. The measure of pareto optima applications to multi-objective metaheuristics. In *International Conference on Evolutionary Multi-Criterion Optimization*, pages 519–533. Springer, 2003. 6

[13] Yuan Gao, Jiayi Ma, Mingbo Zhao, Wei Liu, and Alan L Yuille. Nddr-cnn: Layerwise feature fusing in multi-task cnns by neural discriminative dimensionality reduction. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 3205–3214, 2019. 2

[14] Pengsheng Guo, Chen-Yu Lee, and Daniel Ulbricht. Learning to branch for multi-task learning. In *International Conference on Machine Learning*, 2020. 1, 2, 3, 4, 7

[15] Zichao Guo, Xiangyu Zhang, Haoyuan Mu, Wen Heng, Zechun Liu, Yichen Wei, and Jian Sun. Single path one-shot neural architecture search with uniform sampling. In *Proceedings of the European Conference on Computer Vision*, pages 544–560. Springer, 2020. 3

[16] David Ha, Andrew Dai, and Quoc V Le. Hypernetworks. *arXiv preprint arXiv:1609.09106*, 2016. 1, 2, 3, 4

[17] Yizeng Han, Gao Huang, Shiji Song, Le Yang, Honghui Wang, and Yulin Wang. Dynamic neural networks: A survey. *arXiv preprint arXiv:2102.04906*, 2021. 2

[18] Gao Huang, Danlu Chen, Tianhong Li, Felix Wu, Laurens van der Maaten, and Kilian Q Weinberger. Multi-scale dense networks for resource efficient image classification. *arXiv preprint arXiv:1703.09844*, 2017. 2

[19] Eric Jang, Shixiang Gu, and Ben Poole. Categorical reparameterization with gumbel-softmax. *arXiv preprint arXiv:1611.01144*, 2016. 5

[20] Xu Jia, Bert De Brabandere, Tinne Tuytelaars, and Luc V Gool. Dynamic filter networks. *Advances in Neural Information Processing Systems*, 29:667–675, 2016. 2

[21] Alex Kendall, Yarin Gal, and Roberto Cipolla. Multi-task learning using uncertainty to weigh losses for scene geometry and semantics. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 7482–7491, 2018. 2

[22] Iasonas Kokkinos. Ubernet: Training a universal convolutional neural network for low-, mid-, and high-level vision using diverse datasets and limited memory. In *Proceedings of the IEE/CVFE Conference on Computer Vision and Pattern Recognition*, pages 6129–6138, 2017. 2

[23] Alex Krizhevsky, Geoffrey Hinton, et al. Learning multiple layers of features from tiny images. 2009. 2, 6

[24] Changlin Li, Guangrun Wang, Bing Wang, Xiaodan Liang, Zhihui Li, and Xiaojun Chang. Dynamic slimmable network. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 8607–8617, 2021. 2, 3

[25] Yanwei Li, Lin Song, Yukang Chen, Zeming Li, Xiangyu Zhang, Xingang Wang, and Jian Sun. Learning dynamic routing for semantic segmentation. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 8553–8562, 2020. 2, 3, 6

[26] Xi Lin, Zhiyuan Yang, Qingfu Zhang, and Sam Kwong. Controllable pareto multi-task learning. *arXiv preprint arXiv:2010.06313*, 2020. 1, 2, 3

[27] Xi Lin, Hui-Ling Zhen, Zhenhua Li, Qing-Fu Zhang, and Sam Kwong. Pareto multi-task learning. *Advances in Neural Information Processing Systems*, 32:12060–12070, 2019. 2

[28] Gidi Littwin and Lior Wolf. Deep meta functionals for shape representation. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 1824–1833, 2019. 2

[29] Hanxiao Liu, Karen Simonyan, and Yiming Yang. Darts: Differentiable architecture search. In *International Conference on Learning Representations*, 2019. 3

[30] Lanlan Liu and Jia Deng. Dynamic deep neural networks: Optimizing accuracy-efficiency trade-offs by selective execution. In *Proceedings of the AAAI Conference on Artificial Intelligence*, 2018. 2, 3

[31] Mingsheng Long, Zhangjie Cao, Jianmin Wang, and Philip S Yu. Learning multiple tasks with multilinear relationship networks. *arXiv preprint arXiv:1506.02117*, 2015. 2

[32] Rabeeh Karimi Mahabadi, Sebastian Ruder, Mostafa Dehghani, and James Henderson. Parameter-efficient multi-task fine-tuning for transformers via shared hypernetworks. *arXiv preprint arXiv:2106.04489*, 2021. 5

[33] Debabrata Mahapatra and Vaibhav Rajan. Multi-task learning with user preferences: Gradient descent with controlled ascent in pareto optimization. In *International Conference on Machine Learning*, pages 6597–6607. PMLR, 2020. 2, 5

[34] Ishan Misra, Abhinav Shrivastava, Abhinav Gupta, and Martial Hebert. Cross-stitch networks for multi-task learning. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 3994–4003, 2016. 2

[35] Roozbeh Mottaghi, Xianjie Chen, Xiaobai Liu, Nam-Gyu Cho, Seong-Whan Lee, Sanja Fidler, Raquel Urtasun, and Alan Yuille. The role of context for object detection and semantic segmentation in the wild. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2014. 2, 6

[36] Aviv Navon, Aviv Shamsian, Gal Chechik, and Ethan Fetaya. Learning the pareto front with hypernetworks. In *International Conference on Learning Representations*, 2021. 1, 2, 3, 6

[37] Augustus Odena, Dieterich Lawson, and Christopher Olah. Changing model behavior at test-time using reinforcement learning. *arXiv preprint arXiv:1702.07780*, 2017. 2, 3

[38] Tabish Rashid, Mikayel Samvelyan, Christian Schroeder, Gregory Farquhar, Jakob Foerster, and Shimon Whiteson. Qmix: Monotonic value function factorisation for deep multi-agent reinforcement learning. In *International Conference on Machine Learning*, pages 4295–4304. PMLR, 2018. 2

[39] Clemens Rosenbaum, Tim Klinger, and Matthew Riemer. Routing networks: Adaptive selection of non-linear functions for multi-task learning. *arXiv preprint arXiv:1711.01239*, 2017. 6

[40] Sebastian Ruder. An overview of multi-task learning in deep neural networks. *arXiv preprint arXiv:1706.05098*, 2017. 1, 2, 3

[41] Sebastian Ruder, Joachim Bingel, Isabelle Augenstein, and Anders Søgaard. Latent multi-task architecture learning. In *Proceedings of the AAAI Conference on Artificial Intelligence*, 2019. 2

[42] Elad Sarafian, Shai Keynan, and Sarit Kraus. Recomposing the reinforcement learning building blocks with hypernetworks. In *International Conference on Machine Learning*, pages 9301–9312. PMLR, 2021. 2

[43] Jürgen Schmidhuber. Learning to control fast-weight memories: An alternative to dynamic recurrent networks. *Neural Computation*, 4(1):131–139, 1992. 2

[44] Ozan Sener and Vladlen Koltun. Multi-task learning as multi-objective optimization. In *Advances in Neural Information Processing Systems*, 2018. 2

[45] Nathan Silberman, Derek Hoiem, Pushmeet Kohli, and Rob Fergus. Indoor segmentation and support inference from rgbd images. In *Proceedings of the European Conference on Computer Vision*, 2012. 2, 6

[46] Ximeng Sun, Rameswar Panda, Rogerio Feris, and Kate Saenko. Adashare: Learning what to share for efficient deep multi-task learning. *arXiv preprint arXiv:1911.12423*, 2019. 2

[47] Simon Vandenhende, Stamatios Georgoulis, Bert De Brabandere, and Luc Van Gool. Branched multi-task networks: deciding what layers to share. *arXiv preprint arXiv:1904.02920*, 2019. 4

[48] Andreas Veit and Serge Belongie. Convolutional networks with adaptive inference graphs. In *Proceedings of the European Conference on Computer Vision*, pages 3–18, 2018. 2

[49] Dequan Wang, Evan Shelhamer, Shaoteng Liu, Bruno Olshausen, and Trevor Darrell. Tent: Fully test-time adaptation by entropy minimization. *arXiv preprint arXiv:2006.10726*, 2020. 3, 5

[50] Xin Wang, Fisher Yu, Zi-Yi Dou, Trevor Darrell, and Joseph E Gonzalez. Skipnet: Learning dynamic routing in convolutional networks. In *Proceedings of the European Conference on Computer Vision*, pages 409–424, 2018. 2

[51] Bichen Wu, Xiaoliang Dai, Peizhao Zhang, Yanghan Wang, Fei Sun, Yiming Wu, Yuandong Tian, Peter Vajda, Yangqing Jia, and Kurt Keutzer. Fbnet: Hardware-aware efficient convnet design via differentiable neural architecture search. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 10734–10742, 2019. 3

[52] Tianhe Yu, Saurabh Kumar, Abhishek Gupta, Sergey Levine, Karol Hausman, and Chelsea Finn. Gradient surgery for multi-task learning. In *Advances in Neural Information Processing Systems*, 2020. 2

[53] Zhihang Yuan, Bingzhe Wu, Guangyu Sun, Zheng Liang, Shiwan Zhao, and Weichen Bi. S2dnas: Transforming static cnn model for dynamic inference via neural architecture search. In *Proceedings of the European Conference on Computer Vision*, 2020. 2, 3

[54] Amir R Zamir, Alexander Sax, , William B Shen, Leonidas Guibas, Jitendra Malik, and Silvio Savarese. Taskonomy: Disentangling task transfer learning. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2018. 1

[55] Yu Zhang and Qiang Yang. A survey on multi-task learning. *IEEE Transactions on Knowledge and Data Engineering*, 2021. 2

[56] Eckart Zitzler and Lothar Thiele. Multiobjective evolutionary algorithms: a comparative case study and the strength pareto approach. *IEEE transactions on Evolutionary Computation*, 3(4):257–271, 1999. 6