Efficient Controllable Multi-Task Architectures

Abhishek Aich[⋆,†], Samuel Schulter[†], Amit K. Roy-Chowdhury[⋆],
Manmohan Chandraker[†,‡], Yumin Suh[†]

[⋆]University of California, Riverside, [†]NEC Labs America, [‡]University of California, San Diego

Abstract

We aim to train a multi-task model such that users can adjust the desired compute budget and relative importance of task performances after deployment, without retraining. This enables optimizing performance for dynamically varying user needs, without heavy computational overhead to train and save models for various scenarios. To this end, we propose a multi-task model consisting of a shared encoder and task-specific decoders where both encoder and decoder channel widths are slimmable. Our key idea is to control the task importance by varying the capacities of task-specific decoders, while controlling the total computational cost by jointly adjusting the encoder capacity. This improves overall accuracy by allowing a stronger encoder for a given budget, increases control over computational cost, and delivers high-quality slimmed sub-architectures based on user's constraints. Our training strategy involves a novel 'Configuration-Invariant Knowledge Distillation' loss that enforces backbone representations to be invariant under different runtime width configurations to enhance accuracy. Further, we present a simple but effective search algorithm that translates user constraints to runtime width configurations of both the shared encoder and task decoders, for sampling the sub-architectures. The key rule for the search algorithm is to provide a larger computational budget to the higher preferred task decoder, while searching a shared encoder configuration that enhances the overall MTL performance. Various experiments on three multi-task benchmarks (PASCALContext, NYUDv2, and CIFAR100-MTL) with diverse backbone architectures demonstrate the advantage of our approach. For example, our method shows a higher controllability by ∼ 33.5% in the NYUD-v2 dataset over prior methods, while incurring much less compute cost.

1. Introduction

Multi-task learning (MTL) often aims to solve multiple related tasks together using a single neural network for economy of deployment [1, 2]. Humans can handle multiple tasks with diverse trade-offs (e.g., due to availability of resources,
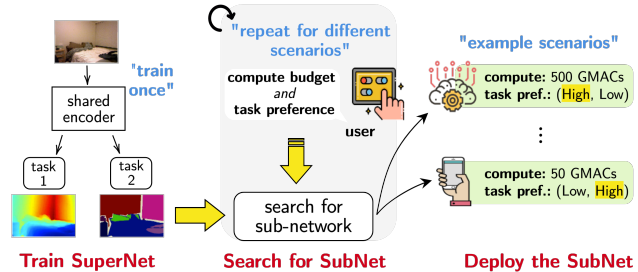

Figure 1. **Problem Setup.** We aim to provide users precise control on compute allocation as per their MTL performance preference, with the ability to change these dynamically without re-training. To accomplish this, we provide a strategy where a MTL SuperNet is trained only once but allows crafting SubNets that can be sampled based on the user's MTL constraints (compute cost *and* task preference) at test-time. "High" task preference for task i implies the performance for task i is more important than other tasks.

adaptable reaction time, etc.), however, most existing MTL architectures are incapable of transforming themselves to handle multiple user constraints without being retrained for each scenario. In this paper, we address the problem of designing controllable dynamic convolutional neural networks (CNN) for MTL that can adjust jointly for two types of user requirements, task preference and compute budget.

Real-world MTL systems are seeing growing applications ranging from autonomous cars [3] to video cameras for traffic analysis [4], with respective task performance preferences. For example, observe Fig. 1. A single MTL architecture can allow two users to use the same model but with custom task preferences based on the available compute cost. The user with higher compute (*e.g.* self-driving cars) may expect higher performance on task *1*, but the user with lower compute (*e.g.* traffic cameras) would prefer higher performance on task *2* given the budget. It will be extremely inefficient to create and train MTL architectures for all such possible variations of user requirements due to expensive design and deployment costs [5–7]. This brings forth the need for flexible MTL architectures that allow test-time trade-offs based on relative task importance *and* resource allocation.

Some prior methods have introduced dynamic MTL networks [8–11] in an effort to incorporate changing user task

preferences at test-time. However, such methods do not account for changing the user's computational budget as they assume fixed computation cost resulting in limited applicability. Recently, Controllable Dynamic Multi-task Architecture (CDMA) [7] introduced a multi-stream (equal to number of tasks) architecture to handle both changing task preferences and compute budgets. For controllability, it adjusts branching locations in the encoder and generates encoder weights using external hypernetworks [12] while fixing decoders.

In this paper, we propose a multi-task method called 'Efficient Controllable Multi-Task architectures' (ECMT) that consists of a shared encoder and task-specific decoders where the channel widths of both modules are slimmable [13]. Our key idea is to control the task importance by varying the capacities of task-specific decoders while controlling the compute budget by jointly adjusting the encoder capacity. This is based on our observation that a larger backbone achieves overall higher multi-task accuracy (even with task conflicts) compared to separately trained multiple smaller backbones. Further, in contrast to adjusting the branching points of multiple encoder streams [7], our approach can achieve overall higher controllability by adopting one stronger backbone for a given compute budget. Since decoder widths affect both accuracy and computational cost by a considerable amount, especially for dense prediction tasks, adjustment of decoder capacities is sufficient to control the task preferences. Constraining to control the task preference only through the decoder capacities further avoids adversarial effects when changing the shared encoder as it may cause different effects to each task, which is hard to control. Finally, adjusting both decoder and encoder largely increases the control over the computational cost.

As the training is performed only once and the encoder is shared among tasks, ECMT optimizes the sub-architectures by distilling [14] the encoder knowledge of the parent architecture, that is capable of handling task conflicts given its large capacity [15–17]. In particular, it uses a novel 'Configuration Invariant knowledge distillation' (CI-KD) strategy to make the embeddings of the shared encoder invariant to the varying sub-architecture configuration. At test-time, ECMT uses the joint constraints and extracts a sub-architecture by searching for the most suitable encoder and decoder width configuration using the proposed evolution-based algorithm [18] designed for MTL models. The key rule for the search algorithm is to provide a larger computational budget to the higher preferred task decoder, while searching a shared encoder configuration that enhances overall MTL performance.

Interestingly, without any need for external hypernetworks (to predict large tensor weights of the parent architecture) and with a shared encoder (that allows task scalability), ECMT demonstrates strong *task preference - task accuracy - efficiency* trade-offs. Our extensive experiments on benchmark datasets demonstrate strong MTL controllability across

a wide range of joint preferences (*e.g.*, an increase in Hypervolume [19] of ∼34% is observed when compared to state-of-the-art [7] during testing in the NYUD-v2 dataset [20]). To summarize, our contributions in this paper are:

1. We present a new method to sample high-performing efficient MTL sub-architectures from a single MTL SuperNet that can satisfy both user preferences of task performance and computational budget, dynamically without retraining.

2. Our method includes two key components:

• A *training* strategy to enhance the MTL performance of sub-architectures in order to have minimal performance drop even if user's constraints become restricted. In particular, it uses a CI-KD loss to transfer the encoder knowledge of the parent model, which is capable of handling multi-task conflicts, to the encoders of sub-models.

• A subsequent *search* strategy that translates the task preferences to sample the task decoders for better performance and searches for shared encoder width configuration that supports the decoders for overall better MTL performance.

3. We show superior controllability on sampling sub-models compared to prior methods. For example, we show a higher controllability by ∼ 33.5% in the NYU-v2 [20] (3 tasks) dataset and ∼ 55% in Pascal-Context [21] (5 tasks) dataset over state-of-the-art method CDMA [7].

2. Related Works

Multi-task Learning (MTL). Growing demands for MTL capable systems has led to a huge growth in methods for designing effectual architectures that can leverage *shared* feature representations [1, 2, 22] among tasks. Broadly, either these methods (*a*) [23–29] provide individual encoders (with respective decoders) for each task and optimize to decrease the distance between the parameters (*soft* sharing), or (*b*) optimize a single (shared) encoder for all tasks [11, 30–35] followed by individual decoders (*hard* sharing). A significant number of prior works mainly focus on building MTL strategies under static (or fixed) task preference and compute budget. Hence, any change in these constraints results in re-training the model from scratch. In order to save on designing costs and stringently follow user requirements, some prior works have presented strategies to obtain custom MTL architectures by search mechanisms [36–43] similar to traditional neural architecture search (NAS). Such MTL NAS methods are designed for a single model matching only *one* compute budget and are unsuitable for dynamic changing requirements. Different from these, we aim to provide a novel solution for MTL under dynamic (or changing) task preference and compute budget without re-training.

Controllable MTL Neural Networks. [7, 9, 11, 32, 44] have proposed controllable MTL architectures where a parent network is trained to extract sub-architectures that fol-

lows user's constraints without re-training. For example, [9, 44] presented strategies to train hypernetworks to predict weights of a MTL SuperNet in order to match user's *changing* task performance preference. A major drawback of these approaches is that they assume a static compute budget, making them incapable of handling dynamic compute resources. Some prior single task learning (STL) methods such as Once-for-all networks [5] and BigNAS [6] provide train and search strategies for dynamic compute resources. However, by design they do not provide any recipe for incorporating user MTL task preferences. In short, none of these aforementioned works provide all of dynamic and multi-task aspect as ECMT. Different from [9, 32, 44], CDMA [7] presented a method to handle multiple or joint MTL user constraints where both task preference and compute budget constraints are dynamic. It uses two hypernetworks to predict both network architecture and parameters to match joint constraints. Further, its parent model follows the soft-parameter sharing MTL setup. As a result, CDMA requires additional memory overhead due to two external hypernetworks, faces scalability issues due to soft sharing of encoder, and doesn't consider the decoder in it's search space. Better than CDMA, ECMT uses a novel training paradigm for hard-parameter sharing MTL architecture and uses a simple strategy to sample high performing sub-architectures. The search space is defined by the layer width [13, 45] and encompasses both the shared encoder and the task decoders.

Slimmable Neural Networks. Slimmable networks [13, 45, 46] proposed a method to train STL neural networks to support operating using different filter levels controlled by width (or number of filters) multipliers; e.g., a width multiplier 0.5 'slims' down all the parent model layers resulting in a sub-network that has 50% of filters. These width multipliers are uniform across the architecture, and can be tuned to match the user's memory budget with minimal performance degradation. This strategy has inspired solutions for various STL research areas such as generative adversarial networks [47], semantic segmentation [48] and others [49, 50]. Similar to [13], [5, 6] explored different freedom directions such as depth and kernel sizes to achieve strong accuracy-efficiency trade-off in STL setup. While we leverage the idea of slimmable models, ECMT is strikingly different from the aforementioned works as they do not support the capability of task performance preference trade-off. Further, [5, 6, 49, 50] have been mostly explored only for classification tasks. ECMT is designed for MTL setup that extends beyond classification tasks to dense prediction tasks.

3. Problem Statement

Notations. We denote data distribution as composed of training set $\mathcal{D}_{tr}$, validation set $\mathcal{D}_{val}$, and testing set $\mathcal{D}_{te}$ with N tasks. Each task shares the input image x with

corresponding outputs $\mathcal{Y} = \{y_1, y_2, \cdots, y_N\}$. We create a MTL parent architecture or SuperNet $\mathcal{S}$ composed of a single shared encoder among tasks and N task decoders. $\mathcal{S}$ is end-to-end non-uniformly slimmable: every layer can be tuned to have it's own set of filters $\mathcal{F}$ (also called *width* [13]) which is independently controlled by a separate width ratio $\omega \in (0, 1]$. Let $\boldsymbol{\omega} = [\omega_{\min}, \cdots, \omega_{\max}]$ be the set of possible values of width ratios with $\omega_{\max}$ ($\omega_{\min}$) representing the maximum (minimum) possible values. A sub-network or SubNet $\mathcal{S}^{(\varsigma)}$ can be created from $\mathcal{S}$ by setting width ratios for all L layer denoted using a L-tuple $\mathcal{S}^{(\varsigma)} = \langle \omega_1^{(\varsigma)}, \omega_2^{(\varsigma)}, \cdots, \omega_L^{(\varsigma)} \rangle \in \boldsymbol{\omega} \times \boldsymbol{\omega} \times \cdots \boldsymbol{\omega}$. Let the set of loss functions for the N tasks be $\{\mathcal{L}_1, \mathcal{L}_2, \cdots, \mathcal{L}_N\}$. Formally, task preference indicates amount of available compute budget for the given task. We denote the task preference list be $\boldsymbol{\mathcal{T}} = [\tau_1, \tau_2, \cdots, \tau_N]$ with $\tau_i \in [0.0, 1.0]$, where higher value indicates higher preference. Finally, we denote the weights of $\mathcal{S}$ as θ.

Problem Statement. Our goal is to train the MTL SuperNet $\mathcal{S}$ that allows crafting multiple MTL SubNets $\mathcal{S}^{(\varsigma)}$ operable for a wide range of joint MTL user budgets (compute budget F_{user}, and task preference $\boldsymbol{\mathcal{T}}_{user}$) with minimal performance drop. The SuperNet $\mathcal{S}$ (and the SubNets $\mathcal{S}^{(\varsigma)}$) takes image x as input and predicts N task outputs $\mathcal{Y}$. To train $\mathcal{S}$, we define the following problem:

$$\arg\min_{\boldsymbol{\theta}} \quad \mathbb{E}_{x, \mathcal{Y} \sim \mathcal{D}_{tr}} \sum_{n=1}^{N} \rho_n \mathcal{L}_n(\boldsymbol{\theta}) \qquad (1)$$

Here, ρ_n is the weight of nth task loss. Once $\mathcal{S}$ is trained, the joint constrained search for obtaining $\mathcal{S}^{(\varsigma)}$ can be expressed as the following problem:

$$\min_{\mathcal{S}^{(\varsigma)}} \quad \mathbb{E}_{x, \mathcal{Y} \sim \mathcal{D}_{val}} \sum_{n=1}^{N} \rho_n \mathcal{L}_n(\mathcal{S}^{(\varsigma)})$$

$$\text{s.t. compute}(\mathcal{S}^{(\varsigma)}) \leq F_{user},$$

$$\text{task preference}(\mathcal{S}^{(\varsigma)}) = \boldsymbol{\mathcal{T}}_{user} \qquad (2)$$

4. Proposed Method

During training, we solve the problem in Eq. 1 by constructing a SuperNet $\mathcal{S}$ parameterized by layer-wise width ratios in $\boldsymbol{\omega}$ (see Sec. 4.1). During inference, we solve Eq. 2 by searching for the most suitable encoder and decoder width configuration using an evolution-based search algorithm based on the joint constraints (see Sec. 4.2). The training is performed only once, whereas the search is performed for each deployment scenario.

4.1. Training the MTL SuperNet

Overview. Fig. 2-(A) provides an overview of the proposed training pipeline. At its core, the training strategy aims to
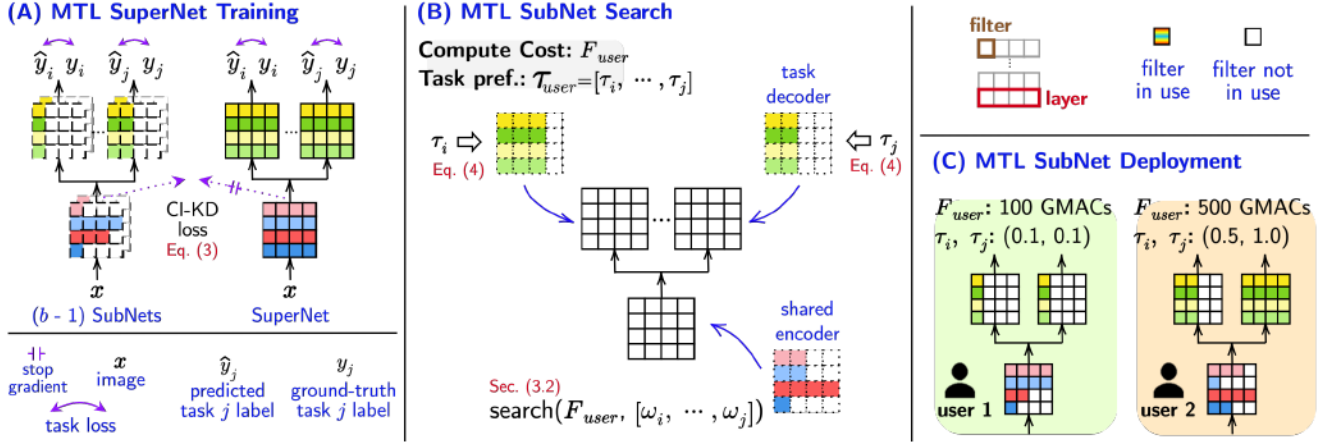
Figure 2. Illustration of our pipeline. Our overall pipeline to obtain MTL SubNets is presented. In (**A**), we present the SuperNet training strategy where we train the MTL SuperNet collaboratively with the MTL SubNets. We propose a knowledge distillation loss to transfer the knowledge of the largest capacity encoder, which has less task conflicts, to smaller capacity encoders of the SubNets. In (**B**), we demonstrate our novel strategy to search SubNets (both the shared encoder and decoders) based on the user's joint constraints (task preference and available compute budget). In (**C**), the search algorithm can provide SubNets as per multiple user preferences.

enhance the MTL performance of the SubNets sampled from the SuperNet in order to have minimal performance drop even if user's constraints become restricted. To this end, we leverage the Sandwich Rule (SR) training [45] and make necessary modifications for our MTL setup. In particular, it involves a novel (width) configuration-invariant knowledge distillation loss aimed at teaching the SubNet encoders from the SuperNet encoder. We now discuss the pipeline in detail.

Training with the Sandwich Rule. The SR training [45] for single-task learning (STL) requires that in each training iteration, we update the SuperNet with the collectively accumulated loss gradients of the model at b widths. We follow [45] and choose $b = 4$ which includes the model at largest width $\omega_{\max}$, smallest width $\omega_{\min}$, and $b-2$ randomly chosen models at non-uniform widths. Further, the STL SubNets in SR [45] are optimized *only* using the predictions of the largest width model (*i.e.* SuperNet). We build upon this rule and introduce the following changes.

In contrast to the aforementioned, we enforce each SubNet to learn the MTL data distribution directly from the available ground-truth labels y. The training loss of collective learning (*i.e.* training each SubNet as the SuperNet from ground-truth labels) is denoted as $\mathcal{L}_{co}$. Training SubNets with ground-truth labels helps us avoid the following pitfall: we do not need to train the MTL SubNets from the output predictions of a weak parent MTL model (i.e. predictions in the initial iterations can be weak as it is being trained from scratch). We could tackle this by training the parent model standalone, but this is contrary to our goal of saving on design costs of training *once*. Furthermore, we could also adopt knowledge distillation strategies proposed in prior MTL works [51, 52]. However, doing so would create additional training overhead

for STL models for each task, which is also not our end-goal. Hence, we present a new methodology of distilling the knowledge of the parent model $\mathcal{S}$ to SubNets without using output predictions, which brings us to our proposed encoder-based knowledge distillation (KD) loss.

Configuration Invariant KD (CI-KD) Loss. Our CI-KD loss is an *in-place distillation loss* $\mathcal{L}_{kd}$, which transfers the encoder knowledge of $\mathcal{S}$ to the encoders of sub-networks $\mathcal{S}^{(\varsigma)}$. The encoder of $\mathcal{S}$ is capable of handling multi-task conflicts due to its high capacity [15–17] and we aim to teach the encoders of the smaller models from its features. In particular, we propose to minimize distance between the encoder features computed from parent model $\mathcal{S}$ and all the ith child model involved in the sandwich setup. Now, this loss cannot be directly estimated: the features of $\mathcal{S}$'s encoder z and other child models $z^{(i)}$ are of different sizes due to the different configurations of the SubNet encoders. To make the shared encoder feature size *configuration invariant*, we compute the average features along the channel dimensions for all models in the Sandwich. We then minimize the mean square error loss between these channel-averaged features of the parent model and the $b-1$ child models as follows.

$$\mathcal{L}_{kd} = \frac{1}{b-1} \sum_{i}^{b-1} \text{MSE}(\hat{z}, \hat{z}^{(i)}) \qquad (3)$$

Here, $\hat{z}$ and $\hat{z}^{(i)}$ are the encoder features of $\mathcal{S}$ and ith child model (in the Sandwich), averaged along the channel dimension. Note that $\hat{z}$ is detached from the computational graph as we do not intend to update the parent model with $\mathcal{L}_{kd}$. This distillation loss has been illustrated in Fig. 3. To summarize, the SuperNet learning loss only includes Eq. 1, whereas the ith SubNet learning loss includes both with Eq.

ALGORITHM 1
ECMT training for SuperNet $\mathcal{S}$.

Require: Width list $\boldsymbol{\omega}$, *e.g.* $[0.3, 0.4, \cdots 1.0]$.
Require: Number of models b in Sandwich Rule, *e.g.*, $b = 4$.
1: **for** $t = 1, \cdots, T_{epoch}$ **do**
2:     Get input image x and task labels $\mathcal{Y}$.
3:     Clear gradients, `optimizer.zero_grad()`.
4:     Get task outputs & enc. feature z from $\mathcal{S}$; $\hat{\mathcal{Y}}, z = \mathcal{S}(x)$.
5:     Get total loss $\mathcal{L}_{\text{CO}}$ using $\{\hat{\mathcal{Y}}, \mathcal{Y}\}$ in Eq. 1.
6:     Accumulate gradients, $\mathcal{L}_{\text{CO}}$`.backward()`.
7:     Stop gradients of z, $z = z$`.detach()`.
8:     Randomly sample $(b-1)$ models $\{S_b = \mathcal{S}^{(1)}, \cdots, \mathcal{S}^{(b\ 1)}\}$.
9:     Add smallest width model to S_b.
10:    **for** $\mathcal{S}^{(i)}$ in S_b **do**
11:       Get task outputs & enc. feature z from $\mathcal{S}^{(i)}$
12:       Compute loss $\mathcal{L}^{(i)} = \mathcal{L}_{\text{CO}} + \lambda \mathcal{L}_{\text{kd}}$.
13:       Accumulate gradients, $\mathcal{L}^{(i)}$`.backward()`.
14:    **end for**
15:    Update SuperNet weights, `optimizer.step()`.
16: **end for**

Figure 3. Illustration of our Configuration Invariant Knowledge Distillation (CI-KD) Loss. This loss encourages the shared encoder features to be invariant of the ith SubNet capacity, by enforcing them to be close to the SuperNet's shared encoder features.

1 and $\lambda \mathcal{L}_{\text{kd}}$ (λ is the weight of the CI-KD loss). The training algorithm is given in Algo. 1.

4.2. Searching based on Joint User Preferences

Overview. Fig. 2-(**B**) provides a simple method to sample sub-networks that follow the user's joint constraints of task preference $\boldsymbol{\tau}_{user}$ and compute budget F_{user}. We divide the search into two parts. In *Step 1*, we sample the width ratios of the task decoders based on the user's task preference using a simple rule proposed in Eq. 4. With the decoder configuration fixed, we search for an optimal shared encoder configuration in *Step 2*. The goal of the search is to support the sampled task decoders for better performance than randomly choosing the configuration while satisfying the overall compute budget. *Step 2* particularly involves a MTL accuracy predictor that provides quick feedback on the overall model configuration during the search cycle and saves repeated evaluation related computations. We now expand on each step in detail.

Step 1: **Setting the task decoders.** We propose to set the width ratios of the task decoders based on the task preference $\boldsymbol{\tau}_{user} = \{\tau_i\}_i^N$ as they are independent for each task. In particular, we map each τ_i to the discrete uniform range of $\boldsymbol{\omega} \sim \mathcal{U}(\omega_{\min}, \omega_{\max})$. Assuming $\boldsymbol{\tau} \sim \mathcal{U}(0, 1)$ as a uniform distribution with unit density when $0 \leq \tau \leq 1$ (0 otherwise), τ_i is mapped to a decoder width ratio ω_i as:

$$\omega_i = \omega_{\min} + (\omega_{\max} - \omega_{\min})\tau_i \qquad (4)$$

Clearly, $\omega_i \propto \tau_i$ *i.e.* the decoder of the task with higher preference will be assigned a higher width ratio. This design choice is motivated by the reason to allow a larger computational budget in the available user's budget to the higher preferred task decoder. Once all the decoders are fixed using Eq. 4, we search for a shared encoder width configuration to support the aforementioned width decoder configuration.
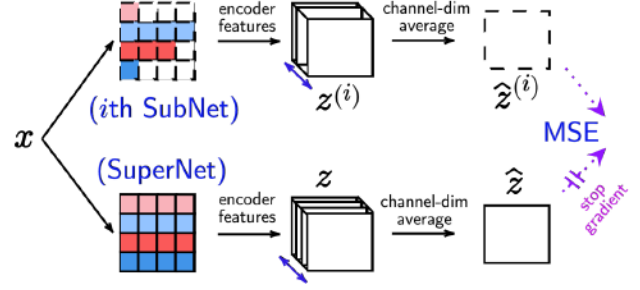
Step 2: **Searching for the shared encoder.** The aim is to sample a width ratio configuration for the shared encoder, that supports the best performance out of the sampled decoders. Randomly choosing the encoder configuration is one possible option, but it doesn't necessarily result in the best architecture due to our large search space. In order to leverage the large search space designed by our non-uniform layer setup and find a better performing model, we use an evolution-based search algorithm [18]. The search algorithm consists of three key components. First, we initialize a pool $\mathcal{P}$ of P models ($\mathcal{P} = \{\mathcal{S}^{(1)}, \mathcal{S}^{(2)}, \cdots, \mathcal{S}^{(P)}\}$), all with the fixed decoder configuration obtained from *Step 1*. Each of these models are characterized by the same width ratio across all encoder layers. Next, we evolve $\mathcal{P}$ in order to find a better performing model than the initialized ones by leveraging the flexibility of choosing width ratios for each layer mutually exclusively in $\mathcal{S}^{(\varsigma)}$. We randomly choose $K < L$ encoder layers and change the width ratio ω_k by the rule:

$$\widehat{\omega}_k = \omega_k + \eta \, \text{sign}(F_{user} - F^{(\varsigma)}) \qquad (5)$$

Here, $F^{(\varsigma)}$ is the computational cost of $\mathcal{S}^{(\varsigma)}$ (*e.g.* GMACs), F_{user} is the computational budget set by the user, and $\text{sign}(\cdot)$ extracts the sign of the input variable. We set $\eta = 0.1$ as we use the design specification $\omega_i - \omega_j = 0.1$. The motivation of this design choice is to push the shared encoder configuration towards larger capacity, which are known to handle task conflicts better [15–17]. This evolution step creates a new model $\widehat{\mathcal{S}^{(\varsigma)}}$ which is added back to $\mathcal{P}$. In the end, the best performing model from the search is provided for deployment. At all steps, we ensure that each model $\mathcal{P}$ satisfies the user's compute budget constraint. In order to quickly evaluate the quality of models in $\mathcal{P}$, we build a subsidiary neural network $\mathcal{R}$ that provides a feedback on $\widehat{\mathcal{S}^{(\varsigma)}}$'s approximate MTL performance. This MTL accuracy predictor $\mathcal{R}$ eliminates the need for repeated cost of getting the *measured* accuracy by providing a *predicted* accuracy. Specifically, $\mathcal{R}$ is optimized to take $\mathcal{S}^{(\varsigma)}$'s width configuration as input and predict the approximate performance of this

ALGORITHM 2

ECMT SubNet search.

Require: MTL Accuracy Predictor $\mathcal{R}$, Number of search cycles T_{cycle}
Require: Model pool size P
Require: User MTL constraints (F_{user}, $\mathcal{T}_{user}$)
1: Compute and fix decoder widths using $\mathcal{T}_{user}$ using Eq. 4.
2: Initialize the model pool $\mathcal{P}$ with P random encoder widths, including uniform encoder widths, that satisfy memory constraint F_{user}
3: Get the most preferred task $\mathcal{T} = \max(\mathcal{T}_{user})$
4: **for** $t = 1, \cdots, T_{cycle}$ **do**
5:     Randomly choose a model $\mathcal{S}^{(i)}$ from $\mathcal{P}$
6:     Change the width of a randomly chosen encoder layer with Eq. 5 in $\mathcal{S}^{(i)}$ and create $\widehat{\mathcal{S}^{(\varsigma)}}$
7:     Add the mutated model $\widehat{\mathcal{S}^{(\varsigma)}}$ to $\mathcal{P}$
8:     Delete the worst model (which results in highest loss in the preferred task $\mathcal{T}$) computed using $\mathcal{R}$.
9: **end for**
10: Return the width configuration of the best performing model in $\mathcal{P}$ predicted using $\mathcal{R}$.

configuration [5]. To train $\mathcal{R}$, we first create K examples of $[\mathcal{A}^{(\varsigma)}, (\mathcal{L}_1, \cdots, \mathcal{L}_N)]$ pairs by randomly sampling M Sub-Nets with different configurations $\mathcal{A}^{(\varsigma)}$, and computing their task losses on $\mathcal{D}_{val}$. $\mathcal{A}^{(\varsigma)}$ contains the list of width ratios computed for the shared encoder and the task decoders. In our experiments, we choose $M = 2000$. The architecture of $\mathcal{R}$ has been provided in Algo. 3 and training procedure is provided in Sec. 5. Note that, we only need to collect the data pairs and train $\mathcal{R}$ once, making this overhead negligible in comparison to training $\mathcal{S}$. Further, this cost remains constant regardless of changing user requirements. Fig. 2-(C) provides some examples of the resultant MTL architectures. We summarize our proposed MTL search algorithm in Algo. 2. In Algo. 2, we call a candidate model "mutated" (in *Line* 7) when we change a random encoder layer (in *Line* 6).

5. Experiments

In this section, we demonstrate the ability of ECMT to extract efficient architectures based on a user's joint multi-task learning preferences. We show that our proposed framework can scale to a large number of tasks while allowing proficient *task preference - task accuracy - efficiency* trade-off with a single MTL model. Through extensive experiments on three datasets with various architectures, we provide new insights to the community regarding controllable MTL architectures which can be re-used for diverse deployment scenarios without re-training.

Training implementation details. We use PyTorch [53] for our implementation.

• **NYUD-v2 and Pascal-Context.** We follow [2] to set the following hyperparameters. The SuperNet is trained with Adam Optimizer [54] with learning rate 0.0001, and weight decay 0.0001. We use a poly learning rate scheduler. We set the total epochs T_{epochs} as 100 for PASCAL-Context/NYUD-v2. The training batch size is set to 8. The

task loss weights ρ_n are set as follows- *NYU-v2* $\rightarrow$ Semantic segmentation: 1, Depth: 1, and Surface normals: 10; *PASCAL-Context* $\rightarrow$ Semantic segmentation: 1, Human parts segmentation: 2, Saliency: 5, Edge: 50, Surface normals: 10. The training data augmentations are set from [41]. We further add Random Erasing [55] with probability 0.5 to the aforementioned. The shared encoder is initialized with ImageNet [56] pre-trained weights, whereas the decoders are trained from scratch. The CI-KD loss weight λ is set to 1 and 0.001 for NYUD-v2 and Pascal-Context, respectively. ω is set as $[0.6, 0.7, \cdots, 1.0]$ for NYUD-v2, and $[0.5, 0.6, \cdots, 1.0]$ for Pascal-Context. The input channels to the DeepLabv3 decoder for NYUD-v2 and Pascal-Context are set to 256 and 128, respectively.

• **CIFAR100.** All the task loss weights are set to 1. The batch size is set to 256. We set the total epochs T_{epochs} is set as 75. The SuperNet is trained with Adam Optimizer [54] with learning rate 0.001, and weight decay 0.0001. The whole SuperNet is trained from scratch. The CI-KD loss weight λ is set to 0.01. ω is set as $[0.2, 0.3, \cdots, 1.0]$.

All experiments use 90% of training set for all datasets. All other hyperparameters are set as per default PyTorch settings.

Evaluation criteria. Following [7], we use hypervolume (HV) [19] to measure controllability with respect to task preferences. HV helps to evaluate the preference-loss trade-off curve (or Pareto-front [11]) among the different tasks in the loss space [57]. Basically, HV accounts for both the quality of the models (volume they dominate), and also the diversity, measured in the overlap between dominated regions. Higher value indicates better controllability assuming same memory budget. Bold values indicate best results.

Inference implementation details. For comparisons with CDMA and PHN for controllability, we follow CDMA's method of task preference sampling. In particular, we sample task preferences $\tau \sim P_\tau$, where P_τ is a Dirichlet distribution of order N with parameters $[\alpha_1, \alpha_2, \cdots, \alpha_N]$ ($\alpha_i > 0$). Following [7], we set $\alpha_i = 0.2 \, \forall \, i$ for PASCAL-Context/NYUD-v2, and $\alpha_i = 1.0 \, \forall \, i$ for CIFAR100. HV is computed across 20 task preferences following [7]. Deep-Speed [58] is used to compute the architecture GMACs. For any $\mathcal{T}_{user}$, we always aim to set the highest width for *most preferred task* for best results. Hence, we min-max norm $\mathcal{T}_{user}$ before using Eq. 4. Any value $\in [0, 1]$ can be used for each task preference (sum need not be 1). We set the number of search cycles T_{cycle} as 150 for CIFAR100 and 10 for NYUD-v2/Pascal-Context. We set the model population size to 50. The search runtime is proportional to T_{cycle}.

Datasets. We conduct our experiments using three multi-task datasets, namely PASCAL-Context [21] (5 tasks) and NYU-v2 [20] (3 tasks), and CIFAR-100 [59, 60] (20 tasks). The PASCAL-Context dataset provides joint semantic segmentation, human parts segmentation, saliency estimation,

ALGORITHM 3
Pseudocode of our MTL Accuracy Predictor $\mathcal{R}$.

```python
import torch.nn as nn

# ----- multi-task performance predictor
class AccuracyPredictor(nn.Module):
    def __init__(self, input_channels: int,
        task_names: list):
        super().__init__()
        self.model = nn.Sequential(
            nn.Linear(input_channels, 100),
            nn.ReLU(),
            nn.Linear(100, 100),
            nn.ReLU(),
            nn.Linear(100, 50),
            nn.ReLU()
        )
        self.task_names = task_names

        self.heads = nn.ModuleDict({task:
            nn.Linear(50, 1) for task in
            self.task_names})

    def forward(self, model_widths:
        torch.Tensor):
        feats = self.model(model_widths)
        preds = {t_id: self.heads[t_id](feats)
            for t_id in self.task_names}

        return preds
```

Table 1. **Evaluation on NYUD-v2**. Reference point for HV: $[4, 4, 4]$. † indicates smaller compute cost (GMACs).

Method	GMACs ($\downarrow$)	Control Params.($\downarrow$)	HV ($\uparrow$)
PHN	21.02	21.04M	02.36
PHN-BN	21.02	02.23M	11.72
CDMA†	25.98	00.03M	09.53
CDMA	29.04	00.03M	12.42
MTL-Static	22.66	-	35.64
ECMT †	10.00	00.20M	**34.91**
ECMT	20.00	00.20M	34.89

Table 2. **Evaluation on PascalContext**. Reference point for HV: $[3, 3, 3, 3, 3]$. † indicates smaller compute cost (GMACs).

Method	GMACs ($\downarrow$)	Control Params.($\downarrow$)	HV ($\uparrow$)
PHN	06.28	21.50M	42.61
PHN-BN	06.28	03.63M	72.27
CDMA†	06.81	15.32M	73.20
CDMA	07.21	15.32M	75.52
MTL-Static	09.43	-	242.79
ECMT †	04.00	0.021M	166.75
ECMT	09.00	0.021M	**166.97**

surface normal estimation, and edge detection. The NYU-v2 dataset provides joint semantic segmentation, depth estimation and surface normal estimation. The CIFAR-100 (MTL) dataset provides a split of 20 five-way classification tasks extracted from the original dataset. Semantic segmentation, saliency estimation and human parts segmentation are evaluated using mean intersection over union (mIoU). Depth estimation and surface normals are evaluated using the root mean square error (rmse).

Baselines. As reported in Sec. 2, CDMA [7] is our main baseline which designs MTL architectures, jointly controllable with task preference and memory constraint. Further, we compare with PHN [44] (and its variation PHN-BN [7, 44]), which **only** allow task controllability. PHN uses a hypernetwork to predict the weights of a shared backbone based SuperNet conditioned on a task preference. Further, we also compare with MTL-Static models trained for highest user constraints. It provides the upper bound of the HV ECMT can achieve. The MTL-Static models and ECMT are trained with 90% samples of the original training split. The other 10% split is used to train the MTL accuracy predictor.

Architectures. Our SuperNet $\mathcal{S}$ consists of a shared encoder (backbone) with N task decoders. For dense prediction tasks (PascalContext, NYUD-v2), we use the DeepLabv3 architecture [61] for each task, whereas we use linear layers for classification tasks (CIFAR100-MTL). We follow CDMA

and choose the backbone architectures as MobileNetV2 [62] for PASCAL-Context, ResNet34 [63] for NYUD-v2, and ResNet9 [64] for CIFAR100-MTL. MobileNetV2 and ResNet34 are pre-trained on ImageNet [56]. Note that, in all cases, both the encoder and the task decoders are non-uniformly slimmable. Following [6, 45], we do not accumulate batch norm statistics while training the SuperNet as these are ill-defined due to varying SubNet configurations. During testing, we re-calibrate the batch norm statistics for each SubNet without any fine-tuning of its parameters.

MTL accuracy predictor details. The MTL accuracy predictor $\mathcal{R}$ is a three layer feed-forward neural network summarized in Algo. 3. Given a SubNet model $\mathcal{S}^{(\zeta)}$, we create a list $\mathcal{A}^{(\zeta)}$ consisting of width ratios starting with the shared encoder, followed by the decoder's width ratios. $\mathcal{A}^{(\zeta)}$ is then fed to the $\mathcal{R}$ to get the predicted MTL accuracy for all N tasks. To train $\mathcal{R}$, we first create $M = 2000$ examples of $[\mathcal{A}^{(\zeta)}, (\mathcal{L}_1, \cdots, \mathcal{L}_N)]$ pairs by randomly sampling M Sub-Nets with different configurations $\mathcal{A}^{(\zeta)}$, and computing their task losses on $\mathcal{D}_{val}$. We train this predictor for 150 epochs using L1-loss using the Adam optimizer with a learning rate of 0.001. The batch size is set to 16.

5.1. Comparison with Baselines

We present the overall multi-task performance of sampling sub-architectures with three different distributions NYUD-v2 (3 tasks), Pascal-Context (5 tasks) and CIFAR-100 (20 tasks) in Tab. 1-3. Here we compare with state-of-the-art methods CDMA (allows compute budget controlla-

Table 3. **Evaluation on CIFAR100-MTL**. Reference point for HV: $[1, 1, \cdots, 1]$. † indicates smaller compute cost (GMACs)

Method	GMACs ($\downarrow$)	Control Params.($\downarrow$)	HV ($\uparrow$)
PHN	073.13	11.03M	0.002
PHN-BN	073.13	00.31M	0.007
CDMA†	129.23	03.10M	0.009
CDMA	174.36	03.10M	**0.010**
MTL-Static	072.92	-	0.0089
ECMT †	038.40	00.02M	0.0026
ECMT	072.92	00.02M	0.0082

Figure 4. **Loss curves comparison on NYUD-v2 (3 tasks).** Due to strategy to translate task preferences to task decoders, task losses remain low even if the preference decreases. This supports our training paradigm to improve the performance of the smaller architectures, not done in CDMA and PHN.

bility along with task controllability), PHN and PHN-BN (allows only task controllability). Following CDMA [7], we report HV (reference point set according to CDMA) across 20 task preferences. Similar to CDMA, we analyze ECMT under both small and large extremes of compute budgets (indicated using GMACs). We also account for the external networks used to sample sub-architectures under 'Control Parameters'. For PHN and CDMA, this is the cost of the hypernetworks, while for ECMT, the cost pertains to the MTL accuracy predictor $\mathcal{R}$. Note that results for both CDMA and ECMT are computed from one model while varying compute and task preference constraints.

ECMT demonstrates high HV values compared to the existing methods, especially in NYUD-v2 and Pascal-Context. The high HV along with lower GMACs supports our claim that the proposed shared encoder and individual decoders are capable of solving multi-task problems while handling task trade-offs. For example, ECMT shows a higher HV by $\sim 33.5\%$ in the NYUD-v2 dataset and $\sim 55\%$ in Pascal-Context dataset, while using comparatively much less compute in both the cases. We believe this strong performance is due to our strategy to incorporate task decoders in the search space which is ignored by CDMA. For the classification task of CIFAR-100, ECMT achieves similar HV with lesser compute cost. We believe that the impact of our task preference translation as per Eq. 4 is smaller for CIFAR-100 given that the decoder of the classifier (fully connected layers) is computationally lighter than dense prediction decoders. Finally, our strategy to translate the task preferences to the decoders allows us to maintain the HV even with highly restricted compute cost. This is highly advantageous compared to

Table 4. **Ablation Study on NYUD-v2.** $\mathcal{T}_1$: Semantic Seg., $\mathcal{T}_2$: Human-Parts Seg., $\mathcal{T}_3$: Depth Est., Here, the colored row indicates performance of largest model for reference. Bottom two rows indicate the impact of loss $\mathcal{L}_{kd}$ on the performance of smallest possible models for our method. HV is computed by keeping the budget indicated under GMACs for all the cases.

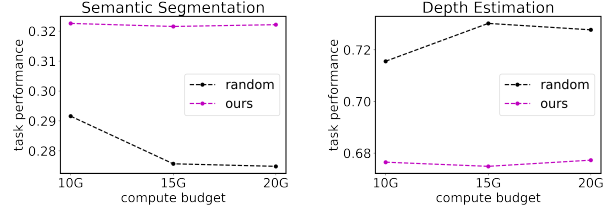| Learning Loss		GMACs	Task Performance			Control
$\mathcal{L}_{co}$	$\mathcal{L}_{kd}$		$\mathcal{T}_1$ ($\uparrow$)	$\mathcal{T}_2$ ($\downarrow$)	$\mathcal{T}_3$ ($\downarrow$)	HV ($\uparrow$)
–	–	22.66	36.03	26.82	0.63	34.86
✓	✗	08.39	30.35	28.86	0.70	32.87
✓	✓		32.38	28.52	0.67	33.81

Figure 5. **Search algorithm comparison on NYUD-v2**. Sematic seg.: *higher* and Depth est.: *lower* is better. Our proposed search algorithm provides better architectures with negligible search cost.
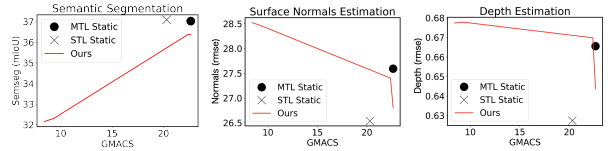

Figure 6. **Saving design costs.** NYUD-v2 (3 tasks) with ResNet34 backbone. Our training method allows to sample multiple high performing MTL architectures based on the user compute cost after training only once, in contrast to static MTL/STL architectures where the training has to be repeated for changing constraints.

PHN and CDMA, where the change in control parameters or the compute cost deteriorates the HV.

As the HV is a function of the loss values computed over changing task preferences, we visualize the task preference - task loss trade-off curves in Fig. 4. Due to our strategy to translate task preferences to task decoders, the task losses remain low even if the task preference decreases. This is because lower width SubNets (created due to lower task preferences) are enhanced to perform closely to the SuperNet. This further attests to our training paradigm to improve the performance of the smaller MTL architectures, a feature absent in CDMA and PHN.

5.2. Ablation Study

Effectiveness of CI-KD loss and shared encoder size. In Tab. 4, we analyze the importance of proposed loss functions in our SuperNet training using two different backbones. It can be observed that the CI-KD loss $\mathcal{L}_{kd}$ pushes the performance of the smallest network towards the parent network for both the cases. For example, for the ResNet34 backbone SuperNet, CI-KD loss increases the semantic segmentation
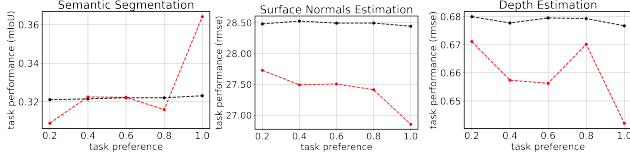
Figure 7. **Marginal evaluation on NYU-v2.** Our proposed method to translate task preference to task decoders is effective in two encoder configurations (largest, **smallest**). Different tasks have different response to encoder configuration which shows the necessity of our search algorithm to find an optimal encoder configuration.

by almost 2 points whereas the HV increases by 1 point for the smallest architecture with 37% less compute cost.

Effectiveness of our search strategy. Our search algorithm is aimed to find the best shared encoder configuration that supports the sampled task decoders based on Eq. 4. To show its efficacy, we compare it with the strategy of choosing the width ratios of the encoder randomly in Fig. 5 for the NYUD-v2 dataset. We train a MTL SuperNet with the training set $\mathcal{D}_{tr}$. The 'random' search algorithm then samples width ratios from the available width list $\boldsymbol{\omega}$ and chooses the encoder configuration satisfying the user's compute budget. Our proposed search, on the other hand, uses the accuracy predictor $\mathcal{R}$ created with the validation set $\mathcal{D}_{val}$ and chooses an optimal encoder configuration that supports the decoder for best task performance. Note that the time for our search strategy is negligible compared to training a sub-architecture from scratch. Further, computation overhead due to our search algorithm is only needed when the user preferences change and does not influence the final inference time.

Marginal task evaluation. In Fig. 7, we visualize the task performance for ECMT when the task preference changes, by marginalizing over preference values of the other tasks. This is done for two extreme cases of the encoder configuration: by setting the encoder widths to the largest and smallest size, respectively. Clearly, an increase in task preference leads to an increase in the task performance. This is a direct result of our proposed method to set the decoder width ratios according to Eq. 4. Further, it also shows that different tasks respond differently to the encoder configurations. For example, the task of semantic segmentation performs almost similarly for the task preference range [0.2, 0.8] for the both configurations, whereas the other tasks clearly present a bigger difference. This shows the necessity of our search algorithm to find a better encoder configuration. Fig. 7 also shows the accuracy-compute trade-off available per task.

Savings on design costs. Along with the aforementioned advantages of task performance tradeoff, our training method in Sec. 4 allows to sample multiple high performing MTL architectures while training only once. This saves expensive training costs and allows multiple deployments from one model. We demonstrate this feature of ECMT in Fig. 6 for
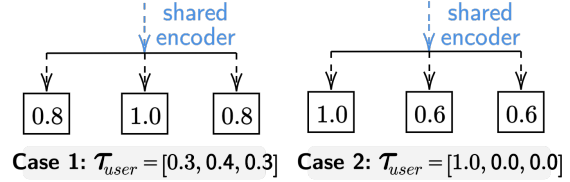

Figure 8. **Illustration of our task preference translation.** We visualize the width ratios computed for the task decoders for NYUD-v2 (3 tasks). Note that we normalize the task preference using max-min normalization before computing the decoder widths.

NYUD-v2 (3 task) dataset for ResNet34 backbone. Here, the static MTL and STL architectures can only be used for a fixed compute budget. In contrast, ECMT can operate with variable compute costs and gracefully degrade in performance with decreasing budget. Note that one can further enhance the MTL performance of our parent architecture using state-of-the-art methods summarized in [1, 2].

Qualitative examples. We visualize some examples of the task decoder widths computed using the task preferences from Eq.(4) in Fig. 8. Each block represents the task decoder with its selected width ratio for the given $\boldsymbol{\tau}_{user}$. The design choice of $\boldsymbol{\omega}$ for NYUD-v2 is $[0.6, 0.7, \cdots, 1.0]$.

6. Conclusions

We tackle the problem of multi-task architecture deployment based on user joint preferences (task performance *and* available compute). The core challenge is to train the multi-task architecture *once* but permit instant customization of the network for diverse preferences. We propose to train non-uniformly slimmable MTL architectures parameterized by layer wise width configurations, resulting in a large space of MTL architectures. At inference, we use an evolution-based search algorithm to sample precise MTL sub-architectures based on the joint preferences. In contrast to prior controllable MTL methods solely focused on encoders and providing poor task scalability, our method uses a shared encoder and the task decoders as part of search space. We show the effectiveness of our method on various multi-task settings, providing large search space for sampling efficient MTL models covering a wide range of MTL preferences.

Limitations and Future Works. Our work has the following limitations that presents new opportunities for interesting future works. ECMT has only been explored for CNN based architectures and should be investigated for transformer based models. Further, the task relativity should be considered when extracting a sub-network as it is considered an important factor in MTL.

References

[1] Yu Zhang and Qiang Yang. A survey on multi-task learning. *IEEE Transactions on Knowledge and Data Engineering*, 34 (12):5586–5609, 2021. 1, 2, 9

[2] Simon Vandenhende, Stamatios Georgoulis, Wouter Van Gansbeke, Marc Proesmans, Dengxin Dai, and Luc Van Gool. Multi-task learning for dense prediction tasks: A survey. *IEEE transactions on pattern analysis and machine intelligence*, 44(7):3614–3633, 2021. 1, 2, 6, 9

[3] Andrej Karpathy. Multi-task learning in the wilderness. *ICML. url: https://slideslive. com/38917690/multitask-learning-in-the-wilderness*, 2019. 1

[4] Qiong Chen, Zimu Zheng, Chuang Hu, Dan Wang, and Fangming Liu. On-edge multi-task transfer learning: Model and practice with data-driven task allocation. *IEEE Transactions on Parallel and Distributed Systems*, 31(6):1357–1371, 2019. 1

[5] Han Cai, Chuang Gan, Tianzhe Wang, Zhekai Zhang, and Song Han. Once for all: Train one network and specialize it for efficient deployment. In *International Conference on Learning Representations*, 2020. 1, 3, 6, 9

[6] Jiahui Yu, Pengchong Jin, Hanxiao Liu, Gabriel Bender, Pieter-Jan Kindermans, Mingxing Tan, Thomas Huang, Xiaodan Song, Ruoming Pang, and Quoc Le. Bignas: Scaling up neural architecture search with big single-stage models. In *ECCV*, pages 702–717. Springer, 2020. 3, 7

[7] Dripta S Raychaudhuri, Yumin Suh, Samuel Schulter, Xiang Yu, Masoud Faraki, Amit K Roy-Chowdhury, and Manmohan Chandraker. Controllable dynamic multi-task architectures. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 10955–10964, 2022. 1, 2, 3, 6, 7, 8

[8] Michinari Momma, Chaosheng Dong, and Jia Liu. A multi-objective/multi-task learning framework induced by pareto stationarity. In *International Conference on Machine Learning*, pages 15895–15907. PMLR, 2022. 1

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

[10] Aviv Navon, Aviv Shamsian, Gal Chechik, and Ethan Fetaya. Learning the pareto front with hypernetworks. *arXiv preprint arXiv:2010.04104*, 2020.

[11] 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. 1, 2, 6

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

[13] Jiahui Yu, Linjie Yang, Ning Xu, Jianchao Yang, and Thomas Huang. Slimmable neural networks. *arXiv preprint arXiv:1812.08928*, 2018. 2, 3

[14] Geoffrey Hinton, Oriol Vinyals, and Jeff Dean. Distilling the knowledge in a neural network. *arXiv preprint arXiv:1503.02531*, 2015. 2

[15] Shengkun Tang, Yaqing Wang, Zhenglun Kong, Tianchi Zhang, Yao Li, Caiwen Ding, Yanzhi Wang, Yi Liang, and Dongkuan Xu. You need multiple exiting: Dynamic early exiting for accelerating unified vision language model. *arXiv preprint arXiv:2211.11152*, 2022. 2, 4, 5

[16] Derrick Xin, Behrooz Ghorbani, Ankush Garg, Orhan Firat, and Justin Gilmer. Do current multi-task optimization methods in deep learning even help? *arXiv preprint arXiv:2209.11379*, 2022.

[17] Vitaly Kurin, Alessandro De Palma, Ilya Kostrikov, Shimon Whiteson, and M Pawan Kumar. In defense of the unitary scalarization for deep multi-task learning. *arXiv preprint arXiv:2201.04122*, 2022. 2, 4, 5

[18] Esteban Real, Alok Aggarwal, Yanping Huang, and Quoc V Le. Regularized evolution for image classifier architecture search. In *Proceedings of the aaai conference on artificial intelligence*, volume 33, pages 4780–4789, 2019. 2, 5

[19] 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. 2, 6

[20] 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

[21] 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 conference on computer vision and pattern recognition*, pages 891–898, 2014. 2, 6

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

[23] 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

[24] 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.

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

[26] Yongxin Yang and Timothy M Hospedales. Trace norm regularised deep multi-task learning. *arXiv preprint arXiv:1606.04038*, 2016.

[27] Yongxin Yang and Timothy Hospedales. Deep multi-task representation learning: A tensor factorisation approach. *arXiv preprint arXiv:1605.06391*, 2016.

[28] Andrei A Rusu, Neil C Rabinowitz, Guillaume Desjardins, Hubert Soyer, James Kirkpatrick, Koray Kavukcuoglu, Razvan Pascanu, and Raia Hadsell. Progressive neural networks. *arXiv preprint arXiv:1606.04671*, 2016.

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

[30] Deblina Bhattacharjee, Tong Zhang, Sabine Süsstrunk, and Mathieu Salzmann. Mult: an end-to-end multitask learning transformer. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 12031–12041, 2022. 2

[31] 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.

[32] 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, 3

[33] Rich Caruana. *Multitask learning*. Springer, 1998.

[34] Marvin Teichmann, Michael Weber, Marius Zoellner, Roberto Cipolla, and Raquel Urtasun. Multinet: Real-time joint semantic reasoning for autonomous driving. In *2018 IEEE intelligent vehicles symposium (IV)*, pages 1013–1020. IEEE, 2018.

[35] 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

[36] Yuan Gao, Haoping Bai, Zequn Jie, Jiayi Ma, Kui Jia, and Wei Liu. Mtl-nas: Task-agnostic neural architecture search towards general-purpose multi-task learning. In *Proceedings of the IEEE/CVF Conference on computer vision and pattern recognition*, pages 11543–11552, 2020. 2

[37] Thanh Vu, Yanqi Zhou, Chunfeng Wen, Yueqi Li, and Jan-Michael Frahm. Toward edge-efficient dense predictions with synergistic multi-task neural architecture search. In *Proceedings of the IEEE/CVF Winter Conference on Applications of Computer Vision*, pages 1400–1410, 2023.

[38] David Brüggemann, Menelaos Kanakis, Anton Obukhov, Stamatios Georgoulis, and Luc Van Gool. Exploring relational context for multi-task dense prediction. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 15869–15878, 2021.

[39] Pengsheng Guo, Chen-Yu Lee, and Daniel Ulbricht. Learning to branch for multi-task learning. In *International Conference on Machine Learning*, 2020.

[40] 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.

[41] 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. 6

[42] Trevor Standley, Amir Zamir, Dawn Chen, Leonidas Guibas, Jitendra Malik, and Silvio Savarese. Which tasks should be learned together in multi-task learning? In *International Conference on Machine Learning*, pages 9120–9132. PMLR, 2020.

[43] 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

[44] Aviv Navon, Aviv Shamsian, Gal Chechik, and Ethan Fetaya. Learning the pareto front with hypernetworks. In *International Conference on Learning Representations*, 2021. 2, 3, 7

[45] Jiahui Yu and Thomas S Huang. Universally slimmable networks and improved training techniques. In *Proceedings of the IEEE/CVF international conference on computer vision*, pages 1803–1811, 2019. 3, 4, 7

[46] Jiahui Yu and Thomas Huang. Autoslim: Towards one-shot architecture search for channel numbers. *arXiv preprint arXiv:1903.11728*, 2019. 3

[47] Liang Hou, Zehuan Yuan, Lei Huang, Huawei Shen, Xueqi Cheng, and Changhu Wang. Slimmable generative adversarial networks. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 35, pages 7746–7753, 2021. 3

[48] Danna Xue, Fei Yang, Pei Wang, Luis Herranz, Jinqiu Sun, Yu Zhu, and Yanning Zhang. Slimseg: Slimmable semantic segmentation with boundary supervision. In *Proceedings of the 30th ACM International Conference on Multimedia*, pages 6539–6548, 2022. 3

[49] 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. 3

[50] Ting-Wu Chin, Ari S Morcos, and Diana Marculescu. Joslim: J oint widths and weights o ptimization for slim mable neural networks. In *Machine Learning and Knowledge Discovery in Databases. Research Track: European Conference, ECML PKDD 2021, Bilbao, Spain, September 13–17, 2021, Proceedings, Part III 21*, pages 119–134. Springer, 2021. 3

[51] Wei-Hong Li and Hakan Bilen. Knowledge distillation for multi-task learning. In *Computer Vision–ECCV 2020 Workshops: Glasgow, UK, August 23–28, 2020, Proceedings, Part VI 16*, pages 163–176. Springer, 2020. 4

[52] Geethu Miriam Jacob, Vishal Agarwal, and Björn Stenger. Online knowledge distillation for multi-task learning. In *Proceedings of the IEEE/CVF Winter Conference on Applications of Computer Vision*, pages 2359–2368, 2023. 4

[53] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, et al. Pytorch: An imperative style, high-performance deep learning library. *Advances in neural information processing systems*, 32, 2019. 6

[54] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014. 6

[55] Zhun Zhong, Liang Zheng, Guoliang Kang, Shaozi Li, and Yi Yang. Random erasing data augmentation. In *Proceedings of the AAAI conference on artificial intelligence*, volume 34, pages 13001–13008, 2020. 6

[56] Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. Imagenet: A large-scale hierarchical image database. In *2009 IEEE conference on computer vision and pattern recognition*, pages 248–255. Ieee, 2009. 6, 7

[57] 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

[58] DeepSpeed. https://github.com/microsoft/DeepSpeed, 2023. 6

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

[60] 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

[61] Liang-Chieh Chen, George Papandreou, Florian Schroff, and Hartwig Adam. Rethinking atrous convolution for semantic image segmentation. *arXiv preprint arXiv:1706.05587*, 2017. 7

[62] Mark Sandler, Andrew Howard, Menglong Zhu, Andrey Zhmoginov, and Liang-Chieh Chen. Mobilenetv2: Inverted residuals and linear bottlenecks. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 4510–4520, 2018. 7

[63] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 770–778, 2016. 7

[64] Resnet9. https://github.com/davidcpage/cifar10-fast, 2018. 7