AutoMTL: A Programming Framework for Automating Efficient Multi-Task Learning

Anonymous Author(s)
Affiliation
Address
email

Abstract

Multi-task learning (MTL) jointly learns a set of tasks by sharing parameters among tasks. It is a promising approach for reducing storage costs while improving task accuracy for many computer vision tasks. The effective adoption of MTL faces two main challenges. The first challenge is to determine what parameters to share across tasks to optimize for both memory efficiency and task accuracy. The second challenge is to automatically apply MTL algorithms to an arbitrary CNN backbone without requiring time-consuming manual re-implementation and significant domain expertise. This paper addresses the challenges by developing the first programming framework AutoMTL that automates efficient MTL model development for vision tasks. AutoMTL takes as inputs an arbitrary backbone convolutional neural network (CNN) and a set of tasks to learn, and automatically produces a multi-task model that achieves high accuracy and small memory footprint simultaneously. Experiments on three popular MTL benchmarks (CityScapes, NYUv2, Tiny-Taskonomy) demonstrate the effectiveness of AutoMTL over state-of-the-art approaches as well as the generalizability of AutoMTL across CNNs. AutoMTL is open-sourced and available at `http://`.

1 Introduction

AI-powered applications increasingly adopt Convolutional Neural Networks (CNNs) for solving many vision-related tasks (e.g., semantic segmentation, object detection), leading to more than one CNNs running on resource-constrained devices. Supporting many models simultaneously on a device is challenging due to the linearly increased computation, energy, and storage costs. An effective approach to address the problem is multi-task learning (MTL) where a set of tasks are learned jointly to allow some parameter sharing among tasks. MTL creates *multi-task models* based on CNN architectures called *backbone models*, and has shown significantly reduced inference costs and improved generalization performance in many computer vision applications [23, 35].

The effective adoption of MTL faces two main challenges. The first challenge is the resource-efficient architecture design–that is, to determine what parameters of a backbone model to share across tasks to optimize for both resource efficiency and task accuracy. Many prior works [19, 22, 11, 23, 32] rely on manually-designed MTL model architectures which share several initial layers and then branch out at an ad hoc point for all tasks. They often result in unsatisfactory solutions due to the enormous architecture search space. Several recent efforts [39, 1, 17] shift towards learning to share parameters across tasks. They embed policy-learning components into a backbone CNN and train the policy to determine which blocks in the network should be shared across which task or where to branch out for different tasks. Their architecture search spaces lack the flexibility to dynamically adjust model capacity based on given tasks, leading to sub-optimal solutions as the number of tasks grows.

Submitted to 36th Conference on Neural Information Processing Systems (NeurIPS 2022). Do not distribute.

The second major challenge is the automation. Manual architecture design calls for significant domain expertise when tweaking neural network architectures for every possible combination of learning tasks. Although neural architecture search (NAS)-based approaches automate the model design to some extent, the implementation of these works is deeply coupled with a specific backbone model. Some of them [1, 39, 3, 17] could theoretically support broader types of CNNs. They, however, require significant manual efforts and expertise to re-implement the proposed algorithms whenever the backbone changes. Our user study (Section 4.2) suggests that it takes machine learning practitioners with proficient PyTorch skills 20 to 40 hours to re-implement Adashare [39], a state-of-the-art NAS-based MTL approach, on a MobileNet backbone. The learning curve is expected to be much longer and more difficult for general programmers with less ML expertise. The lack of automation prohibits the effective adoption of MTL in practice.

In this paper, we address the two challenges by developing AutoMTL, the first programming framework that automates resource-efficient MTL model development for vision tasks. AutoMTL takes as inputs an arbitrary backbone CNN and a set of tasks to learn, and then produces a multi-task model that achieves high task accuracy and small memory footprint (measured by the number of parameters). A backbone CNN essentially specifies a computation graph where each node is an operator. The key insight is to treat each operator as a basic unit for sharing. Each task can select which operators to use to determine the sharing patterns with other tasks. The operator-level sharing granularity not only enables the automatic support of arbitrary CNN backbone architectures, but also leads to a stretchable architecture search space that contains multi-task models with a wide range of model capacity. To our best knowledge, we are the first work considering parameter sharing in MTL at the operator level.

AutoMTL features a source-to-source compiler that automatically transforms a user-provided backbone CNN to a supermodel that encodes the multi-task architecture search space in the operator-level granularity. It also offers a set of PyTorch-based APIs to allow users to flexibly specify the input backbone model. AutoMTL then performs gradient-based architecture searches on the supermodel to identify the optimal sharing patterns among tasks. Our experiments on several MTL benchmarks with a different number of tasks show that AutoMTL could produce efficient multi-task models with smaller memory footprint and higher task accuracy compared to state-of-the-art methods. AutoMTL also automatically supports arbitrary CNN backbones without any re-implementation efforts and improves the accessibility of MTL to general programmers.

The main contributions of our work are as follows:

- We propose a *Multi-Task Supermodel Compiler* (MTS-Compiler), which transforms a user-provided backbone CNN into a multi-task supermodel that encodes the architecture search space. The compiler decouples architecture search with the backbone CNN, removing the manual efforts in re-implementing MTL on new backbone models.
- We propose a *Stretchable Architecture Search Space* that offers flexibility in deriving multi-task models with a wide range of model capacity based on task difficulties and interference. We further propose a novel data structure called *Virtual Computation Node* to encode the search space and enable compiler-based multi-task supermodel transformation.
- Built on top of the supermodel, we adopt the Gumbel-Softmax approximation and standard back-propagation to jointly optimize sharing policies and network weights. Under this context, we propose *a policy regularization term* on the sharing policy to promote parameter sharing for high memory efficiency.
- We implement the AutoMTL system that seamlessly integrates the MTS-Compiler, a set of PyTorch-based APIs, the architecture search algorithm, and a training pipeline. AutoMTL provides an easy-to-use solution for resource-efficient multi-task model development.
- Experiments on three popular MTL benchmarks (CityScapes [8], NYUv2 [38], Tiny-Taskonomy [44]) using three common CNNs (Deeplab-ResNet34 [6], MobileNetV2 [37], MNasNet [41]) demonstrate that the multi-task model produced by AutoMTL outperforms state-of-the-art approaches in terms of model size and task accuracy.

2 Related Work

Multi-task learning (MTL) tackles a set of tasks jointly by learning a multi-task model for all tasks. Compared to single-task learning, it can achieve higher prediction accuracy and improved efficiency for each task by leveraging commonalities across related tasks [4, 2, 35]. MTL uses hard or soft

parameter sharing [35]. In hard parameter sharing, a set of parameters in the backbone model are shared among tasks. In soft parameter sharing [30, 36, 16], each task has its own set of parameters. Task information is shared by applying regularization on parameters during training (e.g., enforcing the weights of the model for each task to be similar).

In this paper, we focus on hard parameter sharing as it produces memory-efficient multi-task models. One of the widely-used hard parameter sharing strategies is proposed by Caruana [5, 4], which shares the bottom layers of a model across tasks. Following this paradigm, researchers [28, 31, 40, 7, 24] manually design multi-task models that share several bottom layers and then branch out at an ad hoc point for some tasks. Due to the enormous architecture search space, it is unclear how to effectively decide what parameters to share given a backbone model with a set of tasks in interest.

Inspired by the recent progress on Neural Architecture Search (NAS) [47, 34, 14, 33], recent works attempt to *learn* the sharing patterns across tasks. Deep Elastic Network (DEN) [1] uses reinforcement learning (RL) to determine whether each filter in convolutional layers can be shared across tasks. Similarly, AdaShare [39] learns task-specific policies that select which network blocks to execute for a given task. However, these works cannot dynamically adapt multi-task model capacity based on given tasks. BMTAS [3] and Learn to Branch [17] focus on constructing tree-like structures for multi-task models, which has less flexibility than DEN and AdaShare in terms of the search space. Some other works [15, 43] adopt NAS techniques to explore feature fusion opportunities across tasks. However, their primary goal is to improve task accuracy instead of memory efficiency. Also, their general architecture search space requires substantial search time.

More importantly, existing studies focus on search algorithms but ignore the ease of programming and extensibility. Their implementation is highly coupled with specific backbone models. It is time-consuming for general programmers to re-implement the algorithms once the backbone models change, hindering their adoption in broader applications. Although AutoKeras [21] is able to generate multi-task models automatically for given tasks, it limits all tasks to share the entire backbone architecture for feature extraction. The underlying architecture search process aims at choosing a suitable backbone model, instead of learning adaptive sharing patterns for multiple tasks.

3 AutoMTL

AutoMTL allows users to provide an arbitrary backbone CNN and a set of vision tasks, and then automatically generate a multi-task model with high accuracy and low memory footprint by sharing parameters among tasks. Figure 1 illustrates the workflow of AutoMTL. Given a backbone model (Figure 1(a)), a user can specify the model using either the AutoMTL APIs or in the *prototxt* format (Figure 1(b)). The model specification will be parsed by the *MTS-compiler* to generate a multi-task supermodel that encodes the entire search space (Figure 1(c)). AutoMTL then identifies the optimal multi-task model architecture (Figure 1(d)) from the supermodel using gradient-based architecture search algorithms implemented in the *Architecture Search* component. AutoMTL supports the model specification in the format of *prototxt* as it is general enough to support various CNN architectures and also simple for our compiler to analyze. *prototxt* files are serialized using Google's Protocol Buffers serialization library. AutoMTL API is currently implemented on top of PyTorch. We next elaborate on the two major components of this framework, *MTS-Compiler* and *Architecture Search*.

3.1 Multi-Task Supermodel Compiler

The Multi-Task Supermodel Compiler (MTS-Compiler) transforms the input backbone model into a *multi-task supermodel* that encodes the architecture search space. The challenge is how to design the architecture search space and the multi-task supermodel so that (1) the search space allows the multi-task model capacity to be *flexibly adjusted* based on the set of tasks to avoid task interference and (2) the transformation can be *fully automated* and support an arbitrary CNN backbone.

To address the challenge, we propose a *Stretchable Architecture Search Space* that contains multi-task models with a wide range of model capacities by treating each operator in the backbone model as the basic unit for sharing. The compiler duplicates each operator in the backbone model so that each task can determine whether it wants to share parameters with other tasks by selecting which operator to use. We further design a novel data structure called *Virtual Computation Node* to embed the

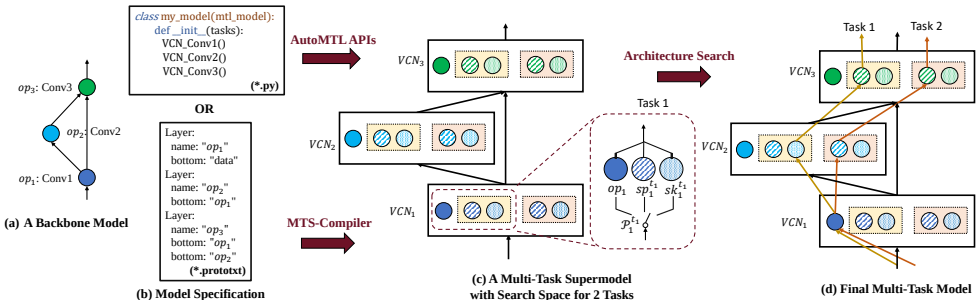


Figure 1: Illustrations of (a) an input backbone model, (b) two types of model specifications, (c) the multi-task supermodel with search space produced by our AutoMTL APIs and the proposed MTS-Compiler, and (d) the final multi-task model found by AutoMTL. $VCN_1 \sim VCN_3$ represent the proposed data structure Virtual Computation Node (VCN). In each VCN, op is the original operator in the backbone model, while sp^{t_i} and sk^{t_i} are the task-specific copy of op and the skip connection for task t_i respectively. $\mathcal{P}^{t_i}$ is a variable (a.k.a policy) that determines which operator will be executed for task t_i.

search space and enable compiler-based automatic transformation of an arbitrary CNN to a multi-task supermodel.

Stretchable Architecture Search Space. As shown in Figure 1(c), for each operator in the backbone model, each task can choose from one of the three options to indicate whether it wants to share the operator with other tasks: (1) the *backbone operator*, (2) a *task-specific copy* of the backbone operator, and (3) a *skip connection*. Skip connection is an identify function if the input and output dimension match or a down-sample function otherwise. The motivation for a skip connection option is that a task can skip the operator to improve inference efficiency.

Formally, assume a set of N tasks $\mathbf{T} = \{t_1, t_2, ..., t_N\}$ defined over a dataset. For the i-th task t_i and the l-th backbone operator op_l, the task may select the backbone operator itself, implying that it can share the parameters in this operator with other tasks. Otherwise, it can select the task-specific copy $sp_l^{t_i}$, or the skip connection $sk_l^{t_i}$, as shown in Figure 1(c). Given a set of N tasks and a backbone model with L operators, the size of our search space is $3^{N \times L}$. Suppose the backbone capacity is C (measured by the number of parameters), the capacity of a multi-task model in our search space would be in the range of $(0, C \times N]$, where $C \times N$ indicates all tasks choose to use their own operators (i.e., independent models) while 0 represents all the skip connections are selected.

The proposed search space has the following three major benefits compared to existing NAS-based MTL methods. First, compared to AdaShare [39] and DEN [1] which attempt to pack multiple tasks into a single CNN backbone, it can extend the representation power of the backbone model if needed by preserving more task-specific operators. This capability effectively avoids performance degradation caused by task interference as the number of tasks increases (See Section 4.2). Second, compared to a more general search space [43] that is defined without requiring a user-provided backbone model, it still provides users a certain degree of control over the size of the searched multi-task model – one can specify a smaller backbone model if the computation resource is limited. Last but not least, in terms of search efficiency, although our search space is larger than AdaShare, we still have a comparable low search cost by adopting gradient-based search algorithms. When comparing to a general search space in FBNetV5 [43], our search space can be explored $13 \sim 133X$ faster (Detailed in Section 4.2).

Multi-Task Supermodel. A suitable multi-task supermodel abstraction is necessary to allow *automatic* transformation of an arbitrary CNN backbone to a multi-task supermodel. Our idea is to represent the multi-task supermodel as a computation graph whose topology remains the same as that of the backbone model but nodes are replaced with Virtual Computation Nodes (VCNs). The MTS-Compiler first parses the input backbone into a list of operators and then iterates the operators to initialize the corresponding VCNs. The multi-task supermodel is a list of VCNs. The detailed compiling procedure is in Appendix Section A.

Specifically, for each operator in the given backbone model, the corresponding VCN in the multi-task supermodel contains: (1) a list of parent VCN nodes, recording where inputs come from; (2) the backbone operator; (3) task-specific copies of the backbone operator, one for each task; (4) skip connections, one for each task; (5) policy variables, one for each task determining which operator to

execute for each task. (1) and (2) encode the computation graph of the backbone model. (2), (3), and (4) together encode the architecture search space. (5) determines the sharing patterns across tasks. Figure 1(c) illustrates a multi-task supermodel. Details about the policy variable will be presented in Section 3.2.

3.2 Architecture Search

The architecture search component aims at efficiently exploring the search space encoded in the multi-task supermodel. We adopt the *differentiable policy approximation* to enable joint training of sharing policy and the supermodel. Under this context, we propose a *policy regularization* mechanism to promote parameter sharing for memory efficiency.

Policy Approximation. We introduce a trainable policy variable $\mathcal{P}_l^{t_i}$ to determine which operator to use for the i-th task t_i and the l-th VCN in the multi-task supermodel. $\mathcal{P}_l^{t_i}$ is zero if the backbone operator op_l is used for the task, one if the task-specific copy $sp_l^{t_i}$ is adopted, and two if the skip connection $sk_l^{t_i}$ is selected. Architecture search is to find the optimal sharing policy $\mathbf{P} = \{\mathcal{P}_l^{t_i} | l \leq L, i \leq N\}$, that yields the best overall performance over the set of N tasks $\mathbf{T}$ given a multi-task supermodel with L VCNs. As the number of potential configurations for $\mathbf{P}$ is $3^{N \times L}$ (i.e., the size of the search space) which grows exponentially with the number of operators and tasks, it is not practical to manually find such a $\mathbf{P}$ to get the optimal sharing pattern. Therefore, we adopt a gradient-based architecture search algorithm that optimizes the sharing policy $\mathbf{P}$ and the multi-task model parameters jointly via standard back-propagation. Gradient-based searches usually allow faster architecture search compared with traditional reinforcement learning [1] or evolutionary algorithm-based approaches [25].

Because the policy variable $\mathcal{P} \in \mathbf{P}$ is discrete and thus non-differentiable, we apply Gumbel-Softmax Approximation [20] and derives a soft differentiable policy:

$$\mathcal{P}'(k) = \frac{\exp((G_k + \log(\pi_k))/\tau)}{\sum_{k \in \{0,1,2\}} \exp((G_k + \log(\pi_k))/\tau)}, \tag{1}$$

where $k \in \{0, 1, 2\}$ represents the three operator options, $G_k \sim Gumbel(0, 1)$ are i.i.d samples drawn from the standard Gumbel distribution, $\pi = [\pi_0, \pi_1, \pi_2]$ is policy distribution ($\pi_0 + \pi_1 + \pi_2 = 1$), and τ is the temperature parameter. As $\tau \to 0$, the softmax computation smoothly approaches the `argmax`, which means that the soft policy $\mathcal{P}'$ approaches the one-hot encoding of policy $\mathcal{P}$. The initial τ is set to 5 and gradually decreased to 0 as in [18, 42].

After learning the distribution π, the discrete task-specific policy $\mathcal{P}$ is sampled from the learned π to decide which operator to execute in each VCN for each task and the multi-task architecture can be constructed accordingly. Figure 1(d) illustrates a multi-task model given a sharing policy.

Policy Regularization. We propose a policy regularization term $\mathcal{L}_{reg}$ to encourage sharing operators across tasks to reduce the memory overhead. Specifically, for the soft policy $\mathbf{P}' = \{\mathcal{P}'^{t_i}_l | l \leq L, i \leq N\}$, we minimize the sum of the SoftPlus [10] of $\mathcal{P}'^{t_i}_l(1) - \mathcal{P}'^{t_i}_l(0)$ and $\mathcal{P}'^{t_i}_l(2) - \mathcal{P}'^{t_i}_l(0)$ for each task in each VCN, where $\mathcal{P}'^{t_i}_l(0), \mathcal{P}'^{t_i}_l(1), \mathcal{P}'^{t_i}_l(2)$ are the probability of selecting the shared operator, the task-specific copy, and the skip connection for the i-th task in the l-th VCN respectively. To further reduce the computation cost, the regularization term is weighted for different operators to promote the parameter sharing of bottom layers. More formally, we define $\mathcal{L}_{reg}$ as,

$$\mathcal{L}_{reg} = \sum_{i \leq N} \sum_{l \leq L} \frac{L - l}{L} \left\{ \ln(1 + \exp^{\mathcal{P}'^{t_i}_l(1) - \mathcal{P}'^{t_i}_l(0)}) + \ln(1 + \exp^{\mathcal{P}'^{t_i}_l(2) - \mathcal{P}'^{t_i}_l(0)}) \right\}, \tag{2}$$

where $\ln(1 + \exp^x)$ is the SoftPlus function. l is the depth of the current VCN and L is the maximum depth. Finally, the overall loss $\mathcal{L}$ is defined as,

$$\mathcal{L} = \sum_i \lambda_i \mathcal{L}_i + \lambda_{reg} \mathcal{L}_{reg}, \tag{3}$$

where $\mathcal{L}_i$ represents the task-specific loss, λ_i is a hyperparameter controlling how much each task contributes to the overall loss, and λ_{reg} is a hyper-parameter balancing task-specific losses and $\mathcal{L}_{reg}$.

Training Pipelines. AutoMTL implements a three-stage training pipeline to generate a well-trained multi-task model. The first stage *pre-train* aims at obtaining a good initialization for the multi-task

supermodel by pre-training on tasks jointly [46]. During pre-training, for each task, the output of each VCN is the average of the three operator options (i.e, the backbone operator, the task-specific copy, and the skip connection) so that all the parameters could get warmed up together. The second stage *policy-train* jointly optimizes the sharing policy and the model parameters. The model parameters and the policy distribution parameters are trained alternately to stabilize the training process. After the policy distribution parameters converge, AutoMTL samples a sharing policy from the distribution to generate a multi-task model. The last stage *post-train* trains the identified multi-task model until it converges. The model parameters are trained from scratch while the sharing policy is fixed.

4 Experiments

We conduct a set of experiments to examine the superiority of AutoMTL compared to several state-of-the-art approaches in terms of task accuracy, model size, and inference time.

4.1 Experiment Settings

Datasets and Tasks. Our experiments use three popular datasets in multi-task learning (MTL), **CityScapes** [8], **NYUv2** [38], and **Tiny-Taskonomy** [44]. CityScapes contains street-view images and two tasks, semantic segmentation and depth estimation. The NYUv2 dataset consists of RGB-D indoor scene images and three tasks, 13-class semantic segmentation defined in [9], depth estimation whose ground truth is recorded by depth cameras from Microsoft Kinect, and surface normal prediction with labels provided in [12]. Tiny-Taskonomy contains indoor images and its five representative tasks are semantic segmentation, surface normal prediction, depth estimation, keypoint detection, and edge detection. All the data splits follow the experimental settings in AdaShare [39].

Loss Functions and Evaluation Metrics. Semantic segmentation uses a pixel-wise cross-entropy loss for each predicted class label. Surface normal prediction uses the inverse of cosine similarity between the normalized prediction and ground truth. All other tasks use the L1 loss. Semantic segmentation is evaluated using mean Intersection over Union and Pixel Accuracy (mIoU and Pixel Acc, the higher the better) in both CityScapes and NYUv2. Surface normal prediction is evaluated using mean and median angle distances between the prediction and the ground truth (the lower the better), and the percentage of pixels whose prediction is within the angles of $11.25°$, $22.5°$ and $30°$ to the ground truth as [12] (the higher the better). Depth estimation uses the absolute and relative errors between the prediction and the ground truth are computed (the lower the better). In addition, the percentage of pixels whose prediction is within the thresholds of $1.25, 1.25^2, 1.25^3$ to the ground truth, i.e. $\delta = \max\{\frac{p_{pred}}{p_{gt}}, \frac{p_{gt}}{p_{pred}}\} < thr$, is used following [13] (the higher the better). Tiny-Taskonomy is evaluated using the task-specific loss of each task directly, as in [39].

Because evaluation metrics from different tasks have different scales, we also use a single **relative performance** metric [29] with respect to the single-task baseline to compare different approaches. The relative performance Δt_i of a method A on task t_i is computed as follows,

$$\Delta t_i = \frac{1}{|M|} \sum_{j=0} (-1)^{s_j} (M_{A,j} - M_{STL,j}) / M_{STL,j} \times 100\%,$$

where s_j is 1 if the metric M_j is the lower the better and 0 otherwise. $M_{A,j}$ and $M_{STL,j}$ are the values of the metric M_j for the method A and the **Single-Task** baseline respectively. Besides, the overall performance is the average of the above relative values over all tasks, namely $\Delta t = \frac{1}{N} \sum_{i=1} \Delta t_i$, where N is the number of tasks. The model size is evaluated using the number of model parameters.

Baselines for Comparison. We compare with following baselines: the **Single-Task** baseline where each task has its own model and is trained independently, the vanilla **Multi-Task** baseline [4] where all tasks share the backbone model but have separate prediction heads, popular MTL methods (e.g., **Cross-Stitch** [30], **Sluice** [36], **NDDR-CNN** [16], **MTAN** [27]), and state-of-the-art NAS-based MTL methods (e.g. **DEN** [1], **AdaShare** [39], and **Learn to Branch**[1] [17]). We use the same backbone model in all baselines and in our approach for fair comparisons. We use Deeplab-ResNet-34 as the backbone model and the Atrous Spatial Pyramid Pooling (ASPP) architecture as the task-specific

[1] We implemented its tree-structured multi-task model for Taskonomy based on the architecture reported in the paper by ourselves since there is no public code.

Table 1: Quantitative Results on CityScapes. (Abs. Prf. & Rel. Prf.)

Model	# Params ↓		Semantic Seg.			Depth Estimation						Δt ↑
	Abs. (M)	Rel. (%)	mIoU ↑	Pixel Acc. ↑	Δt_1 ↑	Error ↓		δ, within ↑			Δt_2 ↑	
						Abs.	Rel.	1.25	1.25^2	1.25^3		
Single-Task	42.569	-	36.5	73.8	-	0.026	0.38	57.5	76.9	87.0	-	-
Multi-Task	21.285	-50.0	<u>42.7</u>	68.1	+4.6	0.026	0.39	58.8	80.5	89.9	+1.5	+3.1
Cross-Stitch	42.569	+0.0	40.3	74.3	+5.5	**0.017**	<u>0.34</u>	70.0	86.3	<u>93.1</u>	**+17.2**	+11.4
Sluice	42.569	+0.0	39.8	74.2	+4.8	<u>0.018</u>	0.35	68.9	85.8	92.8	+15.3	+10.1
NDDR-CNN	44.059	+3.5	41.5	74.2	<u>+7.1</u>	<u>0.018</u>	0.35	69.9	86.3	93.0	+15.9	<u>+11.5</u>
MTAN	51.296	+20.5	40.8	74.3	+6.2	**0.017**	0.36	<u>71.0</u>	86.3	92.8	+16.4	+11.3
DEN	23.838	-44.0	38.0	74.2	+2.3	<u>0.018</u>	0.41	68.2	84.5	91.6	+11.3	+6.8
AdaShare	21.285	-50.0	40.6	<u>74.7</u>	+6.2	<u>0.018</u>	0.37	**71.4**	**86.8**	<u>93.1</u>	+15.5	+10.9
AutoMTL	28.819	-32.3	**42.8**	**74.8**	**+9.3**	<u>0.018</u>	**0.33**	70.0	<u>86.6</u>	**93.4**	<u>+17.1</u>	**+13.2**

Table 2: Results on NYUv2. (Rel. Prf.)

Model	# Params (%) ↓	Δt_1 ↑	Δt_2 ↑	Δt_3 ↑	Δt ↑
Multi-Task	-66.7	-11.4	+2.0	+4.3	-1.7
Cross-Stitch	+0.0	-2.6	+8.7	+3.9	+3.3
Sluice	+0.0	-6.2	+7.1	+3.3	+1.4
NDDR-CNN	+5.0	-12.9	+7.0	-4.4	-3.5
MTAN	+3.7	<u>-1.8</u>	**+11.5**	+2.9	<u>+4.2</u>
DEN	-62.7	-7.7	+5.6	-38.9	-13.7
AdaShare	-66.7	-4.3	<u>+9.3</u>	<u>+6.2</u>	+3.8
AutoMTL	-45.1	**+0.2**	+8.0	**+7.8**	**+5.3**

Table 3: Results on Taskonomy. (Rel. Prf.)

Models	# Params (%) ↓	Δt_1 ↑	Δt_2 ↑	Δt_3 ↑	Δt_4 ↑	Δt_5 ↑	Δt ↑
Multi-Task	-80.0	-3.7	-1.4	<u>-4.5</u>	+0.0	+4.2	-1.1
Cross-Stitch	+0.0	+0.9	-3.5	**+0.0**	-1.0	-2.4	-1.2
Sluice	+0.0	-3.7	-1.5	-9.1	+0.5	+2.4	-2.3
NDDR	+8.2	-4.2	-0.9	<u>-4.5</u>	+0.5	+4.2	-1.0
MTAN	-9.8	-8.0	-2.5	<u>-4.5</u>	+0.0	+2.8	-2.4
DEN	-77.6	-28.2	-2.6	-22.7	<u>+2.5</u>	+4.2	-9.3
AdaShare	-80.0	+2.3	-0.6	<u>-4.5</u>	**+3.0**	<u>+5.7</u>	<u>+1.2</u>
Learn to B.	-71.2	**+9.4**	<u>+5.3</u>	<u>-4.5</u>	-2.5	-2.4	+1.1
AutoMTL	-50.1	<u>+3.0</u>	**+8.2**	**+0.0**	**+3.0**	**+7.1**	**+4.3**

t_1: Semantic Seg., t_2: Surface Normal, t_3: Depth Est., t_4: Keypoint Det., t_5: Edge Det..

head [6]. Both of them are popular architectures for pixel-wise prediction tasks. We also evaluate the effectiveness of AutoMTL on MobileNetV2 [37], and MNasNet [41].

4.2 Results

Performance Comparison. Table 1∼3 report the task performance on each dataset respectively. For CityScapes, both the absolute and the relative performance of all metrics are reported (see Table 1). Due to the limited space, only the relative performance is reported for NYUv2 and Tiny-Taskonomy (see Table 2 and 3). Details of full comparisons can be found in Appendix Section C.

According to Table 1, AutoMTL outperforms all the baselines on 4 metrics (bold) and is the second-best on 2 metrics (underlined) in terms of task performance. With 17.7% increase in the number of model parameters, the task performance of AutoMTL is far better than the vanilla Multi-Task baseline. Compared to the soft-parameter sharing methods, Cross-Stitch, Sluice, and NDDR-CNN, which are unable to reduce the memory overhead, AutoMTL could achieve higher task performance with fewer parameters. When comparing with DEN and AdaShare, the most competitive approaches in MTL, AutoMTL is better in terms of the task performance but with more parameters ($11.7\%/17.7\%$). This is because, unlike DEN and AdaShare which pack tasks into the given backbone model, our search space allows each task to select more task-specific operators to increase the capability of the backbone model. It turns out that a small amount of increase in model parameters could translate to a significant gain in task performance.

The superiority of AutoMTL can be observed more clearly in Tables 2∼3 when more tasks are jointly trained together. AutoMTL outperforms most of the baselines in both task performance and model size. When compared with state-of-the-art NAS-based MTL methods, DEN, AdaShare, and Learn to Branch, AutoMTL could achieve a substantial increase in task accuracy with only a few more parameters. Although DEN and AdaShare need fewer model parameters, the representation power of their multi-task models is limited by the backbone model due to their non-stretchable search space, making it essential for users to select a suitable backbone model with sufficient capacity for multiple tasks. This problem goes worse when the number of tasks increases. As shown in Table 3, DEN and AdaShare have limited task performance improvement or even suffer from task performance degradation (see columns Δt_2 and Δt_3) on the Taskonomy dataset with five tasks. Similarly, the

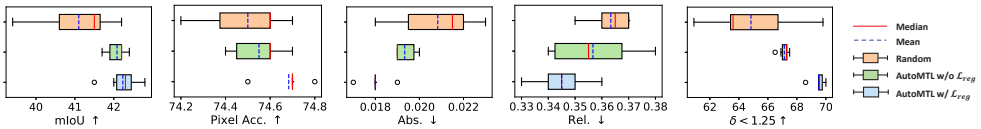
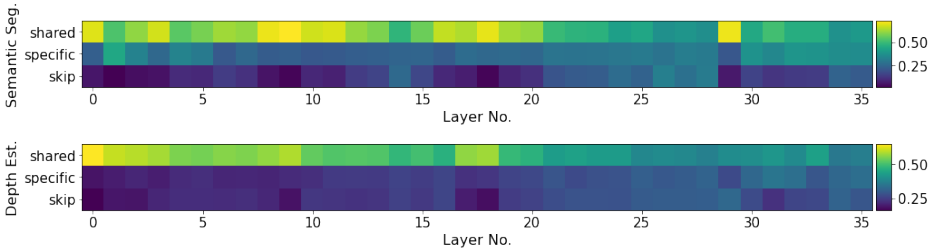
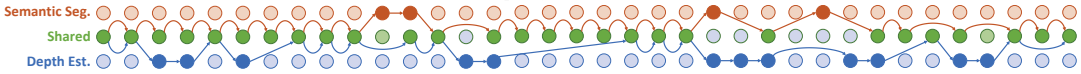


Figure 2: **Ablation Study on CityScapes.** The figures show the distributions of different evaluation metrics for three groups of multi-task models. The orange bar corresponds to the group of models generated from *Random* policies; the green and blue bars correspond to those sampled from the trained policy with or without the policy regularization (*AutoMTL w/o* $\mathcal{L}_{reg}$ and *AutoMTL w/* $\mathcal{L}_{reg}$).



(a) Learned policy distributions



(b) Sampled feature sharing pattern

Figure 3: Policy Visualization for CityScapes.

tree-like multi-task model search space in Learn to Branch limits the flexibility of the sharing patterns in its generated multi-task model, causing severe task interference as shown in columns Δt_3 to Δt_5 in Table 3. In contrast, AutoMTL could generate a multi-task model with larger capacity if necessary, leading to higher task performance, 13.6% higher than DEN, 3.1% than AdaShare, and 3.2% than Learn to Branch.

Furthermore, for semantic segmentation in NYUv2 (see columns Δt_1 in Table 2) and surface normal prediction in Taskonomy (see columns Δt_2 in Table 3), the performance of almost all the multi-task baselines are worse than the Single-Task baseline. It indicates that this particular task is negatively interfered by the other tasks when sharing parameters across them. In contrast, AutoMTL is still able to improve the performance of the two tasks because they tend to select more task-specific operators in our search space in order to reduce interference from the other tasks. Such sharing pattern can be observed from the learned policy distributions for NYUv2 and Taxonomy in Appendix Section F.

We also compare the inference time of different multi-task models in Table 4. AutoMTL could achieve a shorter inference time than independent models, soft-parameter sharing methods, and DEN. It is because AutoMTL allows some computation reuse when consecutive initial layers are shared among tasks and uses skip connections to further reduce the computation overhead. AutoMTL also achieves a competitive inference speed compared with AdaShare even though we provide additional task-specific options in the multi-task models.

Table 4: Inference Time (ms).

Model	CityScapes (2 tasks)	NYUv2 (3 tasks)
Ind. Models	71.01	107.65
Multi-Task	29.52	32.43
Cross-Stitch	71.01	107.65
Sluice	71.01	107.65
NDDR	67.41	101.89
MTAN	98.99	133.33
DEN	87.05	127.41
AdaShare	56.79	71.06
AutoMTL	60.84	80.41

In addition, the time cost of our architecture search process (the policy-train stage) is 12-13 GPU hours on CityScapes, 36-37 GPU hours on NYUv2, and about 120 GPU hours on Taskonomy using Invidia RTX8000, which are $13 \sim 133X$ faster than FBNetV5 [43] (1600 GPU hours on V100[2]), a state-of-the-art multi-task architecture search framework.

Ablation Studies. We present ablation studies to show the effectiveness of the architecture search process (the policy-train stage) and the proposed policy regularization term (Eq. 2). In Figure 2, we

[2]Since the work is not open-sourced, we have no task performance comparison with it and the search costs here are extracted from the original paper. RTX8000 and V100 have similar computation capability and are hence comparable.

use a boxplot to show the distributions of different evaluation metrics for three groups of multi-task models. The orange group (*Random*) of models are generated from policies that are randomly initialized without policy-train, while the green (*AutoMTL w/o* $\mathcal{L}_{reg}$) and the blue (*AutoMTL w/* $\mathcal{L}_{reg}$) groups are sampled from policies after the policy-train stage. The policy of the blue group is trained with the regularization but that of the green group is not. We generate six different models with seed $10 \sim 60$ in each group to compare their performance with less bias.

We make two main observations from Figure 2. First, both *AutoMTL w/o* $\mathcal{L}_{reg}$ and *AutoMTL w/* $\mathcal{L}_{reg}$ achieve better task performance than *Random* in terms of the mean and the standard deviation of all the evaluation metrics. It indicates that the architecture search process is necessary and effective in predicting a good sharing pattern among tasks. Second, the mean of all metrics for *AutoMTL w/* $\mathcal{L}_{reg}$ is also better than *AutoMTL w/o* $\mathcal{L}_{reg}$, indicating that the proposed policy regularization term plays an important role in improving task performance. It echos the well-recognized benefits of parameter sharing among tasks in reducing overfitting and improving prediction accuracy. Similar ablation studies on NYUv2 are shown in Appendix Section D.

Policy Visualization. We further visualize the learned sharing policies to reveal insights on the discovered multi-task architecture. Figure 3 shows the visualization of the learned policy distribution and the feature sharing pattern on CityScapes. Visualizations for the other datasets are in Appendix Section F. For each layer in each task, Figure 3(a) illustrates its policy distribution π introduced in Section 3.2. A brighter block indicates a higher probability of that operator being selected. The figure indicates that tasks tend to share bottom layers. Besides, Figure 3(b) provides a feature sharing pattern sampled from the learned policy distribution. The red arrows connect the operators used by semantic segmentation and the blue ones correspond to depth estimation. Operators that are not selected are semi-transparent. Overall, semantic segmentation is more likely to share operators with other tasks than depth estimation. Depth estimation has more than 25% of operators are skip connections, implying that this task prefers a more compact model than the backbone. Skip connections and operator sharing among the two tasks decrease the number of parameters in the multi-task model.

Results on Other Backbone Models. We also demonstrate the generality of AutoMTL by conducting experiments on CityScapes with two other typical backbone models MobileNetV2 [37] and MNasNet [41]. Without changing hyperparameters on this dataset, AutoMTL achieves 7.4% and 9.5% higher relative task performance with 33.5% and 35.9% fewer model parameters than the single-task baseline and 6.5% and 11.1% higher relative task performance than the Multi-Task baseline for MobileNetV2 and MNasNet respectively. Detailed results are reported in Appendix Section H.

User Study on Ease-of-Use. There is no re-implementation cost in AutoMTL when the backbone model changes. The compilation of a given backbone specified in prototxt format to a multi-task supermodel takes only $\sim$0.6s. On the contrary, users with proficient PyTorch skills in our user study still expect $20 \sim 40$ hours to complete re-implementation of the state-of-the-art NAS-based MTL approach Adashare [39]. Details are described in Appendix Section H.

5 Conclusion

In this work, we propose the first programming framework AutoMTL that generates compact multi-task models given an arbitrary input backbone CNN model and a set of tasks. AutoMTL features a multi-task supermodel compiler that automatically transforms any given backbone CNN into a multi-task supermodel that encodes the proposed stretchable architecture search space. Then through policy approximation and regularization, the architecture search component effectively identifies good sharing policies that lead to both high task accuracy and memory efficiency. Experiments on three popular multi-task learning benchmarks demonstrate the superiority of AutoMTL over state-of-the-art approaches in terms of task accuracy and model size.

Limitations and Broader Impact Statement. Our research facilitates the adoption of multi-task learning techniques to solve many tasks at once in resource-constraint scenarios. Particularly, we offer the first systematic support for automating efficient multi-task model development for vision tasks. The support of other AI tasks (e.g., NLP tasks) is left as future work. It has a positive impact on applications that tackle multiple tasks such as environment perceptions for autonomous vehicles and human-computer interactions in robotic, mobile, and IoT applications. The negative social impact of our research is difficult to predict since it shares the same pitfalls with general deep learning techniques that suffer from dataset bias, adversarial attacks, fairness, etc.

References

[1] Chanho Ahn, Eunwoo Kim, and Songhwai Oh. Deep elastic networks with model selection for multi-task learning. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 6529–6538, 2019.

[2] Jonathan Baxter. A model of inductive bias learning. *Journal of Artificial Intelligence Research*, 12:149–198, 2000.

[3] 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.

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

[5] Richard Caruana. Multitask learning: A knowledge-based source of inductive bias. In *Proceedings of the 10th International Conference on Machine Learning*, pages 41–48. Morgan Kaufmann, 1993.

[6] Liang-Chieh Chen, George Papandreou, Iasonas Kokkinos, Kevin Murphy, and Alan L Yuille. Deeplab: Semantic image segmentation with deep convolutional nets, atrous convolution, and fully connected crfs. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 40(4):834–848, 2017.

[7] Sumanth Chennupati, Ganesh Sistu, Senthil Yogamani, and Samir A Rawashdeh. Multinet++: Multi-stream feature aggregation and geometric loss strategy for multi-task learning. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops*, pages 0–0, 2019.

[8] Marius Cordts, Mohamed Omran, Sebastian Ramos, Timo Rehfeld, Markus Enzweiler, Rodrigo Benenson, Uwe Franke, Stefan Roth, and Bernt Schiele. The cityscapes dataset for semantic urban scene understanding. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 3213–3223, 2016.

[9] Camille Couprie, Clément Farabet, Laurent Najman, and Yann LeCun. Indoor semantic segmentation using depth information. *arXiv preprint arXiv:1301.3572*, 2013.

[10] Charles Dugas, Yoshua Bengio, François Bélisle, Claude Nadeau, and René Garcia. Incorporating second-order functional knowledge for better option pricing. *Advances in Neural Information Processing Systems*, pages 472–478, 2001.

[11] Nikita Dvornik, Konstantin Shmelkov, Julien Mairal, and Cordelia Schmid. Blitznet: A real-time deep network for scene understanding. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 4154–4162, 2017.

[12] David Eigen and Rob Fergus. Predicting depth, surface normals and semantic labels with a common multi-scale convolutional architecture. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 2650–2658, 2015.

[13] David Eigen, Christian Puhrsch, and Rob Fergus. Depth map prediction from a single image using a multi-scale deep network. *arXiv preprint arXiv:1406.2283*, 2014.

[14] Thomas Elsken, Jan Hendrik Metzen, and Frank Hutter. Neural architecture search: A survey. *The Journal of Machine Learning Research*, 20(1):1997–2017, 2019.

[15] 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.

[16] 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.

[17] Pengsheng Guo, Chen-Yu Lee, and Daniel Ulbricht. Learning to branch for multi-task learning. In *International Conference on Machine Learning*, pages 3854–3863. PMLR, 2020.

[18] Yunhui Guo, Honghui Shi, Abhishek Kumar, Kristen Grauman, Tajana Rosing, and Rogerio Feris. Spottune: transfer learning through adaptive fine-tuning. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 4805–4814, 2019.

[19] Junshi Huang, Rogerio S Feris, Qiang Chen, and Shuicheng Yan. Cross-domain image retrieval with a dual attribute-aware ranking network. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 1062–1070, 2015.

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

[21] Haifeng Jin, Qingquan Song, and Xia Hu. Auto-keras: An efficient neural architecture search system. In *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, pages 1946–1956. ACM, 2019.

[22] Brendan Jou and Shih-Fu Chang. Deep cross residual learning for multitask visual recognition. In *Proceedings of the 24th ACM International Conference on Multimedia*, pages 998–1007, 2016.

[23] 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 IEEE Conference on Computer Vision and Pattern Recognition*, pages 6129–6138, 2017.

[24] Isabelle Leang, Ganesh Sistu, Fabian Bürger, Andrei Bursuc, and Senthil Yogamani. Dynamic task weighting methods for multi-task networks in autonomous driving systems. In *2020 IEEE 23rd International Conference on Intelligent Transportation Systems (ITSC)*, pages 1–8. IEEE, 2020.

[25] Jason Liang, Elliot Meyerson, and Risto Miikkulainen. Evolutionary architecture search for deep multitask networks. In *Proceedings of the Genetic and Evolutionary Computation Conference*, pages 466–473, 2018.

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

[27] Shikun Liu, Edward Johns, and Andrew J Davison. End-to-end multi-task learning with attention. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 1871–1880, 2019.

[28] Mingsheng Long, Zhangjie Cao, Jianmin Wang, and S Yu Philip. Learning multiple tasks with multilinear relationship networks. In *Advances in Neural Information Processing Systems*, pages 1594–1603, 2017.

[29] Kevis-Kokitsi Maninis, Ilija Radosavovic, and Iasonas Kokkinos. Attentive single-tasking of multiple tasks. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 1851–1860, 2019.

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

[31] Vladimir Nekrasov, Thanuja Dharmasiri, Andrew Spek, Tom Drummond, Chunhua Shen, and Ian Reid. Real-time joint semantic segmentation and depth estimation using asymmetric annotations. In *2019 International Conference on Robotics and Automation (ICRA)*, pages 7101–7107. IEEE, 2019.

[32] Rajeev Ranjan, Vishal M Patel, and Rama Chellappa. Hyperface: A deep multi-task learning framework for face detection, landmark localization, pose estimation, and gender recognition. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 41(1):121–135, 2017.

[33] Esteban Real, Alok Aggarwal, Yanping Huang, and Quoc V Le. Regularized evolution for image classifier architecture search. In *Proceedings of the Conference on Artificial Intelligence*, volume 33, pages 4780–4789, 2019.

[34] Esteban Real, Sherry Moore, Andrew Selle, Saurabh Saxena, Yutaka Leon Suematsu, Jie Tan, Quoc V Le, and Alexey Kurakin. Large-scale evolution of image classifiers. In *International Conference on Machine Learning*, pages 2902–2911. PMLR, 2017.

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

[36] Sebastian Ruder, Joachim Bingel, Isabelle Augenstein, and Anders Søgaard. Latent multi-task architecture learning. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 33, pages 4822–4829, 2019.

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

[38] Nathan Silberman, Derek Hoiem, Pushmeet Kohli, and Rob Fergus. Indoor segmentation and support inference from rgbd images. In *European Conference on Computer Vision*, pages 746–760. Springer, 2012.

[39] 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.

[40] Mihai Suteu and Yike Guo. Regularizing deep multi-task networks using orthogonal gradients. *arXiv preprint arXiv:1912.06844*, 2019.

[41] Mingxing Tan, Bo Chen, Ruoming Pang, Vijay Vasudevan, Mark Sandler, Andrew Howard, and Quoc V Le. Mnasnet: Platform-aware neural architecture search for mobile. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 2820–2828, 2019.

[42] 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.

[43] Bichen Wu, Chaojian Li, Hang Zhang, Xiaoliang Dai, Peizhao Zhang, Matthew Yu, Jialiang Wang, Yingyan Lin, and Peter Vajda. Fbnetv5: Neural architecture search for multiple tasks in one run. *arXiv preprint arXiv:2111.10007*, 2021.

[44] Amir R Zamir, Alexander Sax, William Shen, Leonidas J Guibas, Jitendra Malik, and Silvio Savarese. Taskonomy: Disentangling task transfer learning. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 3712–3722, 2018.

[45] Arber Zela, Thomas Elsken, Tonmoy Saikia, Yassine Marrakchi, Thomas Brox, and Frank Hutter. Understanding and robustifying differentiable architecture search. In *International Conference on Learning Representations*, 2020.

[46] Xinbang Zhang, Zehao Huang, Naiyan Wang, Shiming Xiang, and Chunhong Pan. You only search once: Single shot neural architecture search via direct sparse optimization. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 43(9):2891–2904, 2020.

[47] Barret Zoph and Quoc V Le. Neural architecture search with reinforcement learning. *arXiv preprint arXiv:1611.01578*, 2016.

Checklist

1. For all authors...
   (a) Do the main claims made in the abstract and introduction accurately reflect the paper's contributions and scope? [Yes]
   (b) Did you describe the limitations of your work? [Yes] See Section 5
   (c) Did you discuss any potential negative societal impacts of your work? [Yes] See Section 5
   (d) Have you read the ethics review guidelines and ensured that your paper conforms to them? [Yes]
2. If you are including theoretical results...
   (a) Did you state the full set of assumptions of all theoretical results? [N/A]
   (b) Did you include complete proofs of all theoretical results? [N/A]
3. If you ran experiments...
   (a) Did you include the code, data, and instructions needed to reproduce the main experimental results (either in the supplemental material or as a URL)? [No] We will publish the code on Github in the future.
   (b) Did you specify all the training details (e.g., data splits, hyperparameters, how they were chosen)? [Yes] See Section 4.1 and Section B in Appendix.
   (c) Did you report error bars (e.g., with respect to the random seed after running experiments multiple times)? [Yes] See Section 4.2 Ablation Studies.
   (d) Did you include the total amount of compute and the type of resources used (e.g., type of GPUs, internal cluster, or cloud provider)? [Yes] See the last paragraph in Section 4.2.
4. If you are using existing assets (e.g., code, data, models) or curating/releasing new assets...
   (a) If your work uses existing assets, did you cite the creators? [Yes] See Section 4.1.
   (b) Did you mention the license of the assets? [N/A] Our experiments were conducted on publicly available datasets.
   (c) Did you include any new assets either in the supplemental material or as a URL? [No] We did not introduce new datasets.
   (d) Did you discuss whether and how consent was obtained from people whose data you're using/curating? [No] Our experiments were conducted on publicly available datasets.
   (e) Did you discuss whether the data you are using/curating contains personally identifiable information or offensive content? [No] We are not aware of relevant issues in the data we use.
5. If you used crowdsourcing or conducted research with human subjects...

(a) Did you include the full text of instructions given to participants and screenshots, if applicable? [N/A]
(b) Did you describe any potential participant risks, with links to Institutional Review Board (IRB) approvals, if applicable? [N/A]
(c) Did you include the estimated hourly wage paid to participants and the total amount spent on participant compensation? [N/A]

A Compiling Procedure

MTS-Compiler takes as inputs a user-specified backbone model in the format of prototxt and a task list, and then generates a multi-task supermodel represented as a graph of VCNs. Algorithm 1 elaborates the compiling procedure. The $backbone$ is first parsed into a list of operators ops (line 4). Then the compiler will iterate over ops to initialize the corresponding VCNs (line 5-7). The final multi-task supermodel $mtSuper$ is a list of VCNs.

Algorithm 1 Compiling Procedure

1: **Input:** $backbone$: a backbone model in prototxt format; $tasks$: the IDs of tasks to learn.
2: **Output:** $mtSuper$: a multi-task supermodel
3: $mtSuper = [\,]$
4: ops = parse_prototxt($backbone$)
5: **for** op in ops **do**
6: $\quad mtSuper$.append(VCN(op, $tasks$))
7: **end for**
8:
9: **Class** VCN:
10: **function** init($op, tasks$)
11: $\quad self.op$ = op
12: $\quad self.parents = [\,]$
13: $\quad$ **for** p in $op.parentOps$ **do**
14: $\quad\quad self.parents$.append(getVCN(p))
15: $\quad$ **end for**
16: $\quad$ **for** t in $tasks$ **do**
17: $\quad\quad self.sp^t = op$.deepcopy()
18: $\quad\quad self.sk^t$ = SkipConnection()
19: $\quad\quad self.\mathcal{P}^t$ = Gumbel-Softmax([0., 0., 0.])
20: $\quad$ **end for**
21: **end function**

B Hyper-parameter Settings

Table 5 summarizes the hyper-parameters used in training. For CityScapes and NYUv2, AutoMTL spends 10,000 iterations on pre-training the supermodel (pre-train), then 20,000 iterations for training policy and supermodel jointly (policy-train), and finally 30,000 iterations for training the identified multi-task model (post-train). As for Tiny-Taskonomy, the three stages need 20,000, 30,000, 50,000 iterations respectively to converge. The hyper-parameters are chosen by empirical experience in AdaShare [39] and manual search during our experiments.

Table 5: Hyper-parameters for training CityScapes, NYUv2, and Tiny-Taskonomy.

Dataset	weight lr	policy lr	weight lr decay	λ_{seg}	λ_{sn}	λ_{depth}	λ_{kp}	λ_{edge}	λ_{reg}
CityScapes	0.001	0.01	0.5/4,000 iters	1	-	1	-	-	0.0005
NYUv2	0.001	0.01	0.5/4,000 iters	5	20	5	-	-	0.001
Tiny-Taskonomy	0.0001	0.01	0.3/10,000 iters	1	3	2	7	7	0.0005

C Full Comparison of All Metrics on NYUv2 and Taskonomy

The full comparison of all metrics on NYUv2 and Taskonomy are summarized in Table 6 and 7. On NYUv2, AutoMTL could achieve outstanding performance on 7 out of 12 metrics, while on Taskonomy, AutoMTL outperforms all the baselines on almost all the 5 metrics.

D Ablation Studies on NYUv2

The ablation studies are also conducted on NYUv2. The same phenomenon as Section 4.2 Ablation Studies in the main paper can be observed in Figure 4. In short, both the architecture search process and the proposed policy regularization term are indispensable to obtain a feature-sharing pattern with high task performance.

Table 6: Quantitative results on NYUv2. (Abs. Prf.)

Model	# Params (M) ↓	Semantic Seg.		Surface Normal Prediction					Depth Estimation				
		mIoU ↑	Pixel Acc. ↑	Error ↓		θ, within ↑			Error ↓		δ, within ↑		
				Mean	Median	11.25°	22.5°	30°	Abs.	Rel.	1.25	1.25^2	1.25^3
Single-Task	63.855	26.5	**58.2**	17.7	16.3	29.4	**72.3**	**87.3**	0.62	0.24	57.8	85.8	96.0
Multi-Task	21.285	22.2	54.4	17.2	15.8	32.2	70.5	84.8	0.59	0.22	60.9	87.7	96.7
Cross-Stitch	63.855	25.4	57.6	17.2	14.0	41.4	67.7	80.4	0.58	0.23	61.4	88.4	95.5
Sluice	63.855	23.8	56.9	17.2	14.4	38.9	69.0	81.4	0.58	0.24	61.9	88.1	96.3
NDDR-CNN	67.047	21.6	53.9	**17.1**	14.5	37.4	70.9	83.1	0.66	0.26	55.7	83.7	94.8
MTAN	66.217	26.0	57.2	17.2	13.9	**43.7**	70.5	81.9	0.57	0.25	62.7	87.7	95.9
DEN	23.838	23.9	54.9	**17.1**	14.8	36.0	70.6	83.4	0.97	0.31	22.8	62.4	88.2
AdaShare	21.285	24.4	57.8	17.7	**13.8**	42.3	68.9	80.5	0.59	**0.20**	61.3	88.5	96.5
AutoMTL	35.056	**26.6**	**58.2**	17.3	14.4	39.1	70.7	83.1	**0.54**	0.22	**65.1**	**89.2**	**96.9**

Table 7: Quantitative results on Taskonomy. (Abs. Prf.)

Models	# Params (M) ↓	Semantic Seg. ↓	Surface Normal ↑	Depth Est. ↓	Keypoint Det. ↓	Edge Det. ↓
Single-Task	106.424	0.575	0.807	**0.022**	0.197	0.212
Multi-Task	21.285	0.596	0.796	0.023	0.197	0.203
Cross-Stitch	106.424	0.570	0.779	**0.022**	0.199	0.217
Sluice	106.424	0.596	0.795	0.024	0.196	0.207
NDDR-CNN	115.151	0.599	0.800	0.023	0.196	0.203
MTAN	95.994	0.621	0.787	0.023	0.197	0.206
DEN	23.838	0.737	0.786	0.027	0.192	0.203
AdaShare	21.285	0.562	0.802	0.023	**0.191**	0.200
Learn to Branch	30.650	**0.521**	0.850	0.023	0.202	0.217
AutoMTL	53.106	0.558	**0.873**	**0.022**	**0.191**	**0.197**

E Ablation Study on the Training Pipeline

We also conducted additional ablation study on the proposed three-stage training pipeline. Specifically, the necessity of the policy-train stage is verified in Section 4.2 already, so we focus on the pre-train and the post-train stages in this section.

The quantitative results on CityScapes with and without the pre-train stage are shown in Table 8. The results are collected using the same hyper-parameter setting reported in the paper. We can see that AutoMTL with pre-training can obtain higher task performance than the model without pre-training. This observation echoes existing work in NAS [46], which also suggests warming up the supermodel first and then conducting searching. Both ablation studies in [46] and our empirical study demonstrate that such a pre-train stage produces a better initialization for the parameters of the supermodel and eventually results in a more accurate multi-task architecture with a similar amount of parameters.

The post-train stage can either use fine-tuning or training-from-scratch. Table 9 compares the task performance of the two options on the identical sampled multi-task architecture under the same hyper-parameter setting. The results show that re-training the identified multi-task model from scratch would produce higher task performance, which suggests retraining as a better post-train strategy. The observation is consistent with that of the well-known differentiable NAS method DARTS [26], which has become a common practice in recent years [42, 46, 45]. Note that in our paper, we also retrain all baselines from scratch for a fair comparison.

F Policy Visualizations on NYUv2 and Taskonomy

Figure 5 visualizes the learned policy distribution on NYUv2. It can be seen that for top layers near the output, the semantic segmentation and the depth estimation tend to have their own computation operators instead of sharing with other tasks. This trend leads that the two tasks may suffer less from the negative interference between tasks, which becomes a possible explanation of why the model searched by AutoMTL can have better task performance on them than existing methods as analyzed in Section 6.2 in the main paper.

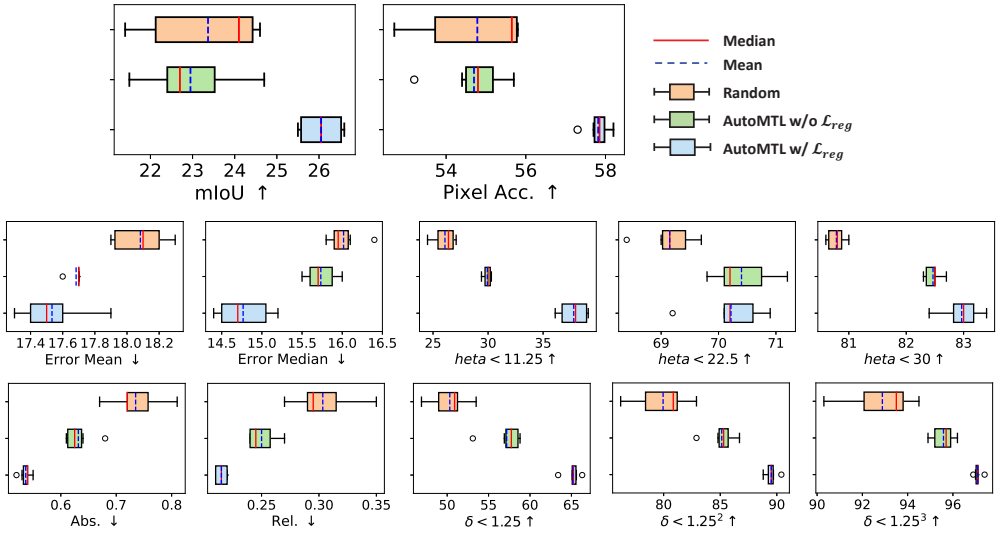


Figure 4: Ablation study on NYUv2. Distributions of different metrics for three groups of multi-task models are exhibited, including models generated from *Random* policies or those sampled from the trained policy with or without the policy regularization (*AutoMTL w/o* $\mathcal{L}_{reg}$ and *AutoMTL w/* $\mathcal{L}_{reg}$).

Table 8: Ablation study about the pre-train stage on CityScapes.

Model	# Params ↓		Semantic Seg.			Depth Estimation						Δt ↑
	Abs. (M)	Rel. (%)	mIoU ↑	Pixel Acc. ↑	Δt_1 ↑	Error ↓ Abs.	Error ↓ Rel.	δ, within ↑ 1.25	δ, within ↑ 1.25^2	δ, within ↑ 1.25^3	Δt_2 ↑	
Single-Task	42.569	-	36.5	73.8	-	0.026	0.38	57.5	76.9	87.0	-	-
AutoMTL w/o pre-train	**28.878**	**-32.1**	41.1	74.5	+6.8	0.020	0.41	65.7	82.7	90.6	+8.2	+7.5
AutoMTL w/ pre-train	30.096	-29.3	**42.8**	**74.8**	**+9.3**	**0.018**	**0.33**	**70.0**	**86.6**	**93.4**	**+17.1**	**+13.2**

Figure 6 visualizes the learned policy distribution on Taskonomy. By comparing the brightness of the three branches in each layer (brighter means higher probability to be chosen), it can be found that the brightness differences between the three branches are more salient in layer No. 0∼10 and No. 30∼35, which indicates that tasks would be more likely to have branch preferences in the top and bottom layers. While for intermediate layers (layer No. 18∼28), the chance of each branch being selected is basically equal. This phenomenon is consistent with traditional multi-task model design principles, which pay more attention to top and bottom layers to decide whether they should be shared or not.

G Task Correlation

We use cosine similarity between task-specific policies to quantify task correlations. Figure 7 illustrates the task correlations in Taskonomy (the darker the higher correlation) and we can make the following observations. (a) Semantic segmentation has a relatively weak correlation with depth estimation compared to other tasks. (b) Surface normal prediction has good correlations with all the other tasks. (c) Depth estimation has low correlations with keypoint and edge detection. (d) Keypoint detection has a strong correlation with edge detection.

H Extension to other Architectures

The users are able to feed any convolution-based model into the proposed MTS-Compiler. We try to demonstrate the superiority of this function by quantifying the manual efforts of using AutoMTL or AdaShare [39] when users switch to different backbones. Specifically, we invited 7 graduate students who are proficient in PyTorch and Machine Learning to convert the backbone model from Deeplab-ResNet34 to MobileNetV2 in both AutoMTL and AdaShare and then record their working time. When using AutoMTL, they firstly spent around 20 minutes reading through our document. After that, all they need to do is to download a MobileNetV2 prototxt from the Internet and use our MTS-Compiler and trainer tools directly. On the other hand, since the public implementation of AdaShare is based on Deeplab-ResNet34, our participants had to re-implement MobileNetV2 as well as

Table 9: Ablation study about the post-train stage on CityScapes.

Model	# Params ↓		Semantic Seg.			Depth Estimation						$\Delta t \uparrow$
						Error ↓		δ, within ↑				
	Abs. (M)	Rel. (%)	mIoU ↑	Pixel Acc. ↑	$\Delta t_1 \uparrow$	Abs.	Rel.	1.25	1.25^2	1.25^3	$\Delta t_2 \uparrow$	
Single-Task	42.569	-	36.5	73.8	-	0.026	0.38	57.5	76.9	87.0	-	-
post-train w/ fine-tune	30.096	-29.3	41.8	74.6	+7.8	0.019	0.38	68.1	84.8	92.2	+12.3	+10.1
post-train w/ retrain	30.096	-29.3	**42.8**	**74.8**	**+9.3**	**0.018**	**0.35**	**69.7**	**85.7**	**92.9**	**+15.6**	**+12.5**

the embedded policy from scratch to fit into the AdaShare training framework. According to the feedback, it generally took $20 \sim 40$ hours to complete coding and debugging.

We also conduct experiments on CityScapes with two other typical backbone models MobileNetV2 [37] and MNasNet [41]. Table 10 and 11 report the task performance when constructing multi-task models on them. AutoMTL always could search out a better multi-task architecture when compared to the vanilla multi-task model.

Table 10: Quantitative results with MobileNetV2 on CityScapes.

Model	# Params (%) ↓	Semantic Seg.			Depth Estimation						$\Delta t \uparrow$
					Error ↓		δ, within ↑				
		mIoU ↑	Pixel Acc. ↑	$\Delta t_1 \uparrow$	Abs.	Rel.	1.25	1.25^2	1.25^3	$\Delta t_2 \uparrow$	
Single-Task	-	25.9	63.5	-	0.043	0.53	32.1	71.1	86.5	-	-
Multi-Task	-50.0	26.3	63.4	**+0.7**	0.042	0.48	33.6	66.5	82.9	+1.2	+0.9
AdaShare	-50.0	**26.7**	61.2	-0.3	**0.032**	**0.46**	42.1	71.6	84.3	+13.6	+6.7
AutoMTL	-33.5	25.8	**63.7**	+0.0	0.035	0.47	**44.4**	**74.8**	**87.3**	**+14.9**	**+7.4**

Table 11: Quantitative results with MNasNet on CityScapes.

Model	# Params (%) ↓	Semantic Seg.			Depth Estimation						$\Delta t \uparrow$
					Error ↓		δ, within ↑				
		mIoU ↑	Pixel Acc. ↑	$\Delta t_1 \uparrow$	Abs.	Rel.	1.25	1.25^2	1.25^3	$\Delta t_2 \uparrow$	
Single-Task	-	**25.5**	63.6	-	0.040	0.49	36.7	73.3	87.3	-	-
Multi-Task	-50.0	25.1	63.5	-0.9	0.040	0.48	35.9	67.8	84.2	-2.2	-1.6
AutoMTL	-35.9	**25.5**	**63.7**	**+0.1**	**0.028**	**0.43**	**53.7**	**77.1**	**87.8**	**+18.9**	**+9.5**

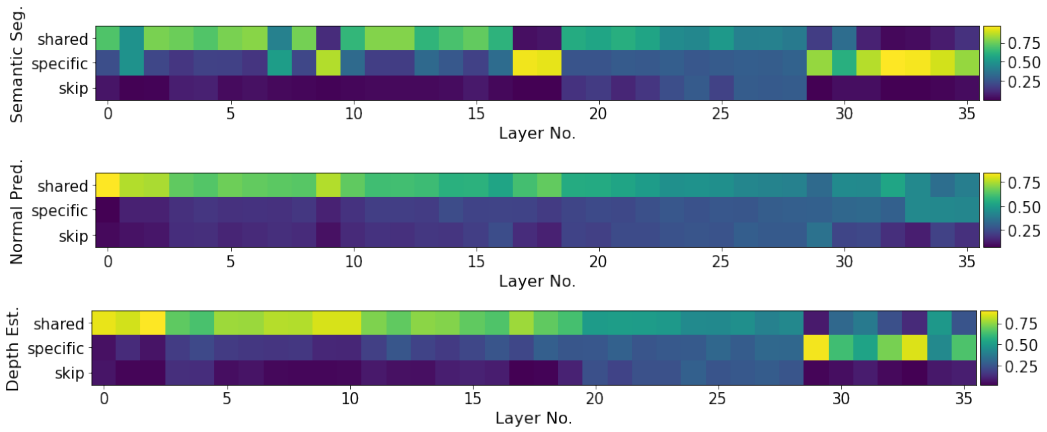


Figure 5: Learned policy distributions for the three tasks in NYUv2.

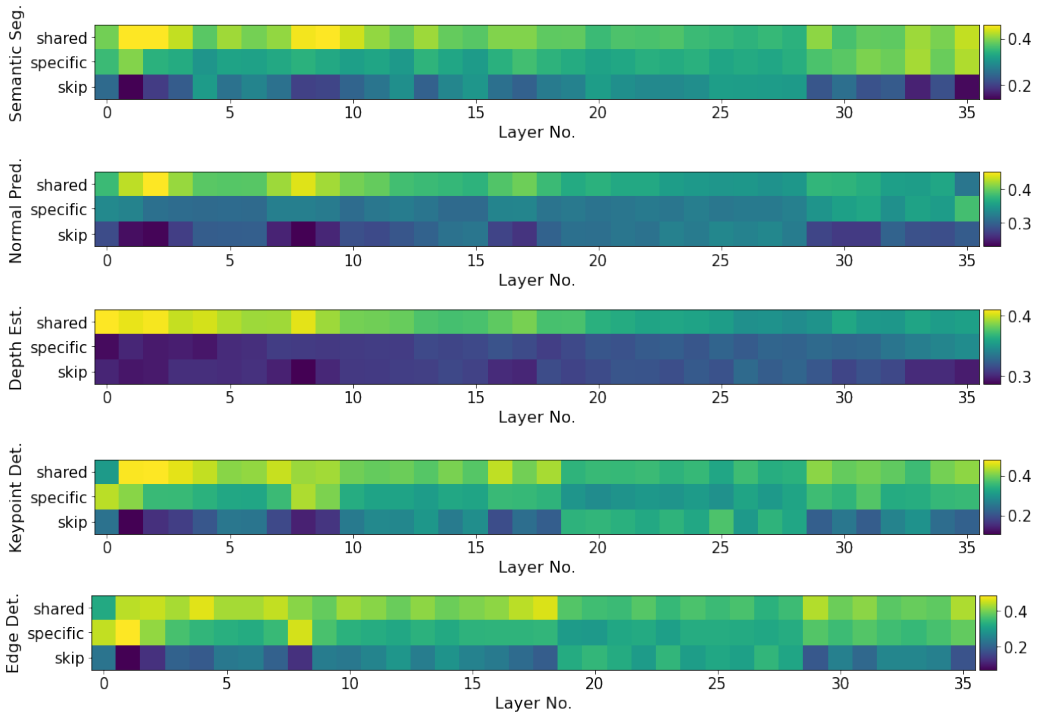


Figure 6: Learned policy distributions for the five tasks in Taskonomy.

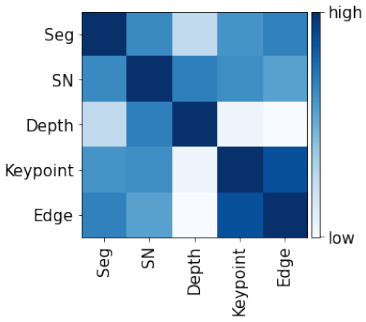


Figure 7: Task correlations in Taskonomy.