UTS UNIVERSITY OF TECHNOLOGY SYDNEY

Learning Architecture for Multiple Tasks in Transfer Learning

by Zhixiong Yue

Thesis submitted in fulfilment of the requirements for the degree of

Doctor of Philosophy

under the supervision of Prof. Yu Zhang and Dr. Christy Liang

University of Technology Sydney
Faculty of Engineering and Information Technology

June 2023

CERTIFICATE OF ORIGINAL AUTHORSHIP

I, Zhixiong Yue, declare that this thesis is submitted in fulfilment of the requirements for the award of Doctor of Philosophy, in the School of Computer Science at the University of Technology Sydney.

This thesis is wholly my own work unless otherwise referenced or acknowledged. In addition, I certify that all information sources and literature used are indicated in the thesis.

I certify that the work in this thesis has not previously been submitted for a degree nor has it been submitted as part of the requirements for a degree at any other academic institution except as fully acknowledged within the text. This thesis is the result of a Collaborative Doctoral Research Degree program with Southern University of Science and Technology.

This research is supported by the Australian Government Research Training Program.

Signature: Production Note:
Signature removed prior to publication.

Date: 2023/06/15

ACKNOWLEDGMENTS

I want to express my deepest gratitude to my principal supervisor at Southern University of Science and Technology, A./Professor Yu Zhang, and my principal supervisor at the University of Technology Sydney, Professor Christy Liang. Without their patience and encouragement, I would never complete this Ph.D. journey. I also want to thank my colleagues and friends both at the Southern University of Science and Technology and the University of Technology Sydney. Their company and help bring me to the end of this journey.

List of Publications

1. **Zhixiong Yue**, Yu Zhang, Jie Liang. Learning Conflict-Noticed Architecture for Multi-Task Learning. *Proceedings of the 2023 National Conference of the American Association for Artificial Intelligence*, Washington, USA, (AAAI 2023). [CORE Rank A*]

2. **Zhixiong Yue**, Baijiong Lin, Yu Zhang, Jie Liang. Effective, Efficient and Robust Neural Architecture Search. *Proceedings of the 2022 International Joint Conference on Neural Networks* (IJCNN 2022), online, 2022. [CORA Rank B]

3. **Zhixiong Yue**, Pengxin Guo, Yu Zhang, Jie Liang. Learning Feature Alignment Architecture for Domain Adaptation. *Proceedings of the 2022 International Joint Conference on Neural Networks* (IJCNN 2022), online, 2022. [CORA Rank B]

4. Mao Lin Huang*, **Zhixiong Yue***, Quang Vinh Nguyen, Jie Liang, Zongwei Luo. Stroke Data Analysis through a HVN Visual Mining Platform. *Proceedings of the 2019 23rd International Conference in Information Visualization* (IV 2019), Adelaide, Australia, 2019. (*Equal Contribution) [CORA Rank B]

5. Feiyang Ye*, Baijiong Lin*, **Zhixiong Yue**, Pengxin Guo, Qiao Xiao, Yu Zhang. Multi-Objective Meta Learning. *Proceedings of the 2021 Advances in Neural Information Processing Systems* (NeurIPS 2021), online, 2021. (*Equal Contribution) [CORA Rank A*]

Under review:

1. **Zhixiong Yue**, Feiyang Ye, Yu Zhang, Jie Liang. Deep Safe Multi-Task Learning. *Will submit to IEEE Transactions on Pattern Analysis and Machine Intelligence* (TPAMI, CORE Rank A*) in July 2022.

Abstract

Recent advance in transfer learning enables deep neural networks to share knowledge between multiple tasks and objectives. However, designing an effective deep neural network for transfer learning relies heavily on human expertise. Even though existing transfer learning methods exhaust their ability to improve the efficiency of the model architecture, manually designing the model architecture may not be realistic for challenging transfer learning problems in real-world scenarios. To summarize, it remains an open question of how to effectively design the transfer learning model architecture.

This thesis focuses on automatically learning the effective architecture of transfer learning models. It addresses four research questions for realistic transfer learning problems: 1) How to design a deep neural network architecture that satisfies multiple objectives; 2) How to design a feature alignment architecture with varying difficulty levels of domain adaptation tasks; 3) How to alleviate the gradient conflict in multi-task learning; 4) How to guarantee that multi-task learning performance is no worse than its single-task counterpart on each task.

To solve problem 1), this thesis presents an effective, efficient, and robust neural architecture search method to design architecture by explicitly balancing the trade-off between the performance, resource consumption, and robustness. We formulate the objective function of the proposed method as a multi-objective bi-level optimization problem and propose an efficient gradient-based algorithm to solve it.

To solve problem 2), this thesis presents a new similarity measure and correspond-

ing alignment architecture search method to learn domain-alignment architecture and domain-invariant feature representation. We further develop the first architecture learning framework for the distance-based domain adaptation method.

To solve problem 3), this thesis presents a compact architecture learning method with good scalability to circumvent negative transfer in multi-task learning, which first introduces purely-specific modules into the search space to mitigate the gradient conflict. The proposed method automatically learns when to switch to purely-specific modules in the tree-structured network architectures when the gradient conflict occurs.

To solve problem 4), this thesis presents a safe multi-task learning model, which learns how to combine the private encoder and public encoder for the downstream private decoder. The proposed method mitigates negative sharing in multi-task learning and improves the performance of all tasks compared with single task learning.

To summarize, this thesis presents a series of architecture learning methods for learning effective architecture that solves realistic transfer learning problems.

TABLE OF CONTENTS

LIST OF FIGURES

FIGURE **Page**

List of Tables

Table **Page**

CHAPTER 1

INTRODUCTION

1.1 Background

Artificial intelligence has demonstrated successful attempts on many real-world problems. Recently in the field's development, deep learning techniques substantially improved the performance of various tasks, such as image classification, image segmentation and language modeling. One of the popular paradigms for deep learning is supervised learning, which relies on labeled training data. However, labeled data may be limited in many scenarios. Hence, transfer learning methods [144], which reduce the dependence on large amount of target domain data by transferring knowledge across domains, stirred a great deal of attention.

Subsequently, researchers extensively studied designing an effective deep learning model to help transfer knowledge and improve generalization. For example, domain

adaptation methods [157] learn a deep learning model on a labeled source domain and share its parameters to perform well on a related target domain. Multi-task learning (MTL) methods [152] jointly learn a parameter sharing model where other tasks can leverage the knowledge learned by a task.

1.2 Motivation

Since the architecture of deep neural networks tends to be very complex in the current era, researchers cannot evaluate every architecture we could design. As a result, the most architecture of transfer learning models is manually designed. However, designing an effective deep neural network for transfer learning relies heavily on human expertise, hindering its wide application. For example, multi-task learning models are hard to design when the task number is enormous, and the task relationship is complicated.

To reduce the dependency of transfer learning models on human expertise, this thesis concludes three unsolved challenges existing methods face and proposes corresponding architecture learning methods to address the four challenges: 1) how to design a deep neural network architecture that satisfies multiple objectives, 2) how to design a feature alignment architecture with varying difficulty levels of domain adaptation tasks, 3) how to alleviate the gradient conflict in multi-task learning, and 4) how to guarantee that multi-task learning performance is no worse than its single-task counterpart on each task. This thesis provides a comprehensive analysis and solutions to these challenges.

1.3 Research Questions and Objectives

This thesis aims to develop a set of methods towards learning architecture in transfer learning and will answer the following research questions:

Research Question 1: *How to design a deep neural network architecture that satisfies multiple objectives?*

Neural network architecture is typically only designed to optimize the accuracy while neglecting other significant objectives, resulting in very limited application scenarios. Performance is not the only factor to be considered in real-world applications. On the contrary, resource consumption and robustness may be more critical. For example, a deep neural network with high computational burden and storage demands is difficult to be deployed to embedded devices (*e.g*. mobile phone and IoT device). Besides, it is well known that the trained neural networks are easily misled by adversarial examples and can not precisely distinguish in- and out-of-distribution samples, making them hard to deploy in safety-sensitive applications such as autonomous driving.

However, it is challenging to balance those trade-offs manually because of the large search space of neural network architecture design. Hence, an open problem exists: Can we find an automatic way to design a robust architecture with competitive performance to be deployed in resource-aware platforms?

Research Question 2: *How to design a feature alignment architecture with varying difficulty levels of domain adaptation tasks?*

Model training is hard for a target domain of interest because collecting enough labeled data is intolerably time-consuming and labor-expensive. One solution is to transfer a deep neural network trained on a data-sufficient source domain to the target domain where only unlabeled data is available. However, this learning paradigm suffers from the shift in data distributions across different domains, which is an obstacle in adapting predictive models for the target task.

Although numerous distance-based DA methods have been proposed, learning the domain-invariant feature representation is still challenging. Distance alone in a high-dimensional space may be challenging to reflect the domain discrepancy adequately [68, 69]. The alignment of hidden feature representations also relies heavily on the network architecture design. However, the network architecture of existing methods is manually designed by experts, hence it is hard to guarantee that hidden feature representations from different domains can be well aligned since the difficulty levels of different DA tasks vary. For instance, a complex task may require a more sophisticated network architecture than an easy task, making a hand-crafted network architecture fail to align tasks with varying difficulty levels, limiting DA methods' capacity and versatility.

Research Question 3: *How to alleviate the gradient conflict in multi-task learning?*

Multi-task learning has been widely used in many applications to enable more efficient learning by sharing part of the architecture across multiple tasks. However, a major challenge is the gradient conflict when optimizing the shared parameters, where the gradients of different tasks could have opposite directions. Directly averaging those

gradients will impair the performance of some tasks and cause negative transfer.

Research Question 4: *How to guarantee that multi-task learning performance is no worse than its single-task counterpart on each task?*

In recent years, multi-task learning attracts much attention due to its good performance in many applications. However, many existing MTL models cannot guarantee that its performance is no worse than its single-task counterpart on each task. Though this phenomenon has been empirically observed by some works, little work aims to handle the resulting problem.

Research Objective 1: *To automatically design architecture by explicitly balancing the trade-off among multiple objectives.* (aims to answer Research Question 1)

Since effectiveness, robustness and computational demands are often considered for deploying a neural network, we will choose the objectives including the validation accuracy for the effectiveness, the number of parameters and FLOPs for the efficiency, the out-of-distribution robustness, and adversarial robustness. We will formulate the entire objective function as a *multi-objective bi-level* optimization problem where the upper-level subproblem is a multi-objective optimization problem by considering the effectiveness, efficiency, and robustness. To solve the resultant problem, we will propose a gradient-based optimization algorithm by combining the multiple gradient descent algorithm and the bi-level optimization algorithm.

Research Objective 2: *To learn feature alignment architecture and domain-invariant feature representations for domain adaptation tasks with varying difficulty levels* (aims to answer Research Question 2)

We will propose a new similarity function to measure the similarity between the source and target domains. A learning model can learn a domain-invariant feature representation by maximizing the population correlation between the source and target domains. Specifically, maximizing population correlation can force the two domains to have similar distributions since the population correlation is the maximum pairwise correlation between source and target samples. To further align hidden feature representations between source and target domains, we will design a reinforcement-based neural architecture search method to learn deep alignment architecture and domain-invariant feature representations for different domain adaptation tasks.

Research Objective 3: *To alleviate the gradient conflict in multi-task learning by learning architectures* (aims to answer Research Question 3)

Different from most existing works that manipulate gradients to mitigate the gradient conflict, we will address gradient conflict from the perspective of architecture learning and propose a conflict-noticed architecture learning method to alleviate the gradient conflict by learning architectures. By introducing purely-specific modules specific to each task in the search space, the CoNAL method can automatically learn when to switch to purely-specific modules in the tree-structured network architectures when the gradient conflict occurs. To handle multi-task problems with a large number of tasks, we will

propose a progressive extension of the CoNAL method.

Research Objective 4: *To achieve safe multi-task learning where no negative sharing occurs* (aims to answer Research Question 4)

We will propose a safe multi-task learning model, which consists of a public encoder shared by all the tasks, private encoders, gates, and private decoders. Specifically, each task has a private encoder, a gate, and a private decoder, where the gate is to learn how to combine the private encoder and public encoder for the downstream private decoder. To improve the scalability, we will further propose an extension to learn a compact architecture via neural architecture search.

1.4 Research Contributions and Innovation

1.4.1 Research Contributions

Contribution 1. An effective, efficient, and robust neural architecture search (E2RNAS) method is proposed to design architecture by explicitly balancing the trade-off among the performance, resource consumption, and robustness.

1) This study provides a practical gradient-based framework for multi-objective neural architecture search and can be seamlessly combined with differentiable architecture search algorithms.

2) This study formulates the objective function of the E2RNAS method as a multi-objective bi-level optimization problem and proposes an efficient gradient-based algo-

rithm to solve it.

Contribution 2. An alignment architecture search with a new similarity measure based on population correlation (AASPC) method is proposed to learn domain-alignment architecture and domain-invariant feature representation.

1) This study proposes a new similarity measure, called population correlation, to measure the domain similarity between the source and target domains.

2) This study provides the first architecture learning framework for distance-based domain adaptation methods. It is also one of few works integrating neural architecture search methods into domain adaptation methods.

Contribution 3. A conflict-noticed architecture learning (CoNAL) method is proposed to alleviate the gradient conflict by learning architectures in multi-task learning.

1) This study first introduces the purely-specific modules in the search space of multi-task architecture, which is different from partially-specific modules in existing methods.

2) This study address the gradient conflict problem in multi-task learning from a new perspective of architecture learning and provide extensive experiments on three challenging domains, including computer vision, natural language processing, and reinforcement learning.

Contribution 4. A simple and effective deep safe multi-task learning (DSMTL) model is proposed to achieve *safe multi-task learning*.

1) This study provides formal definitions for multi-task learning, including *negative sharing*, *safe multi-task learning*, and *partially safe multi-task learning*.

2) This study theoretically proves that the generalization upper-bound of the DSMTL model is lower than that of single-task counterpart.

1.4.2 Research Innovation

Innovation 1. This study is the first to propose a differentiable neural architecture search algorithm that simultaneously optimizes many non-trivial objectives, including robustness (out-of-distribution robustness and adversarial robustness), network performance (accuracy) and network computational demands (number of parameters and FLOPs). Compared to existing differentiable neural architecture search algorithms, E2RNAS can automatically design a robust architecture with competitive performance and balancing computational demands, which is useful to be deployed in resource-aware platforms (e.g. mobile phone and IoT device).

Innovation 2. This study is the first to construct a neural architecture search framework for distance-based domain adaptation methods and propose a new similarity measure PC to measure the domain similarity. Compared to existing distance-based domain adaptation methods which rely on distance metric alone to learn the domain-invariant feature representation with limited capacity and versatility, AASPC can learn deep alignment architecture with different distance metrics. AASPC can handle domain adaption tasks with varying difficulty levels, which is useful in real world scenario.

Innovation 3. This study is the first to address gradient conflict from the perspective of architecture learning and propose CoNAL method to alleviate the gradient conflict by learning architectures. None of existing architecture learning methods for multi-task learning considers to mitigate the gradient conflict during the architecture learning process. CoNAL is the first to introduce purely-specific modules into the search space of learnable architectures and along with conflict-aware learning algorithm to circumvent gradient conflict, which is a major issue in multi-task learning.

Innovation 4. This study is the first to formalize the *negative sharing* problem and propose safe multi-task learning. Compared to many existing MTL models cannot guarantee that its performance is no worse than its single-task counterpart on each task, DSMTL can achieve safe multi-task learning, which is a valuable problem of how to stay safe and avoid negative transfer in multi-task learning.

1.5 Research Significance

This section addresses the research significance of the thesis from both theoretical and practical aspects.

Theoretical significance: This thesis investigates and formally defines the negative sharing phenomenon in multi-task learning. It provides formal definitions for safe multi-task learning and partially safe multi-task learning and builds theoretical foundations for analyzing the generalization bound of multi-task learning model compared with

single-task counterpart. These theoretical results have the potential to inspire future researchers to develop practical multi-task learning methods.

Practical significance: This thesis presents a series of architecture learning methods for learning effective architecture that solves realistic transfer learning problems. Methods proposed in this thesis are validated by real-world datasets. The findings of this research can guide the design of future neural network architecture. Research practitioners can directly use the learned architecture to train models and solve real-world problems. The proposed methods in this research have wild applications in many challenging domains, such as computer vision, natural language processing and reinforcement learning.

1.6 Thesis Organization

Figure1.1 presents a conceptual map demonstrating the structure of this thesis. The chapters of the thesis are organized as follows:

- CHAPTER 2 presents the literature review of three related fields of the thesis, including neural architecture search, domain adaptation, and multi-task learning. It further reveals the limitations of recent research in these fields.
- CHAPTER 3 presents an effective, efficient, and robust neural architecture search method, called E2RNAS. The proposed method designs an architecture by explicitly balancing the trade-off between performance, resource consumption, and robust-

ness. This chapter addresses Research Question 1 to achieve Research Objective 1 while optimizing performance, resource consumption, and robustness objectives.

- CHAPTER 4 presents an alignment architecture search method with population correlation, called AASPC, for domain adaptation. The proposed method first introduces neural architecture search techniques to distance-based domain adaptation methods. This chapter addresses Research Question 2 to achieve Research Objective 2 with a neural architecture search framework.

- CHAPTER 5 presents a conflict-noticed architecture learning method, called CoNAL, to alleviate the gradient conflict by learning architectures. By introducing purely-specific modules specific to each task in the search space, the CoNAL method can automatically learn when to switch to purely-specific modules in the tree-structured network architectures when the gradient conflict occurs. This chapter addresses Research Question 3 to achieve Research Objective 3 from the perspective of architecture learning.

- CHAPTER 6 presents a safe multi-task learning, called DSMTL, model which consists of a public encoder shared by all the tasks, private encoders, gates, and private decoders. Specifically, each task has a private encoder, a gate, and a private decoder, where the gate is to learn how to combine the private encoder and public encoder for the downstream private decoder. This chapter addresses Research Question 4 to achieve Research Objective 4 with a simple and effective model.

- CHAPTER 7 summarises the findings of the studies in this thesis and reveals the

directions for future work.

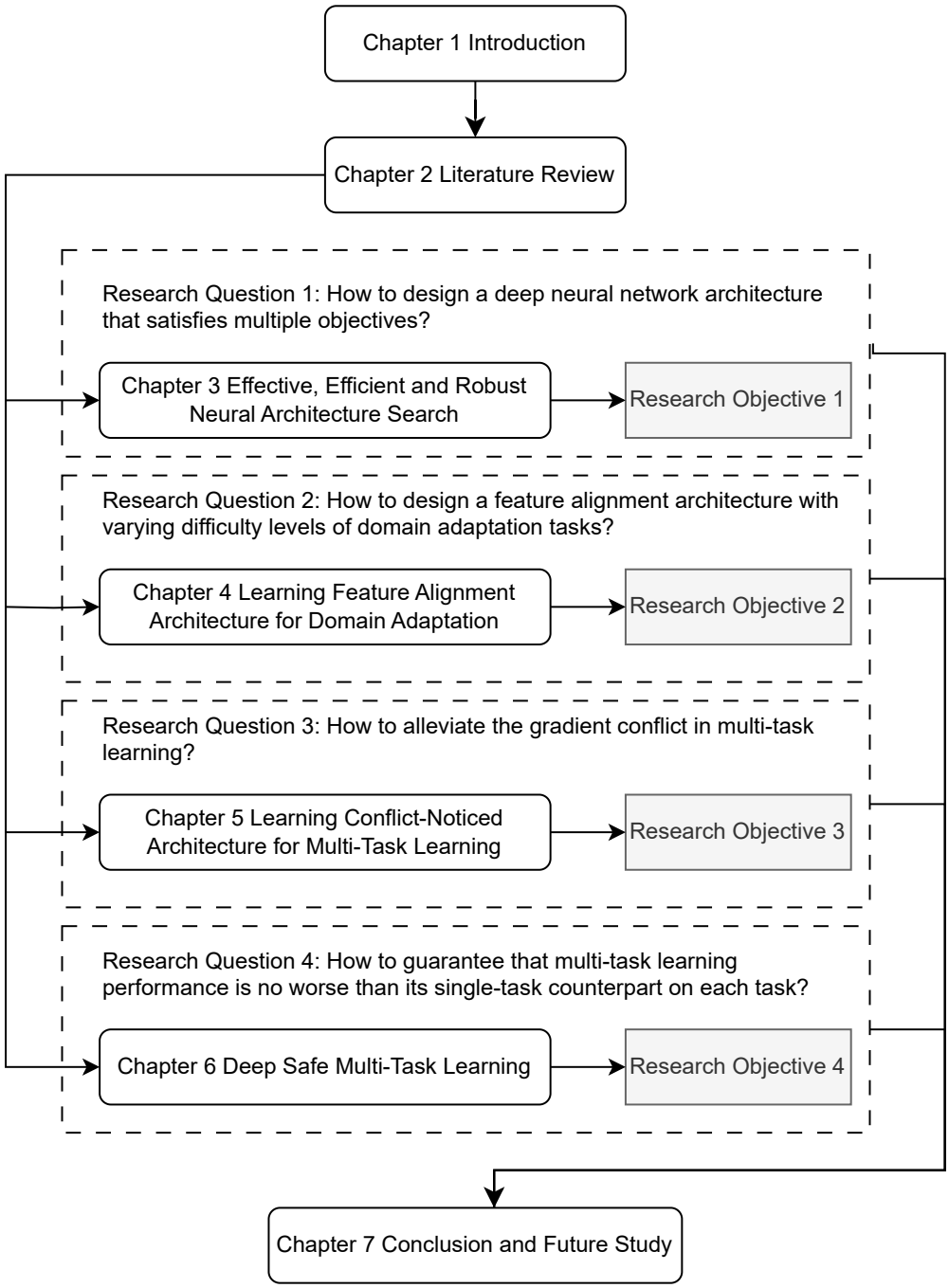


Figure 1.1: Thesis structure overview

CHAPTER 2

LITERATURE REVIEW

This chapter reviews literature related to this thesis, which includes three main parts: neural architecture search, multi-task learning, and domain adaptation.

2.1 Neural Architecture Search

Neural architecture search (NAS) aims to design the architecture of a neural network in an automated way. Compared with the manually designed architecture of neural networks, NAS has demonstrated the capability to find architecture with state-of-the-art performance in various tasks [40, 77, 106]. For example, the NAS-FPN method [40] leverages NAS to learn an effective architecture of the feature pyramid network for object detection.

The architecture generation of NAS can be divided into two parts: the design of

search space, and the architecture optimization. The search space of NAS contains all possible candidate architecture and can be represented as a direct acyclic graph (DAG), where each node and edge indicate a hidden feature and an operation, respectively. After designing the search space, architecture optimization will search for the best architecture from all candidates. There are three categories architecture optimization methods that are commonly used: evolutionary algorithm (EA), reinforcement learning (RL), and gradient descent. According to the category architecture optimization that NAS methods used, NAS methods can be divided into EA-based [32, 72, 90, 108], RL-based [106, 159], and gradient-based [76, 77, 143].

2.1.1 One-shot NAS

Although NAS can achieve satisfactory performance, the high computational cost of the searching procedure makes NAS less attractive. The one-shot NAS methods leverage a supernet to accelerate the search procedure, which contains all the candidate architecture in the search space. In the supernet, weights of operations on edges are shared across different candidate architecture. ENAS [106] employs a reinforcement-based method to train a controller that samples architecture from a supernet with a weight sharing mechanism. DARTS [77] search architecture with a differentiable objective function based on a supernet that uses the softmax function to contain all candidate operations on each edge. The final architecture is determined based on the weights corresponding to the candidate operations on each edge.

2.1.2 One-stage and two-stage NAS

According to the process of NAS, the NAS methods can be divided into one-stage NAS or two-stage NAS. Two-stage NAS methods [77, 106] consist of the searching stage and the retraining stage. The searching stage aims to find optimal network architecture for the search objective and is evaluated on the training dataset. The retraining stage retrains the parameters of the searched architecture from scratch on the training dataset and is evaluated on the testing dataset.

One-stage NAS methods [54, 96] can directly output well-trained neural network models of the searched architecture without retraining. Architecture parameters and model parameters are trained simultaneously, which improves the efficiency of NAS. However, simultaneously optimizing architecture and model parameters may lead to sub-optimal performance since they are highly coupled. To solve this challenge, [10] progressively fine-tuned the smaller networks that search from the shared weights once-for-all network.

2.1.3 Multi-Objective NAS

Multi-objective optimization aims to optimize more than one objective function simultaneously. Among different techniques to solve multi-objective problems, we are interested in gradient-based multi-objective optimization algorithms [27], which leverage the Karush-Kuhn-Tucker (KKT) conditions [62] to find a common descent direction for all objectives. In this paper, we utilize one such method, *i.e*. MGDA [27]. With n objective functions $\{\mathscr{L}_i(\boldsymbol{\theta})\}_{i=1}^{n}$ to be minimized, MGDA is an iterative method by first solving the following

quadratic programming problem as

$$\min_{\gamma_1,\cdots,\gamma_n} \left\| \sum_{i=1}^{n} \gamma_i \nabla_{\boldsymbol{\theta}} \mathscr{L}_i(\boldsymbol{\theta}) \right\|_2^2 \text{ s.t. } \gamma_i \geq 0, \ \sum_{i=1}^{n} \gamma_i = 1, \tag{2.1}$$

where $\|\cdot\|_2$ denotes the ℓ_2 norm of a vector and γ_i can be viewed as a weight for the ith objective, and then minimizing $\sum_{i=1}^{n} \gamma_i \mathscr{L}_i(\boldsymbol{\theta})$ with respect to $\boldsymbol{\theta}$. When convergent, the MGDA can find a Pareto-stationary solution.

Due to the complex application scenarios in the real world, recent works on NAS take multiple objectives instead of only accuracy one objective into consideration, which implicitly indicates a multi-objective optimization problem. Specifically, to search an efficient architecture to be deployed in resource-limited platforms, some workers take the resource-constraint objectives such as the number of parameters, FLOPs, latency, and energy consumption into consideration [5]. Among those works, different techniques are applied to solve this multi-objective optimization problem. For example, some work [32] apply evolutionary algorithms to approximate the entire Pareto front, but the search cost is quite high. Some work [125] regard the combination of multiple targets like accuracy and latency as rewards to optimize the controller sampling an architecture from the search space using reinforcement learning algorithms. The most relevant to our work is the gradient-based method [6, 11, 57], especially the DARTS-based method [6, 57]. GOLD-NAS [6] regards the resource constraint like FLOPs as the regularization terms, whose coefficients gradually increase to prune the architecture during the search procedure. Built on DARTS [77], RC-DARTS [57] considers to search architecture with high accuracy while constraining the model size and FLOPs of the searched architecture within user-defined intervals. Therefore, the proposed objective function is formulated as

a constrained optimization problem, and a projected gradient descent method is applied to solve it.

To search a robust architecture, some works [28, 43] investigate the influence of architecture on adversarial robustness from a NAS perspective and then discover a family of adversarially robust architecture based on their observations. Different with [28, 43], [78] consider adversarial robustness as an optimized objective to search architecture that can defend multiple types of adversarial attacks. This problem is formulated as a multi-objective optimization and solved by an evolutionary algorithm. Besides, similar with [28, 43], [1] study what topology of neural network architecture is best for out-of-distribution robustness.

2.1.4 Limitations

Performance is not the only factor to be considered in real-world applications. On the contrary, resource consumption and robustness may be more critical. For example, a deep neural network with high computational burden and storage demands is difficult to be deployed to embedded devices (*e.g*. mobile phone and IoT device). Besides, it is well known that the trained neural networks are easily misled by adversarial examples and can not exactly distinguish in- and out-of-distribution samples, making them hard to deploy in safety-sensitive applications such as autonomous driving. Hence, there exists an open problem: *Can we find an automatic way to design a robust architecture with competitive performance to be deployed in resource-aware platforms?* This open problem implicitly indicates a trade-off among multiple objectives in NAS. Previous methods do

not consider the trade-off between performance and robustness.

2.2 Domain Adaptation

Domain Adaptation (DA) aims to transfer the knowledge learned from a source domain with labeled data to a target domain without labeled data. Specifically, we study DA under the single source unsupervised setting. That is, there is one source domain and one target domain. Let $\mathbf{x}$ and $\mathbf{y}$ denote the input data and output label, respectively. The source domain $\mathscr{D}_s$ has labeled samples $\{(\mathbf{x}^s, \mathbf{y}^s)\}$ and the target domain $\mathscr{D}_t$ has unlabeled samples $\{\mathbf{x}^t\}$ only. Single-source unsupervised DA methods aim to learn a model from a source domain that can perform well on a different but related target domain. This learning paradigm can cover the target domain without labeled training data, which improves the efficiency of machine learning. However, the main challenge of unsupervised DA is the domain shift between domains, where unreliable predictions may occur on the target domain due to the difference between the source and target distributions [130]. Recent works in unsupervised DA handle the domain shift by learning domain-invariant representations. The basis of these works is justified by the theory of [3]. The performance difference of a classifier between the source and target domain could be bounded by the distance between the data distribution of two domains if the classifier generalizes well from the source domain to the target domain. Recent DA methods that learn domain-invariant representations can be mainly grouped into two categories: discrepancy-based methods and adversarial-based methods.

2.2.1 Discrepancy-based methods

Discrepancy-based methods, which minimize the domain discrepancy between the source and target domains via some measures such as distance. [84] calculate the sum of the multiple kernel variant of maximum mean discrepancies between the fully connected layers and propose a deep adaptation network. [121] propose the correlation alignment by calculating the second-order statistics of the source and target features in the last federated learning layer to minimize the domain shift. [150] propose central moment discrepancy by utilizing the equivalent representation of probability distributions by moment sequences. [132] introduce an adaptation layer and an additional domain confusion loss to learn semantically meaningful and domain-invariant representations.

2.2.2 Adversarial-based methods

Discrepancy-based methods minimize the domain discrepancy between the source and target domains via some measures such as distance. [84] calculate the sum of the multiple kernel variant of maximum mean discrepancies between the fully connected layers and propose a deep adaptation network. [121] propose the correlation alignment by calculating the second-order statistics of the source and target features in the last federated learning layer to minimize the domain shift. [150] propose central moment discrepancy by utilizing the equivalent representation of probability distributions by moment sequences. [132] introduce an adaptation layer and an additional domain confusion loss to learn semantically meaningful and domain-invariant representations.

2.2.3 Limitations

Although numerous DA methods have been proposed, learning the domain-invariant representations is still challenging. In a high-dimensional space, discrepancy measures or adversarial discriminators may be difficult to truly reflect the domain discrepancy. Moreover, all of these methods are developed by using hand-crafted network architecture. Since the difficulty levels of different DA tasks are not the same, accomplishing complex tasks may require a more sophisticated network architecture than easy tasks. Hence, using the same hand-crafted network architecture may limit the capacity and versatility of DA methods.

2.3 Multi-task Learning

MTL [13, 152] aims to improve the generalization performance of multiple learning tasks. Compared with single-task learning, MTL can solve multiple tasks simultaneously, reducing the overall training cost and sharing knowledge from different tasks. Moreover, in some cases, MTL models could make multiple predictions in one forward propagation, which reduces the inference latency of the entire learning model. Therefore, MTL has drawn much attention in recent years. The performance of MTL strongly depends on the use of a proper network architecture [134]. Contrary to the design of a single-task network, the architecture design of multi-task problems often encounters many challenges. Various MTL architecture and parameter sharing schemes have been proposed to improve the performance of the MTL model, which can be divided into four categories:

hard parameter sharing, soft parameter sharing, task routing, and architecture learning.

2.3.1 Hard parameter sharing

In hard parameter sharing methods, the most popular model is the multi-head hard sharing architecture [13], which shares the first several layers among all the tasks and allows the subsequent layers to be specific to different tasks. Based on hard sharing architecture, task balancing methods [18, 58, 139, 147] dynamically assign weights between tasks to balance the loss scale and solve the gradient conflict. With a pre-defined hard sharing network architecture, network capacity might limit these methods and lead to sub-optimal solutions.

2.3.2 Soft parameter sharing

To better handle task relationships, soft parameter sharing methods are proposed. [97] propose a cross-stitch network to combine hidden representations of different tasks linearly. [80] propose a multi-task attention network, which consists of a shared network and an attention module for each task so that both shared and task-specific feature representations can be learned via the attention mechanism. [39] propose a neural discriminative dimensionality reduction layer to enable automatic feature fusing at every layer from different tasks. Such approaches often consist of multiple full-size separate networks, leading to over-fitting on small datasets and too large a model size for deployment.

2.3.3 Task routing

Instead of choosing between soft sharing or hard sharing architecture, task routing methods and other adaptive network methods are proposed. Multi-Agent Reinforcement Learning (MARL) [111] allows the network to dynamically self-organize in response to the input. Task Routing Layer (TRL) [119] allows a single model to fit many tasks with task-specific masks. However, all of these routing-based methods only use one network, which may limit the expressive power of those models. To alleviate this issue, [23] propose an adaptive feature aggregation layer, where a dynamic aggregation mechanism is designed to allow each task to determine the degree of the knowledge sharing between tasks adaptively. [124] propose an adaptive sharing method to learn the sharing pattern through a task-specific policy that selectively chooses which layers to be executed for each task. [133] calculate task affinity scores to construct a branched multi-task network. [34] determines task groupings by co-training all the tasks together and calculating the inter-task affinity. These works dynamically allocate feature sharing or parameter sharing to different tasks. Compared with manually designed architecture, they have better generalization performance but require additional computation to determine task relatedness.

2.3.4 Architecture learning

Instead of the hand-crafting architecture of deep neural networks, NAS [77, 106] can design architecture with good performance automatically. There are some works to leverage NAS to automatically search the architecture of multi-task neural networks

to improve the overall performance of MTL. For example, [89] dynamically widens a multi-layer network to create a tree-like deep architecture, where similar tasks reside in the same branch. [72] propose an evolutionary architecture search algorithm to search blueprints and modules that are assembled into an MTL network. [38] search inter-task layers for better feature fusion across tasks. [45] propose a differentiable architecture search algorithm to learn branching blocks to construct a tree-structured neural network for MTL. [8] automatically determine the branching architecture for the encoder in a multi-task neural network under resource constraints.

2.3.5 Limitations

An important goal in MTL is to improve the performance of all the tasks so that the MTL model could perform no worse than its single-task counterpart on each task, which is called safe multi-task learning [47]. However, most MTL models cannot achieve it with some empirical evidence in [45, 66, 118, 126]. One reason is that some tasks are not highly related to other tasks based on given network architecture. To the best of our knowledge, the only exception is the SMTL model [47] which designs task-private and public encoders for each task via a gating mechanism to achieve safe multi-task learning possibly. However, the model size of the SMTL model grows linearly with respect to the number of tasks, which makes its scalability not so good. The existing architecture learning methods do not consider the negative sharing issue among tasks during the searching process. The searched architecture by those methods may also suffer from negative sharing and thus cannot achieve safe multi-task learning.

2.4 Summary

This chapter comprehensively reviews the related works in the NAS, DA and MTL areas. Notably, it introduces the related works of learning architecture methods in the context of DA and MTL. It describes the advantages of these related works and summarizes the limitations of current methods. In the following chapters, Chapters 3-6, these limitations will be further described and addressed in the corresponding introduction section.

CHAPTER 3

EFFECTIVE, EFFICIENT AND ROBUST NEURAL ARCHITECTURE SEARCH

3.1 Introduction

Deep learning has achieved great successes in many areas, such as computer vision, natural language processing, speech, gaming. The design of the neural network architecture is essential to such success. However, such design relies heavily on experts' knowledge and experience, and even experienced experts cannot design the optimal architecture. Therefore, NAS, which aims to design the architecture of neural networks in an automated way, has attracted great attention in recent years.

Although NAS has demonstrated the capability to find neural network architecture with state-of-the-art performance in various tasks [32, 77, 125], conventional NAS

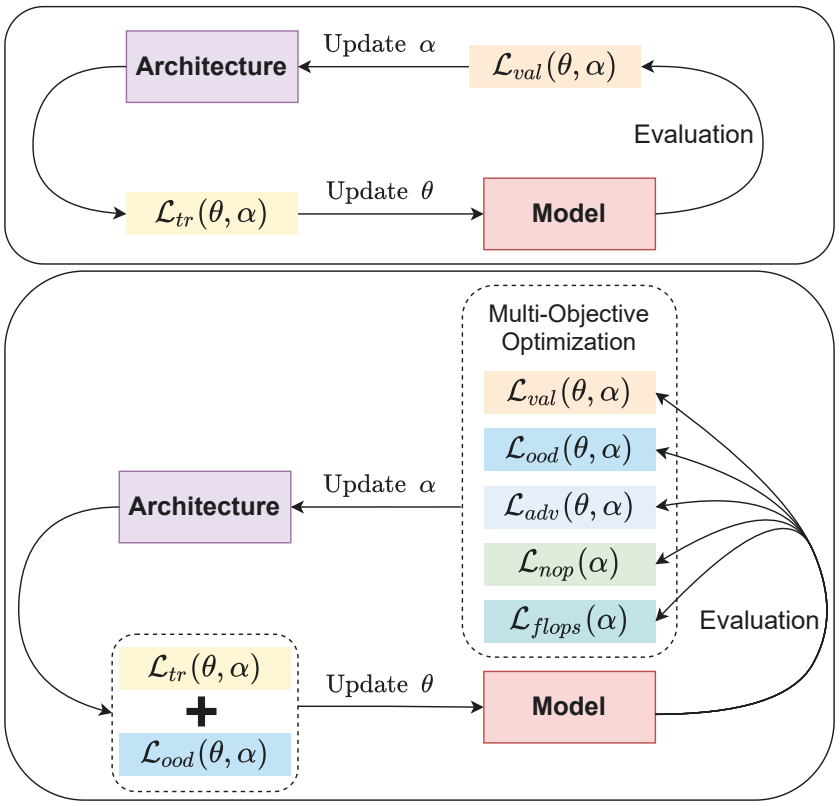


Figure 3.1: Comparison of the architecture searching procedure between DARTS [77] (top) and the proposed E2RNAS (bottom). We formulate E2RNAS as a multi-objective bi-level optimization problem with two key differences with DARTS: 1) we train the model with both in- and out-of-distribution data samples to improve the robustness. 2) we evaluate the E2RNAS model with five objectives, including the validation loss $\mathscr{L}_{val}(\boldsymbol{\theta}, \boldsymbol{\alpha})$ for effectiveness and the number of parameters $\mathscr{L}_{nop}(\boldsymbol{\alpha})$, the number of operations $\mathscr{L}_{flops}(\boldsymbol{\alpha})$ for efficiency, and the out-of-distribution robustness loss $\mathscr{L}_{ood}(\boldsymbol{\theta}, \boldsymbol{\alpha})$ and the adversarial robustness loss $\mathscr{L}_{adv}(\boldsymbol{\theta}, \boldsymbol{\alpha})$ for robustness.

methods are typically only designed to optimize the accuracy during the architecture searching process while neglecting other significant objectives, resulting in very limited application scenarios.

Actually, performance is not the only factor to be considered in real-world applications. On the contrary, resource consumption and robustness may be more critical. For example, a deep neural network with high computational burden and storage demands is difficult to be deployed to embedded devices (*e.g.* mobile phone and IoT device). Besides, it is well

known that the trained neural networks are easily misled by adversarial examples and can not exactly distinguish in- and out-of-distribution samples, making them hard to deploy in safety-sensitive applications such as autonomous driving. Hence, there exists an open problem: *Can we find an automatic way to design a robust architecture with competitive performance to be deployed in resource-aware platforms?*

This open problem implicitly indicates a trade-off among multiple objectives in NAS. Recently, some studies have considered multiple objectives during the architecture searching process. However, most of those works (*e.g*. [5, 57, 125]) only focus on the hardware-aware NAS problem, *i.e*. designing an architecture that can be deployed in resource-limited devices. There exist few studies [1, 28] that statistically investigate the influence of architecture on the robustness, such as the adversarial robustness and out-of-distribution robustness from a NAS perspective. However, they do not consider the trade-off between the performance and robustness.

To answer the open problem, in this chapter, we propose an **E**ffective, **E**fficient, and **R**obust **N**eural **A**rchitecture **S**earch method to design an architecture by explicitly balancing the trade-off among the performance, resource consumption and robustness. Specifically, we consider the validation accuracy for the effectiveness, the number of parameters and FLOPs for the efficiency, and the out-of-distribution robustness and adversarial robustness for the robustness. Built on DARTS, the proposed E2RNAS formulates the entire objective function as a *multi-objective bi-level* optimization problem where the upper-level subproblem is a multi-objective optimization problem by considering the effectiveness, efficiency, and robustness. To solve the resultant problem, we

propose a gradient-based optimization algorithm by combining the Multiple Gradient Descent Algorithm (MGDA) [27] and the bi-level optimization algorithm. In summary, the contributions of this chapter are three-fold.

- We propose the E2RNAS method for searching effective, efficient, and robust network architecture, a practical DARTS-based framework for multi-objective NAS and can be seamlessly combined with DARTS and its variants.
- We formulate the objective function of the E2RNAS method as a multi-objective bi-level optimization problem and propose an efficient gradient-based algorithm to solve it.
- Experiments on benchmark datasets show that the proposed E2RNAS method can find robust architecture with less resource consumption and comparable classification accuracy.

3.2 The E2RNAS Method

In this section, we present the proposed E2RNAS method. We first give an overview of the DARTS method and then introduce how to achieve two kinds of robustness and formulate the objectives to constrain the resource cost, including the number of parameters and FLOPs in the searched architecture. Finally, we present the multi-objective bi-level formulation of the proposed E2RNAS method and its optimization.

3.2.1 Preliminary

DARTS [77] aims to learn a Directed Acyclic Graph (DAG) called cell, which can be stacked to form a neural network architecture. Each cell consists of N nodes $\{d_i\}_{i=0}^{N-1}$, each of which denotes a hidden representation. $\mathscr{O}$ denotes a discrete operation space. The edge (d_i, d_j) of the DAG represents an operation function $o(\cdot)$ (*e.g.* skip connection or 3×3 pooling) from $\mathscr{O}$ with a probability $\alpha_o^{(i,j)}$ to perform at the node d_i. Therefore, we can formulate each edge (d_i, d_j) as a weighted sum function to combine all the operations in $\mathscr{O}$ as $f_{i,j}(d_i) = \sum_{o \in \mathscr{O}} \frac{\exp(\alpha_o^{(i,j)})}{\sum_{o' \in \mathscr{O}} \exp(\alpha_{o'}^{(i,j)})} o(d_i)$. An intermediate node d_j is the sum of its predecessors, *i.e.* $d_j = \sum_{i<j} f_{i,j}(d_i)$. The output of the cell, *i.e.* node d_{N-1}, is the concatenation of all the output of nodes excluding the two input nodes d_0 and d_1. Therefore, $\boldsymbol{\alpha} = \{\alpha_o^{(i,j)}\}_{(i,j) \in \mathbf{E},\, o \in \mathscr{O}}$ can parameterize the searched architecture, where $\mathbf{E}$ denotes the set of all the edges from all the cells. Let $\mathscr{D}_{tr}$ and $\mathscr{D}_{val}$ denote the training dataset and validation dataset, respectively. DARTS is to solve a bi-level optimization problem as

$$\begin{aligned} &\min_{\boldsymbol{\alpha}} \ \mathscr{L}_{val}(\boldsymbol{\theta}^*(\boldsymbol{\alpha}), \boldsymbol{\alpha}) \\ &\text{s.t.} \ \ \boldsymbol{\theta}^*(\boldsymbol{\alpha}) = \arg\min_{\boldsymbol{\theta}} \mathscr{L}_{tr}(\boldsymbol{\theta}, \boldsymbol{\alpha}), \end{aligned} \tag{3.1}$$

where $\boldsymbol{\theta}$ denotes all the weights of the neural network, the average training loss of a neural network is represented by $\mathscr{L}_{tr}(\boldsymbol{\theta}, \boldsymbol{\alpha}) = \frac{1}{|\mathscr{D}_{tr}|} \sum_{(\mathbf{x}, y) \in \mathscr{D}_{tr}} \ell(\boldsymbol{\theta}, \mathbf{x}, y)$ with parameter weight $\boldsymbol{\theta}$ and an architecture $\boldsymbol{\alpha}$ and $\ell(\boldsymbol{\theta}, \mathbf{x}, y)$ denotes the loss function for each sample. $\mathscr{L}_{val}(\boldsymbol{\theta}^*(\boldsymbol{\alpha}), \boldsymbol{\alpha})$ is defined similarly. Here $\min_{\boldsymbol{\alpha}} \mathscr{L}_{val}(\boldsymbol{\theta}^*(\boldsymbol{\alpha}), \boldsymbol{\alpha})$ is called the **Upper-Level (UL)** subproblem and $\min_{\boldsymbol{\theta}} \mathscr{L}_{tr}(\boldsymbol{\theta}, \boldsymbol{\alpha})$ is called the **Lower-Level (LL)** subproblem. When

the search procedure finishes, the final architecture can be determined by the operation with the largest probability in each edge from each cell, *i.e.* , $o^{(i,j)} = \arg\max_{o \in \mathcal{O}} \ \alpha_o^{(i,j)}$.

A well-known issue in using the gradient-based method is the local minima problem, where a solution with a set of parameters gets stuck in a local minimum while the goal is to get to the global minimum. Several techniques have been proposed to mitigate the local minima problem. One common approach is to use stochastic gradient descent, which introduces randomness to the optimization process, allowing the model to escape local minima. Another method is to employ momentum-based optimizers like Momentum[107] or adaptive learning rate algorithms such as Adam[59], which help the optimization process navigate through the parameter space more effectively.

3.2.2 Objective Functions for Robustness

In E2RNAS, we expect the searched architecture to be robust, which means that the trained model with the searched architecture can distinguish test samples whether from in- or out-of-distribution, and its performance is stable when adding some perturbations to the in-distribution samples. To improve the robustness of the searched architecture, we consider both Out-of-Distribution (OoD) robustness and adversarial robustness as objective functions in the UL subproblem.

3.2.2.1 Out-of-Distribution Robustness

Let $\mathscr{C}_{in}$ denote the label distribution of the training dataset $\mathscr{D}_{tr}$ and the validation dataset $\mathscr{D}_{val}$. We use an OoD dataset $\mathscr{D}_{ood}$ with its corresponding label distribution $\mathscr{C}_{out}$

and $\mathscr{C}_{in} \cap \mathscr{C}_{out} = \emptyset$ to measure the OoD robustness of the neural network. Following [67], we formulate the OoD robustness objective function as

$$\mathscr{L}_{ood}(\boldsymbol{\theta}, \boldsymbol{\alpha}) = \frac{1}{|\mathscr{D}_{ood}|} \sum_{\mathbf{x} \in \mathscr{D}_{ood}} \mathrm{KL}(\mathscr{U} \| \mathscr{S}(\mathbf{x}, \boldsymbol{\theta})), \tag{3.2}$$

where $\mathscr{U}$ represents a discrete uniform distribution, *i.e.* $\mathscr{U} = (\frac{1}{k}, \cdots, \frac{1}{k})$ if the neural network is a k-class classification model, $\mathscr{S}(\mathbf{x}, \boldsymbol{\theta})$ is the probability distribution of $\mathbf{x}$ predicted by the neural network with weights in $\boldsymbol{\theta}$, and $\mathrm{KL}(\cdot\|\cdot)$ denotes the Kullback-Leibler (KL) divergence to measure the distance between $\mathscr{U}$ and $\mathscr{S}(\mathbf{x}, \boldsymbol{\theta})$. Namely, the predictive distributions of OoD samples are forced to be uniform distributions. In this way, the maximum predictive probability of OoD samples is lower than in-distribution samples so that the neural network can distinguish in- and out-of-distribution to avoid the overconfidence in its predictions.

3.2.2.2 Adversarial Robustness

To evaluate the adversarial robustness of the neural network with the searched architecture $\boldsymbol{\alpha}$, we first perturb each data sample in the validation dataset $\mathscr{D}_{val}$ by PGD adversarial attack [64] to generate a perturbed validation dataset denoted by $\mathscr{D}_{val}^{adv}$. Then we compute the average loss on this perturbed dataset as

$$\mathscr{L}_{adv}(\boldsymbol{\theta}, \boldsymbol{\alpha}) = \frac{1}{|\mathscr{D}_{val}^{adv}|} \sum_{(\mathbf{x}, y) \in \mathscr{D}_{val}^{adv}} \ell(\theta, \mathbf{x}, y). \tag{3.3}$$

3.2.3 Objective Functions of Resource Constraints

Architecture with less resource consumption have more application scenarios even in resource-constrained mobile devices. Therefore, we regard resource constraints as the

desired objectives and mainly focus on the number of parameters and multiply-add operations (*i.e.* FLOPs).

3.2.3.1 Number of Parameters

By following DARTS [77], we determine the operation on each edge of each cell in the final architecture as the one with the largest probabilities. So the number of parameters in an architecture can be computed as $\mathscr{N}(\boldsymbol{\alpha}) = \sum_{(i,j)\in\mathbf{E}} n_{\text{argmax}_{o\in\mathscr{O}}\ \alpha_o^{(i,j)}}$, where n_o denotes the number of parameters corresponding to the operation $o(\cdot)$. Note that argmax is a non-differentiable operation, making the computation of the gradient of $\mathscr{N}(\boldsymbol{\alpha})$ with respect to $\boldsymbol{\alpha}$ infeasible. To make such operation differentiable, we use the softmax function to approximate the argmax operation and then formulate it as $\hat{\mathscr{N}}(\boldsymbol{\alpha}) = \sum_{(i,j)\in\mathbf{E}} \sum_{o\in\mathscr{O}} \frac{\exp(\alpha_o^{(i,j)})}{\sum_{o'\in\mathscr{O}} \exp(\alpha_{o'}^{(i,j)})} n_o$. Furthermore, to prevent the model to search over-simplified architecture (*i.e.* the one containing too many parameter-free operations) that leads to unsatisfactory performance, we add a lower bound L to the parameter size $\hat{\mathscr{N}}(\boldsymbol{\alpha})$. Therefore, the objective function of the number of parameters can be formulated as

$$\mathscr{L}_{nop}(\boldsymbol{\alpha}) = |\hat{\mathscr{N}}(\boldsymbol{\alpha}) - L|. \tag{3.4}$$

3.2.3.2 FLOPs

Similar to the number of parameters, FLOPs also only depend on the architecture $\boldsymbol{\alpha}$. Thus we formulate the objective function of FLOPs similarly to the number of parameters (*i.e.* Eq. (3.4)). First, we carefully compute the FLOPs f_o of each operation $o(\cdot)$ in $\mathscr{O}$. Then we use the softmax function to approximate the argmax operation and calculate the

total FLOPs $\hat{\mathscr{F}}(\boldsymbol{\alpha}) = \sum_{(i,j)\in\mathbf{E}} \sum_{o\in\mathscr{O}} \frac{\exp(\alpha_o^{(i,j)})}{\sum_{o'\in\mathscr{O}} \exp(\alpha_{o'}^{(i,j)})} f_o$. Finally we constrain $\hat{\mathscr{F}}(\boldsymbol{\alpha})$ with a lower bound F and formulate the objective function of FLOPs as

$$\mathscr{L}_{flops}(\boldsymbol{\alpha}) = |\hat{\mathscr{F}}(\boldsymbol{\alpha}) - F|. \tag{3.5}$$

In practice, we can specify the minimum constraints L in Eq. (3.4) and F in Eq. (3.5) to search an architecture with an expected model size. On the other hand, we can also get a set of Pareto-optimal architecture by adjusting the minimum constraints as shown in Table 3.1.

3.2.4 Multi-Objective Bi-Level Formulation

E2RNAS aims to search the architecture $\boldsymbol{\alpha}$ to minimize the validation loss for the effectiveness, the number of parameters and FLOPs for the efficiency, and the OoD robustness and adversarial robustness for the robustness. Thus, we combine Eqs. (3.2), (3.3), (3.4) and (3.5) to formulate the entire objective function as

$$\begin{aligned} \min_{\boldsymbol{\alpha}} U(\boldsymbol{\theta}^*(\boldsymbol{\alpha}), \boldsymbol{\alpha}) = (&\mathscr{L}_{val}(\boldsymbol{\theta}^*(\boldsymbol{\alpha}), \boldsymbol{\alpha}), \mathscr{L}_{ood}(\boldsymbol{\theta}^*(\boldsymbol{\alpha}), \boldsymbol{\alpha}), \\ &\mathscr{L}_{adv}(\boldsymbol{\theta}^*(\boldsymbol{\alpha}), \boldsymbol{\alpha}), \mathscr{L}_{nop}(\boldsymbol{\alpha}), \mathscr{L}_{flops}(\boldsymbol{\alpha})) \\ \text{s.t. } \boldsymbol{\theta}^*(\boldsymbol{\alpha}) = \arg\min_{\boldsymbol{\theta}} &(\mathscr{L}_{tr}(\boldsymbol{\theta}, \boldsymbol{\alpha}) + \mathscr{L}_{ood}(\boldsymbol{\theta}, \boldsymbol{\alpha})). \end{aligned} \tag{3.6}$$

where $\boldsymbol{\theta}^*(\boldsymbol{\alpha})$ indicates that the network weights $\boldsymbol{\theta}$ depends on the network architecture $\boldsymbol{\alpha}$. Objective loss functions $\mathscr{L}_{val}, \mathscr{L}_{ood}, \mathscr{L}_{adv}$ depend on the network weights and architecture. Objective loss functions $\mathscr{L}_{nop}, \mathscr{L}_{flops}$ only depend on the network architecture.

Problem (3.6) is similar to the bi-level optimization problem (3.1) in the original DARTS, where the LL subproblem is similar, but there exists significant differences

in that the UL subproblem contains five objectives, as shown in Figure 3.1. On the other hand, different from RC-DARTS [57] that directly adds the resource constraint into the original DARTS objective function (3.1) as a constraint and formulates the objective function as a constrained optimization problem or GOLD-NAS [6] that regards resource constraint as the regularization term and then optimizes as a single-objective optimization problem, we evaluate all the objectives in the UL subproblem and cast it as a multi-objective optimization problem solving by a gradient-based multi-objective method, *i.e*. MGDA.

Therefore, problem (3.6) is a multi-objective bi-level optimization problem which is also a generalization of problem (3.1) in the DARTS. It can be understood as a two-stage optimization. Firstly, when given an architecture parameter $\boldsymbol{\alpha}$, we can learn a model with optimal model weights $\boldsymbol{\theta}^*$ via the empirical risk minimization on both training dataset and OoD dataset. Secondly, given $\boldsymbol{\theta}^*$, the architecture parameter $\boldsymbol{\alpha}$ is updated on the validation dataset by making a trade-off among its performance, robustness, and resources consumption. Therefore, we can solve the problem (3.6) in two stages, which are described as follows.

Updating $\boldsymbol{\theta}$ Given the architecture parameter $\boldsymbol{\alpha}_t$, $\boldsymbol{\theta}$ can be simply updated as

$$\boldsymbol{\theta}_{t+1} = \boldsymbol{\theta}_t - \eta_{\boldsymbol{\theta}} \nabla_{\boldsymbol{\theta}} (\mathscr{L}_{tr}(\boldsymbol{\theta}_t, \boldsymbol{\alpha}_t) + \mathscr{L}_{ood}(\boldsymbol{\theta}_t, \boldsymbol{\alpha}_t)), \tag{3.7}$$

where t denotes the index of the iteration and $\eta_{\boldsymbol{\theta}}$ denotes the learning rate.

Updating $\boldsymbol{\alpha}$ After obtaining $\boldsymbol{\theta}_{t+1}$, we can optimize the UL subproblem to update the architecture parameter $\boldsymbol{\alpha}$. As the UL subproblem is a multi-objective optimization

problem, we adopt the MGDA to solve it. In MGDA, we first need to solve problem (2.1), which requires the computation of the gradients of the five objectives with respect to $\boldsymbol{\alpha}$. The gradient of $\mathscr{L}_{nop}(\boldsymbol{\alpha}_t)$ and $\mathscr{L}_{flops}(\boldsymbol{\alpha}_t)$ with respect to $\boldsymbol{\alpha}$ is easy to compute, while the gradient of the remaining three objectives with respect to $\boldsymbol{\alpha}$ is a bit complicate as $\boldsymbol{\theta}^*(\boldsymbol{\alpha}_t)$ is also a function of $\boldsymbol{\alpha}$ and it is too expensive to obtain $\boldsymbol{\theta}^*(\boldsymbol{\alpha}_t)$. Therefore, we use a second-order approximation as

$$\begin{aligned} &\nabla_{\boldsymbol{\alpha}} \mathscr{L}_{val}(\boldsymbol{\theta}^*(\boldsymbol{\alpha}_t), \boldsymbol{\alpha}_t) \\ \approx &\nabla_{\boldsymbol{\alpha}} \mathscr{L}_{val}(\boldsymbol{\theta}_{t+1} - \eta_{\boldsymbol{\theta}} \nabla_{\boldsymbol{\theta}} \mathscr{L}_{tr}(\boldsymbol{\theta}_{t+1}, \boldsymbol{\alpha}_t), \boldsymbol{\alpha}_t). \end{aligned} \tag{3.8}$$

where ξ denotes the learning rate for inner optimization. Obviously when $\eta_{\boldsymbol{\theta}} = 0$, $\boldsymbol{\theta}_{t+1}$ becomes an approximation of $\boldsymbol{\theta}^*(\boldsymbol{\alpha}_t)$ and Eq. (3.8) degenerates to the first-order approximation, which can speed up the gradient computation and reduce the memory cost but lead to worse performance [77]. So we use the second-order approximation in Eq. (3.8). Similarly, the gradient of $\mathscr{L}_{adv}(\boldsymbol{\theta}^*(\boldsymbol{\alpha}_t), \boldsymbol{\alpha}_t)$ and $\mathscr{L}_{ood}(\boldsymbol{\theta}^*(\boldsymbol{\alpha}_t), \boldsymbol{\alpha}_t)$ can be computed approximately. Then we solve the problem (2.1) to get the weight $\Gamma = (\gamma_1, \cdots, \gamma_5)$ for five objectives, respectively. While the problem (2.1) has no analytical solution, we apply the Frank-Wolfe algorithm to solve it.

After that, we can update $\boldsymbol{\alpha}_t$ as

$$\boldsymbol{\alpha}_{t+1} = \boldsymbol{\alpha}_t - \eta_{\boldsymbol{\alpha}} \nabla_{\boldsymbol{\alpha}_t} \big(U(\boldsymbol{\theta}^*(\boldsymbol{\alpha}_t), \boldsymbol{\alpha}_t) \Gamma^T \big), \tag{3.9}$$

where $\eta_{\boldsymbol{\alpha}}$ denotes the learning rate for $\boldsymbol{\alpha}$. The whole algorithm is summarized in Algorithm 1.

Algorithm 1 E2RNAS

Input: Dataset $\mathscr{D}_{tr}$ and $\mathscr{D}_{val}$, OoD dataset $\mathscr{D}_{ood}$, batch size B, perturbation size ϵ, minimum constraint L and F, learning rates $\eta_{\boldsymbol{\alpha}}$ and $\eta_{\boldsymbol{\theta}}$

Output: Learned architecture parameter $\boldsymbol{\alpha}$

Randomly initialized $\boldsymbol{\alpha}_0$ and $\boldsymbol{\theta}_0$;
$t := 0$;
while not converged **do**
    Sample a mini-batch of size B;
    Update $\boldsymbol{\theta}_{t+1}$ according to Eq. (3.7);
    Compute five objective functions and the corresponding gradients;
    Compute weights Γ by solving problem (2.1);
    Update $\boldsymbol{\alpha}_t$ according to Eq. (3.9);
    $t := t + 1$;
end while

3.3 Experiments

In this section, we empirically evaluate the E2RNAS method on CIFAR-10 [60], CIFAR-100 [60] and ImageNet-1K [25] datasets.

3.3.1 Experimental Datasets

The CIFAR-10 dataset contains 50,000 training images and 10,000 testing images from 10 classes, each of which has 6,000 images with a 32×32 resolution in total. The CIFAR-100 dataset contains 100 classes grouped into 20 super-classes, with 500 training images and 100 testing images for each class. For the ImageNet dataset, we use the ImageNet-1K benchmark, which contains 1K high-level categories from the original 22K categories. For OoD test, we use the Street View House Numbers (SVHN) dataset [100], which consists of the images of house numbers captured from the Google street view and contains 10 classes with 73,257 images used for training and 26,032 images for testing.

3.3.2 Implementation Details

3.3.2.1 Search Space

The search space adopts the same setting as DARTS [77]. There are two types of cells, *i.e.* the reduction cell and the normal cell. The reduction cells are located at the 1/3 and 2/3 of the total depth of the network. Other cells in the network belong to the normal cell. There are 7 nodes in each cell for both reduction and normal cells, including four intermediate nodes, two input nodes, and one output node. In both normal and reduction cell, the set of operations $\mathcal{O}$ contains eight operations, including 3×3 separable convolutions, 5×5 separable convolutions, 3×3 dilated separable convolutions, 5×5 dilated separable convolutions, 3×3 max pooling, 3×3 average pooling, identity, zero. For the convolution operator, the ReLU-Conv-BN order is used. Each separable convolution is applied twice.

3.3.2.2 Search Settings

In the search process, by following DARTS [77], half of the whole training set is used for training a model and the other half for validation. A small network of 8 cells is trained for 50 epochs with the batch size as 64 and initial channels as 16. For the **adversarial robustness** objective, the adversarial examples are generated by the FGSM attack follow the setting in [142] with the perturbation size $\epsilon = 2$. For the **OoD robustness** objective, we use the SVHN dataset [100] as the OoD dataset, which is introduced in Section 3.3.1. For the **FLOPs** objective, we use a function f_o to approximate the FLOPs of the corresponding operation $o(\cdot)$. Let C denote the number of channels of input and output channels for the operation. W and H denote the width and height of the

feature map. Ignoring padding and bias, the FLOPs of an $\omega \times \omega$ separable convolution is $2(CHW + C^2HW + \omega^2CHW + 2CHW)$ and the FLOPs of an $\omega \times \omega$ dilated separable convolutions is $CHW + C^2HW + \omega^2CHW + 2CHW$ with $stride = 1$. The FLOPs of a 3×3 average pooling is CHW. The FLOPs of a 3×3 max pooling, identity or none operation is zero. The FLOPs of a skip connection is 0 if $stride = 0$ and is $C^2HW + 2 \times CHW$ if $stride = 1$. The ADAM optimizer [59] with the learning rate 3×10^{-4}, the momentum $\beta = (0.5, 0.999)$, and the weight decay 1×10^{-3} is used to update $\boldsymbol{\alpha}$ in the UL subproblem. The SGD optimizer with the momentum 0.9 and the weight decay 3×10^{-4} is used to update $\boldsymbol{\theta}$ in the LL subproblem. The proposed method is implemented in PyTorch 0.3.1, and all the experiments are conducted on one single NVIDIA Tesla V100S GPU.

3.3.2.3 Retrain Settings

A large network of 20 cells is retrained on the full training set for 600 epochs, with the batch size as 96, the initial number of channels 36, a cutout of length 16, the dropout probability 0.2, and auxiliary towers of weight 0.4.

3.3.2.4 Evaluation Metrics

For the **performance** objective, the accuracy is tested on the full testing set. For the **adversarial robustness** objective, adversarial examples are generated using the PGD attack [64] with the perturbation size $\epsilon = 2/255$ on the full testing set. The PGD attack takes 10 iterative steps with the step size of 2.5ϵ as suggested in [92]. For the **OoD robustness** objective, we evaluate the effectiveness of distinguishing between in- and

out-of-distribution samples by measuring the Area Under the Precision-Recall (AUPR) curve, which is a threshold-independent metric [52, 67] and the PR curve describes the relationship between precision and recall, where precision is computed by TP/(TP+FP), recall is computed by TP/(TP+FN), and TP, FP, TN, and FN denote true positive, true negative, false positive, and false negative. We specify the OoD images as positives to compute the AUPR in this chapter.

3.3.3 Analysis on Experimental Results

3.3.3.1 Searched Architecture on CIFAR-10

The normal and reduction cells searched by the E2RNAS on the CIFAR-10 dataset are presented in Figure 3.2. The found reduction cell does not contain any operation with parameters, which reduces the parameter size of the architecture. Moreover, it is notable that both normal and reduction cells do not include the max pooling operation and begin with the average pooling operation, which indicates the found architecture is potentially OoD robust based on the observations in [1]. This also coincides with experimental results in Table 3.1.

3.3.3.2 Architecture Evaluation on CIFAR-10

The comparison of the proposed E2RNAS method with state-of-the-art NAS methods on the CIFAR-10 dataset is shown in Tables 3.1 and 3.2. Two variants of the E2RNAS method denoted by E2RNAS-S1 and E2RNAS-S2 can obtain a set of Pareto-optimal architecture by adjusting L in Eq. (3.4), Specifically, we set $L = 2.5$ in Eq. (3.4) for

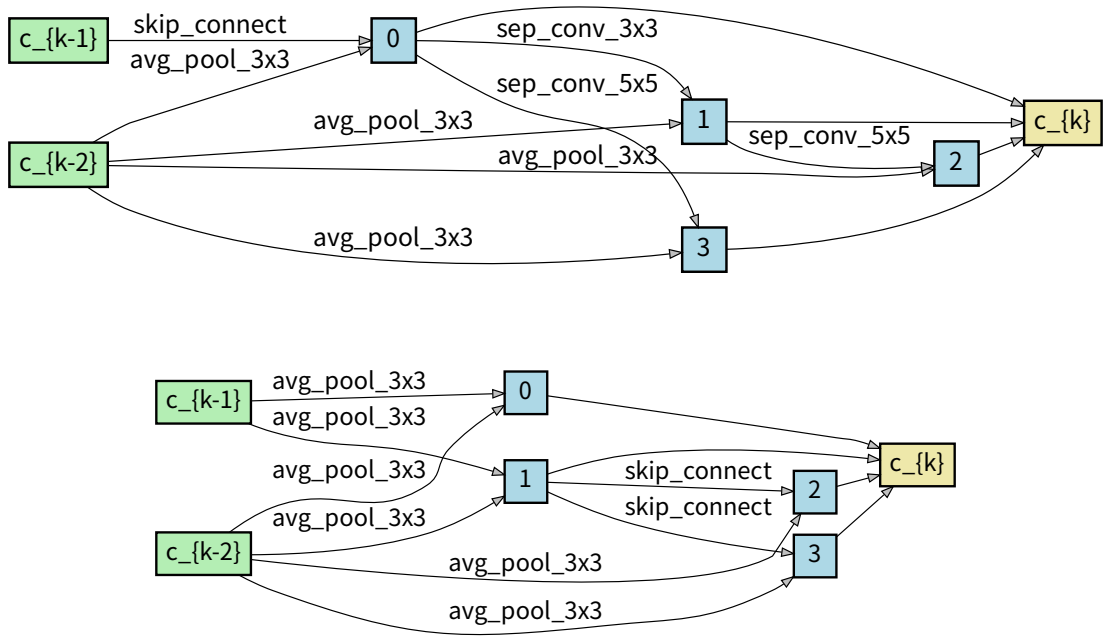


Figure 3.2: The found normal cell (top) and reduction cell (bottom) on the CIFAR-10 dataset by the proposed E2RNAS method. This architecture corresponds to “E2RNAS-S1" in Table 3.1.

Table 3.1: Comparison with gradient-based NAS methods on the CIFAR-10 dataset. † represents training without the cutout augmentation. ‡ indicates the use of the provided genotype in the original chapter. ↑ (↓) indicates a larger (lower) value is better. The search cost is recorded on one single NVIDIA Tesla V100S GPU and includes validation time while searching. We set $L = 2.5$ in Eq. (3.4) for E2RNAS-S1 and $L = 3.5$ for E2RNAS-S2.

Architecture	**Multiple Objective**	**Test Err. (%) ↓**	**Params (M) ↓**	**FLOPs (M) ↓**	**PGD Acc. (%) ↑**	**OoD AUPR (%) ↑**	**Search Cost (GPU days) ↓**
DARTS‡ [77]		2.59	3.3	539	25.68	31.44	0.6
P-DARTS‡ [17]		**2.37**	3.4	543	39.39	21.46	0.25
PC-DARTS‡ [143]		3.78	2.72	442	35.19	30.93	**0.24**
E2RNAS-S1	√	2.87	**2.55**	**425**	36.39	31.38	0.98
E2RNAS-S2	√	2.75	3.53	586	**42.81**	**34.64**	0.98

E2RNAS-S1 and $L = 3.5$ for E2RNAS-S2. Notably, E2RNAS outperforms NAS methods [76, 108] by searching for a more lightweight architecture with lower search costs of three or four orders of magnitude and a comparable test error rate. Moreover, although ENAS [106] slightly outperforms E2RNAS in terms of the search time, it finds a much

Table 3.2: Comparison with various architecture on the CIFAR-10 dataset. ↑ (↓) indicates a larger (lower) value is better. "RL", "Evo." and "SMBO" stand for reinforcement learning-based, evolution-based and sequential model-based optimization NAS method, respectively. "MO-G", "MO-RL" and "MO-Evo." stand for multi-objective gradient-based, multi-objective reinforcement learning-based and multi-objective evolution-based NAS method, respectively. "-" indicates that the corresponding result is not reported. For E2RNAS-S2, we set $L = 3.5$ in Eq. (3.4).

Architecture	**Test Err. (%) ↓**	**Params (M) ↓**	**Search Cost (GPU days) ↓**	**Search Method**
DenseNet-BC [55]	3.46	25.6	-	manual
AmoebaNet-B [108]	2.55	2.8	3150	Evo.
PNAS [76]	3.41	3.2	225	SMBO
ENAS [106]	2.89	4.6	0.5	RL
LEMONADE [32]	3.05	4.7	80	MO-Evo.
Proxyless-R [11]	2.30	5.8	-	MO-RL
RAPDARTS [41]	2.83	2.8	12	MO-G
GOLD-NAS-K [6]	2.57	3.3	1.1	MO-G
RC-DARTS-C42 [57]	2.81	3.3	1	MO-G
FPNASNet [24]	3.01	5.76	-	MO-G
E2RNAS-S2	2.75	3.53	0.98	MO-G

larger architecture with a higher test error. Compared with the original DARTS in [77], E2RNAS achieves a better trade-off among accuracy, efficiency, and robustness. Specifically, although the test error of E2RNAS is slightly higher than DARTS, E2RNAS can find more efficient and robust architecture. For example, compared with DARTS, the E2RNAS-S1 architecture with only 2.55 MB model size and 425 MB FLOPs significantly improves the PGD accuracy by 11.11%, while the corresponding AUPR slightly decreases by 0.06%. Besides, although E2RNAS considers five objectives to trade-off, its search process is slightly slower than DARTS with only one objective. On the inference latency, DARTS is with 28.64 ms latency on a single NVIDIA Tesla V100S GPU, while E2RNAS-S1 is only with 19.42 ms and E2RNAS-S2 is with 27.49 ms on the same device. Therefore,

those results indicate the efficiency of E2RNAS.

Even E2RNAS can achieve comparable performance with variants of DARTS such as P-DARTS [17], and PC-DARTS [143]. For example, compared E2RNAS-S2 with P-DARTS and PC-DARTS, E2RNAS can find an architecture with a higher AUPR and comparable performance on other objectives.

On the other hand, compared with multi-objective NAS methods, E2RNAS outperforms LEMONADE [32], RC-DARTS-C42 [57], RAPDARTS [41], and FPNASNet [24] by finding more lightweight and effective architecture in a shorter search time. Moreover, different from Proxyless-R [11] that searches for architecture with good performance but a large model size, E2RNAS can make a better trade-off between the accuracy and parameter size. Besides, E2RNAS achieves competitive performance with a faster search process when compared with GOLD-NAS-K [6].

In summary, experimental results in Table 3.1 show that E2RNAS can efficiently search a significantly robust architecture with a lower model size and comparable classification accuracy, compared with state-of-the-art NAS methods.

3.3.3.3 Experimental Results on CIFAR-100

We also evaluate the proposed E2RNAS method on the CIFAR-100 dataset. The comparison of E2RNAS with DARTS [77], P-DARTS [17] and PC-DARTS [143] is presented in Table 3.3. The experimental results on the CIFAR-100 dataset are similar to that on the CIFAR-10 dataset in that E2RNAS can find a lightweight and robust architecture with a slightly decreased test accuracy. For example, compared with DARTS, E2RNAS-S1 with

only 2.3 MB model size and 375 MB FLOPs can significantly improve the PGD accuracy and AUPR. Moreover, compared E2RNAS-S2 with P-DARTS and PC-DARTS, E2RNAS also outperforms them in the robustness with a slight drop in the test accuracy. These quantitative experiments indicate that E2RNAS can search robust architecture with lower resource consumption and comparable performance.

Table 3.3: Comparison with state-of-the-art NAS methods on the CIFAR-100 dataset. ‡ indicates the use of the provided genotype in the original chapter. ↑ (↓) indicates a larger (lower) value is better. We set $L = 2.5$ in Eq. (3.4) for E2RNAS-S1 and $L = 3.5$ for E2RNAS-S2.

Architecture	**Test Acc.** (%) ↑	**Params** (M) ↓	**FLOPs** (M) ↓	**PGD Acc.** (%) ↑	**OoD AUPR** (%) ↑
DARTS‡ [77]	**83.94**	3.4	539	13.66	32.83
P-DARTS‡ [17]	83.54	3.5	543	17.34	32.43
PC-DARTS‡ [143]	83.06	3.7	568	18.44	32.75
E2RNAS-S1	79.39	**2.3**	**375**	**22.26**	32.88
E2RNAS-S2	81.70	3.9	643	21.21	**33.30**

Table 3.4: Evaluation of the generalization ability of the proposed E2RNAS method on the CIFAR-10 and CIFAR-100 datasets. “{E2RNAS *on* #method}" means the architecture searched by combining “method" with E2RNAS. ‡ indicates the provided genotype in the original paper. ↑ (↓) indicates a larger (lower) value is better. The search cost is recorded on one single NVIDIA Tesla V100S GPU and includes validation time while searching.

Dataset	**Architecture**	**Test Err.** (%) ↓	**Params** (M) ↓	**FLOPs** (M) ↓	**PGD Acc.** (%) ↑	**OoD AUPR** (%) ↑
CIFAR-10	P-DARTS‡ [17]	**2.37**	3.4	543	39.39	21.46
	E2RNAS *on P-DARTS*	3.37	**3.2**	**509**	**47.85**	**29.82**
	PC-DARTS‡ [143]	**2.72**	3.6	568	35.19	26.07
	E2RNAS *on PC-DARTS*	3.78	**2.7**	**443**	**42.38**	**38.93**
CIFAR-100	P-DARTS‡ [17]	**16.46**	3.5	543	17.34	32.43
	E2RNAS *on P-DARTS*	17.51	**3.0**	**480**	**24.45**	**33.38**
	PC-DARTS‡ [143]	**16.94**	3.7	568	18.44	32.75
	E2RNAS *on PC-DARTS*	17.13	**3.1**	**492**	**24.37**	**34.64**

3.3.4 The Generalization of E2RNAS

3.3.4.1 Combine with other gradient based NAS methods

A notable benefit of E2RNAS is its generalization ability, which means the proposed E2RNAS method can be seamlessly combined with other NAS methods, especially DARTS-based methods, to make a better trade-off among multiple objectives. To reveal it, we combine E2RNAS with two variants of DARTS, *i.e*. P-DARTS [17] and PC-DARTS [143], denoted by "E2RNAS *on P-DARTS*" and "E2RNAS *on PC-DARTS*", respectively.

Although these two methods improve the search process of DARTS from different perspectives, their objective functions are still similar to the original DARTS, *i.e*. problem (3.1). Therefore, built on their method and following their experimental settings, we can adapt the proposed E2RNAS method to combine with them for further improvements, *i.e*. taking resource constraint and robustness into consideration and then reformulating their objective function as a multi-objective bi-level optimization problem similar to the problem (3.6) to further balance a trade-off among performance, resource constraint and robustness.

We evaluate those generalized methods and compare them with their corresponding original method on both the CIFAR-10 and CIFAR-100 datasets. The corresponding experimental results and comparison are presented in Table 3.4. The results show that E2RNAS can make a better trade-off among multiple objectives, which is similar to the results on DARTS, *i.e*. finding a more lightweight architecture with significantly increased robustness and slightly decreased test accuracy. Besides, we find that the

search time only slightly increases when combining E2RNAS with P-DARTS and PC-DARTS, even by only 0.04 GPU days with P-DARTS on the CIFAR-10 dataset, which indicates the efficiency of E2RNAS.

3.3.4.2 Transfer Architecture to ImageNet-1K

We further transfer the architecture searched by E2RNAS on the CIFAR-10 dataset to the larger ImageNet-1K dataset. Table 3.5 shows that the E2RNAS-S2 method performs better than DARTS in terms of the test error and PGD accuracy. The trade-off between the accuracy and robustness is consistent with previous experiments. This experiment shows that the architecture searched by E2RNAS on a smaller dataset is transferable to a larger dataset while keeping the trade-off between different objectives.

Table 3.5: Transfer searched architecture to the ImageNet-1K dataset. ‡ indicates the use of the provided genotype in the original paper. ↑ (↓) indicates a larger (lower) value is better.

Architecture	**Test Err.** (%) ↓	**Params** (M) ↓	**FLOPs** (G) ↓	**PGD Acc.** (%) ↑	**OoD AUPR** (%) ↑
DARTS‡	26.76	4.72	69.5	0.82	14.07
E2RNAS-S1	30.83	**3.76**	**55.4**	0.70	14.46
E2RNAS-S2	**26.07**	4.93	73.3	**4.72**	**14.54**

3.3.5 Ablation Study and Discussion

This section studies how each design in E2RNAS influences its performance on different objectives. We first discuss the design of the LL subproblem of the problem (3.6) and then investigate the effectiveness of MGDA in the UL subproblem. The corresponding results are presented in Table 3.6.

Table 3.6: Ablation study on the CIFAR-10 dataset under the same minimum constraints L and F in Eq. (3.4) and Eq. (3.5), respectively. MGDA is applied to make a trade-off among multiple objectives in UL subproblem and if without MGDA (*i.e.* "E2RNAS *w/o MGDA*"), it means equal weights of the five objectives in problem (3.6) are used. "E2RNAS *w/ AT in LL*" denotes using adversarial training in LL subproblem and "E2RNAS *w/o OoD in LL*" represents that we only minimize $\mathscr{L}_{tr}(\boldsymbol{\theta}, \boldsymbol{\alpha})$ in LL subproblem. ↑ (↓) indicates a larger (lower) value is better.

Architecture	**Test Err.** (%) ↓	**Params** (M) ↓	**FLOPs** (M) ↓	**PGD Acc.** (%) ↑	**OoD AUPR** (%) ↑
E2RNAS	2.76	3.53	586	40.32	31.46
E2RNAS *w/o MGDA*	2.72	4.0	667	39.76	31.17
E2RNAS *w/ AT in LL*	3.34	3.0	476	52.40	32.53
E2RNAS *w/o OoD in LL*	3.26	2.7	439	39.22	31.37

3.3.5.1 Design of LL subproblem

E2RNAS aims to solve a multi-objective bi-level problem (3.6), where the UL subproblem optimizes the architecture $\boldsymbol{\alpha}$ by evaluating multiple objectives using the model $\boldsymbol{\theta}$ learned in the LL subproblem. Hence, a well-designed LL subproblem can improve the performance of our proposed E2RNAS method.

It is well known that adversarial training can significantly improve the adversarial robustness of the model but maybe result in a bad clean performance. Moreover, we find that the trade-off between performance and adversarial robustness exists in the NAS domain. The corresponding results are shown in Table 3.6. Comparing "E2RNAS *w/ AT in LL*" with E2RNAS, despite the PGD accuracy significantly increasing, the test accuracy also decreases, which fits the observation in [28]. Therefore, we do not apply adversarial training in the E2RNAS method to achieve comparable performance on classification accuracy.

Different from adversarial robustness, we find that training the model in the LL

subproblem with OoD data samples (*i.e.* E2RNAS *vs.* "E2RNAS *w/o OoD in LL*" in Table 3.6) can not only improve the OoD metric but also make a better trade-off among other objectives, especially test error *vs.* model size. Hence, we add the OoD loss in the LL subproblem to achieve better performance.

3.3.5.2 Effectiveness of MGDA

The MGDA method is applied to solve the UL subproblem of the problem (3.6), which is a multi-objective problem to minimize five objectives. If without the MGDA method, it means that we solve the UL subproblem by minimizing an equally weighted sum of five objectives (*i.e.* $\Gamma = (0.2, \cdots, 0.2)$ in Eq. (3.9)). We quantitatively compare the performance of E2RNAS with and without MGDA (*i.e.* "E2RNAS" *vs.* "E2RNAS *w/o MGDA*" in Table 3.6) and find that solving with MGDA achieves much better results on the parameter size, FLOPs, OoD metric (*i.e.* AUPR) and PGD accuracy. So instead of using equal weights, applying MGDA can find a good solution of weights and make a trade-off among multiple objectives.

3.4 Summary

This chapter proposes the E2RNAS method that optimizes multiple objectives simultaneously to search an effective, efficient, and robust architecture. The proposed objective function is formulated as a multi-objective bi-level problem, and we design an algorithm to integrate the MGDA with the bi-level optimization. Experiments demonstrate that E2RNAS can find robust architecture with optimized model size and comparable classifi-

cation accuracy on various datasets. In our future study, we are interested in extending the proposed E2RNAS method to search for multiple Pareto-optimal architecture at one time.

CHAPTER 4

LEARNING FEATURE ALIGNMENT ARCHITECTURE FOR DOMAIN ADAPTATION

4.1 Introduction

In the previous chapter, we introduced E2RNAS to search for efficient architecture optimized for multiple objectives. It remains to question whether the searched architecture can transfer to multiple domains. We address this issue by investigating the alignment architecture in domain adaptation.

With access to large-scale labeled data, deep neural networks have achieved state-of-the-art performance among a variety of machine learning problems and applications [30, 50, 51, 61, 101, 109, 146]. However, with intolerably time-consuming and labor-expensive costs, it is hard for a target domain of interest to collect enough labeled data

for model training. One solution is to transfer a deep neural network trained on a data-sufficient source domain to the target domain where only unlabeled data is available. However, this learning paradigm suffers from the shift in data distributions across different domains, which brings a major obstacle in adapting predictive models for the target task.

DA [102, 144] aims to learn a high-performance learner on a target domain via utilizing the knowledge transferred from a source domain, which has a different but related data distribution to the target domain. Many DA methods aim to bridge the gap between source and target domains to apply the classifier learned in the source domain to the target domain. To achieve this goal, recent DA works can be grouped into two main categories: *distance-based* methods [3, 4, 14, 22, 84, 120, 122, 132, 150, 158] and *adversarial-based* DA methods [37, 83, 104, 115, 131].

For distance functions adopted by DA, the first attempt is the *Proxy $\mathcal{A}$-distance* [3], which aims to minimize the generalization error by discriminating between source and target samples. *Maximum Mean Discrepancy* is a popular distance measure between two domains and it has been used in Deep Domain Confusion [132] and Deep Adaptation Network [84].

Although numerous distance-based DA methods have been proposed, learning the domain-invariant feature representation is still challenging. Distance alone in a high-dimensional space may be difficult to reflect the domain discrepancy adequately [68, 69]. The alignment of hidden feature representations also relies heavily on the network architecture design. However, the network architecture of existing methods is manually

designed by experts, hence it is hard to guarantee that hidden feature representations from different domains can be well aligned since the difficulty levels of different DA tasks vary. For instance, a complex task may require a more sophisticated network architecture than an easy task, making a hand-crafted network architecture fail to do the alignment for tasks with varying difficulty levels, limiting DA methods' capacity and versatility.

To alleviate those limitations, in this chapter, we propose a new similarity function called Population Correlation (PC) to measure the similarity between the source and target domains. A learning model can learn a domain-invariant feature representation by maximizing the PC between the source and target domains. Specifically, maximizing PC can force the two domains to have similar distributions since the PC is the maximum pairwise correlations between source and target samples. To further align hidden feature representations between source and target domains, we design a reinforcement-based NAS method called Alignment Architecture Search with Population Correlation to learn deep alignment architecture. In this way, AASPC can better learn domain-invariant feature representations for different DA tasks. To the best of our knowledge, the proposed AASPC method is the first NAS framework designed for distance-based DA methods. AASPC is also one of few works integrating NAS methods into deep DA methods. The contributions of this chapter are summarized as follows:

- We propose a new similarity measure, i.e., PC, to measure the domain similarity. By maximizing the PC between the source and target domains, our method can learn domain-invariant feature representation.
- We design the AASPC framework to search an optimal network architecture to

align hidden features between the source and target domains.

- Experimental results on three benchmark datasets demonstrate the effectiveness of the proposed methods.

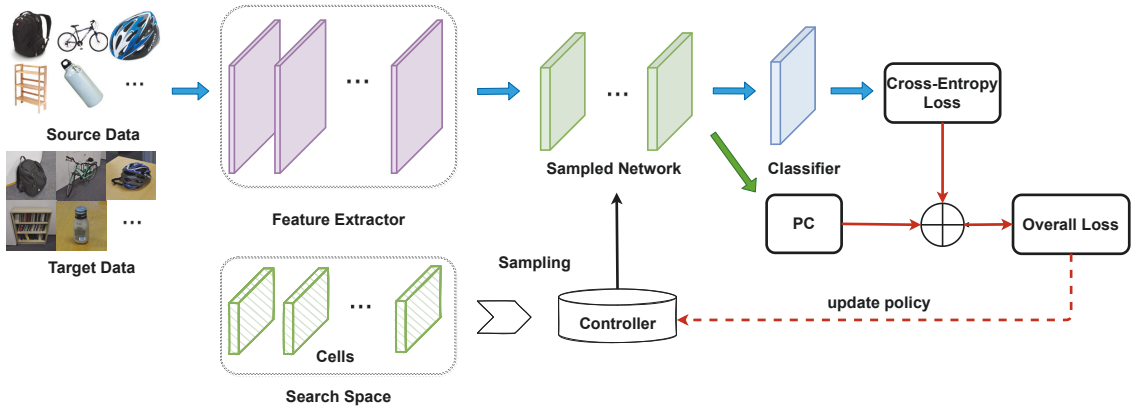


Figure 4.1: Overview of the AASPC framework. Source and target data first go through the feature extractor to extract hidden features. The controller samples cell choices for each cell and connections between the cells from search space to generate the architecture of the sampled network. Source and target data with the extracted feature representation then go through the sampled network. Finally, the cross-entropy loss is minimized and the PC is maximized. The controller's policy is updated by the reward of the negative overall loss.

4.2 Population Correlation

We first present the definition of PC. Here we study DA under the unsupervised setting. That is, the target domain has unlabeled data only. In DA, the source domain $\mathscr{D}_s = \{(\mathbf{x}_i^s, \mathbf{y}_i^s)\}_{i=1}^{n_s}$ has n_s labeled samples and the target domain $\mathscr{D}_t = \left\{\mathbf{x}_j^t\right\}_{j=1}^{n_t}$ has n_t unlabeled samples. To adapt the classifier trained on the source domain to the target domain, one solution is to minimize the domain discrepancy or equivalently maximize the domain similarity. To achieve this, we propose the PC to measure the similarity between the

source and target domains. Specifically, suppose $F(\cdot)$ is the feature extraction network. Then the PC between the source and target domains can be computed based on each pair of source and target samples as

$$\begin{aligned} \mathrm{PC}(\mathscr{D}^s, \mathscr{D}^t) = & \frac{1}{n_s} \sum_{i=1}^{n_s} \max_{j \in [n_t]} \mathrm{corr}\left(F(\mathbf{x}_i^s), F(\mathbf{x}_j^t)\right) \\ & + \frac{1}{n_t} \sum_{j=1}^{n_t} \max_{i \in [n_s]} \mathrm{corr}\left(F(\mathbf{x}_i^s), F(\mathbf{x}_j^t)\right), \end{aligned} \tag{4.1}$$

where $\|\cdot\|_2$ denotes the ℓ_2 norm of a vector, $\mathrm{corr}(\mathbf{x}_1, \mathbf{x}_2) = \frac{\mathbf{x}_1^T \mathbf{x}_2}{\|\mathbf{x}_1\|_2 \|\mathbf{x}_2\|_2}$ denotes the correlation between two vectors, and $[n]$ denotes a set of integers $\{1, \ldots, n\}$ for an integer n. Here we use the cosine similarity to calculate the correlation between two vectors, thus the larger the PC value is, the more similar the two domains are.

For DA tasks, the hidden feature representations learned by the feature extraction network should be not only discriminative to train a strong classifier but also domain-invariant to both the source and target domains. Only maximizing the PC can help learn a domain-invariant feature representation and only minimizing the classification loss is to learn a discriminative feature representation. Therefore, we combine the classification loss and the PC to obtain the final objective function, which is formulated as

$$\mathscr{L}_{\mathrm{PC}} = \frac{1}{n_s} \sum_{i=1}^{n_s} l(C(F(\mathbf{x}_i^s)), y_i^s) - \lambda \mathrm{PC}(\mathscr{D}^s, \mathscr{D}^t), \tag{4.2}$$

where λ is a trade-off parameter, $C(\cdot)$ denotes the classification layer, and $l(\cdot, \cdot)$ denotes the classification loss such as the cross-entropy loss.

By minimizing Eq. (4.2), the final learned feature representations are not only discriminative for classification but also domain-invariant for the adaptation.

4.3 The AASPC method

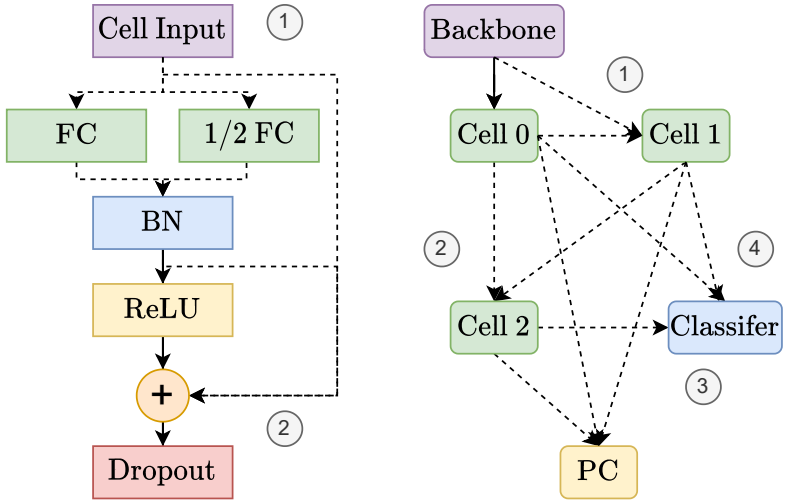


Figure 4.2: The search space of the DAMPC-NAS method. Dashed lines represent possible search choices and numbered grey circles indicate the order of choices generated from the controller.

In this section, we introduce the proposed AASPC framework that finds an optimal alignment architecture for source and target domains. An overview of the AASPC framework is shown in Figure 4.1.

4.3.1 Cell-based Search Space

We design the search space on the top of the Resnet-50 backbone, whose architecture is kept fixed, and hence we only search the architecture after the backbone. The search space of the AASPC method consists of two parts: within cells and between cells. We design the cell as the composition of the fully connected layer, batch-norm layer, dropout layer, and the associated activation functions. Within the cell, we search for the size of the fully connected layer and the location of the skip connection. Specifically, the

Algorithm 2 E2RNAS

Input: source data $\mathscr{D}_s$, target data $\mathscr{D}_t$, the number of training epochs n_{epochs}
Output: The searched architecture with learned weights

initialize controller;
for $i \leftarrow 0$, n_{epochs} **do**
    sample $\mathscr{A}_m$ from $\mathscr{A}_{space}$ with policy $\pi(m;\theta)$;
    fix controller policy $\pi(m;\theta)$;
    for mini-batch in $\mathscr{D}_s$ and $\mathscr{D}_t$ **do**;
        compute $\mathscr{L}_{\text{PC}}$ in Eq. (4.2) with $\mathscr{A}_m$
        update ω_m in $\mathscr{A}_{space}$ with $\mathscr{L}_{\text{PC}}$
    end for
    fix ω in $\mathscr{A}_{space}$;
    calculate reward of $\mathscr{A}_m$ as $R_m = -\mathscr{L}_{\text{PC}}$
    update θ in $\pi(m;\theta)$ with reward R_m
end for
return $\mathscr{A}_m$ with trained weights ω_m

search choice of the fully connected layer in a cell can be 'the same as input size' or 'the half of input size'. The starting location of the skip connection can be chosen from the cell input, the fully connected layer, and the batch-norm layer. We search for input and output connections between the cells of the N cells. For example, if there are three cells in the search space, i.e., $N = 3$, the input of "Cell 1" can be chosen from the outputs of "Backbone" and "Cell 0", and the input of "Cell 2" can be chosen from the outputs of "Cell 0" and "Cell 1", hence the input of a cell can be chosen from the outputs of the previous two cells. The calculation of PC can be chosen from one of all cells' outputs. Moreover, One of the outputs from the N cells, i.e., "Cell 0", "Cell 1" and "Cell 2", can connect to the classifier trained on source domain data. Hence, the total search space has $(2 \times 3)^N 2^{N-1} N^2$ configurations. An illustration of the search space in the AASPC method is shown in Figure 4.2. In experiments, for efficiency, we use the search space with $N = 3$ cells for all experiments.

4.3.2 Searching Alignment Architecture

The searching algorithm for the AASPC method is described in Algorithm 2. AASPC is a reinforcement-based NAS framework which leverages a controller network to sample architecture from the search space. The controller network is a LSTM that samples search choice via a softmax classifier. We denote by θ the learnable parameters of the controller. The policy of the controller is denoted by $\pi(m;\theta)$.

Table 4.1: Accuracy (%) on the Office-31 dataset with ResNet-50 as the backbone.

Type	Method	A→D	A→W	D→A	D→W	W→A	W→D	Avg
Source Only	ResNet-50 [51]	68.9	68.4	62.5	96.7	60.7	99.3	76.1
Dist. Based	JDA [85]	80.7	73.6	64.7	96.5	63.1	98.6	79.5
	DDC [132]	76.5	75.6	62.2	96.0	61.5	98.2	78.3
	DAN [84]	78.6	80.5	63.6	97.1	62.8	99.6	80.4
	D-CORAL [122]	81.5	77.0	65.9	97.1	64.3	99.6	80.9
	JAN [87]	84.7	85.4	68.6	97.4	70.0	99.8	84.3
	MDDA [137]	86.3	86.0	**72.1**	97.1	**73.2**	99.2	85.7
Adv. Based	DANN [36]	79.7	82.0	68.2	96.9	67.4	99.1	82.2
	ADDA [131]	77.8	86.2	69.5	96.2	68.9	98.4	82.9
	CAN [151]	85.5	81.5	65.9	98.2	63.4	99.7	82.4
	DDAN [137]	84.9	88.8	65.3	96.7	65.0	**100.0**	83.5
With NAS	ABAS [110]	87.6	89.4	64.1	98.4	69.3	99.8	84.8
	AASPC (Ours)	**90.8**	**93.1**	70.4	**98.7**	69.1	**100.0**	**87.0**

In each epoch, the training procedure of AASPC consists of two phases. In the first phase, we fix the parameters of the controller θ and train the shared weights ω in the search space $\mathscr{A}_{space}$. Specifically, the controller samples an architecture $\mathscr{A}_m$ from the search space $\mathscr{A}_{space}$ with policy $\pi(m;\theta)$. For each mini-batch from $\mathscr{D}_s$ and $\mathscr{D}_t$, $\mathscr{L}_{\text{PC}}$ is computed according to Eq. (4.2) and the shared weights ω_m of the sampled architecture are updated by minimizing $\mathscr{L}_{\text{PC}}$. In the second phase, we fix all the shared weights ω in

the search space $\mathscr{A}_{space}$ and update the parameter θ of the controller. Specifically, after one epoch of training, $-\mathscr{L}_{\text{PC}}$ is used as the reward to update the policy $\pi(m;\theta)$ in the controller. The gradient is computed via the REINFORCE algorithm [140] with a moving average baseline.

In summary, the AASPC method trains a supernet that contains all shared parameters in the search space during the searching process. The AASPC method samples a child network in each epoch to calculate the loss function defined in Eq. (4.2) and updates its shared parameters in the search space. Parameters in the controller are updated by the reward, which is the negative loss of the sampled child network. After searching, all weights of the final architecture are retained for testing. Different from two-stage one-shot NAS methods, there is no need for the AASPC method to retrain the final architecture from scratch for testing since AASPC can directly optimize the objective in Eq. (4.2), which is just the negative reward for the controller, in an end-to-end manner. In this way, the architecture is optimized alongside child networks' parameters. Therefore, the final architecture derived from the AASPC method can be deployed directly without parameter retraining, which improves the overall efficiency.

4.4 Experiments

In this section, we empirically evaluate the proposed method.

4.4.1 Setup

We conduct experiments on three benchmark datasets, including Office-31 [114], Office-Home [136], and VisDA-2017 [105]. The Office-31 dataset has 4,652 images in 31 categories collected from three distinct domains: *Amazon* (**A**), *Webcam* (**W**) and *DSLR* (**D**). We can construct six transfer tasks: **A** → **W**, **D** → **W**, **W** → **D**, **A** → **D**, **D** → **A**, and **W** → **A**. The Office-Home dataset consists of 15,500 images in 65 object classes under the office and home settings, forming four extremely dissimilar domains: *Artistic* (**Ar**), *Clip Art* (**Cl**), *Product* (**Pr**), and *Real-World* (**Rw**) and 12 transfer tasks. The VisDA-2017 dataset has over 280K images across 12 classes. It contains two very distinct domains: **Synthetic**, which contains renderings of 3D models from different angles and with different lighting conditions, and **Real** that are natural images. We study a transfer task: Synthetic → Real on this dataset.

We compare the proposed **AASPC** method with state-of-the-art DA methods, including Joint Distribution Adaptation (**JDA**) [85], Deep Domain Confusion (**DDC**) [132], Deep Adaptation Network (**DAN**) [84], Domain Adversarial Neural Network (**DANN**) [36], Correlation Alignment for Deep Domain Adaptation (**D-CORAL**) [122], Residual Transfer Networks (**RTN**) [86], Joint Adaptation Networks (**JAN**) [87], Adversarial Discriminative Domain Adaptation (**ADDA**) [131], Conditional Domain Adversarial Networks (**CDAN**) [83], Collaborative and Adversarial Network (**CAN**) [151], Manifold Dynamic Distribution Adaptation (**MDDA**) [137], and Dynamic Distribution Adaptation Network (**DDAN**) [137]. The results of baseline methods are directly reported from DDAN [137] and CDAN [83].

We leverage the ResNet-50 network [51] pretrained on the ImageNet dataset as the backbone for the feature extraction. For optimization, we use the mini-batch SGD with the Nesterov momentum 0.9. The learning rate is adjusted by $\eta_p = \eta_0(1+\alpha p)^{-\beta}$, where p is the index of training steps, η_0 = 0.1, α = 0.001, and β = 0.75. The batch size is set to 128 for all the datasets.

4.4.2 Results

The classification results on the Office-31 dataset are shown in Table 4.1. As illustrated in Table 4.1, the proposed AASPC method achieves the best average accuracy.

In four out of six transfer tasks, AASPC performs the best, especially on transfer tasks A→D and A→W, which is transferring from a large source domain to a small target domain and in the other two tasks, the AASPC method performs slightly worse than the best baseline method, which implies that the proposed AASPC model works well when the source data is sufficient and it can learn transferable feature representations for effective domain adaptation.

Table 4.3 shows the classification results on the Office-Home dataset. According to the results, AASPC achieves the best average accuracy and performs the best in eight out of twelve transfer tasks, while transferring from a large source domain to a small target domain (i.e., Cl→Ar, Pr→Ar, and Rw→Ar), AASPC achieves the best performance and this phenomenon is similar to the Office-31 dataset, which again demonstrates that the proposed AASPC model works well when the source data is sufficient.

According to experimental results on the most challenging VisDA-2017 dataset

Table 4.2: Accuracy (%) on the VisDA-2017 dataset with ResNet-50 as the backbone.

Type	Method	Synthetic→Real
Source Only	ResNet-50	45.6
Dist. Based	DAN	53.0
	RTN	53.6
	JAN	61.6
Adv. Based	DANN	55.0
	CDAN	66.8
With NAS	AASPC (Ours)	**68.75**

Table 4.3: Accuracy (%) on the Office-Home dataset with ResNet-50 as the backbone.

Type	Method	Ar→Cl	Ar→Pr	Ar→Rw	Cl→Ar	Cl→Pr	Cl→Rw	Pr→Ar	Pr→Cl	Pr→Rw	Rw→Ar	Rw→Cl	Rw→Pr	Avg
Source Only	ResNet-50	34.9	50.0	58.0	37.4	41.9	46.2	38.5	31.2	60.4	53.9	41.2	59.9	46.1
Dist. Based	JDA	38.9	54.8	58.2	36.2	53.1	50.2	42.1	38.2	63.1	50.2	44.0	68.2	49.8
	DAN	43.6	57.0	67.9	45.8	56.5	60.4	44.0	43.6	67.7	63.1	51.5	74.3	56.3
	D-CORAL	42.2	59.1	64.9	46.4	56.3	58.3	45.4	41.2	68.5	60.1	48.2	73.1	55.3
	JAN	45.9	61.2	68.9	50.4	59.7	61.0	45.8	43.4	70.3	63.9	52.4	76.8	58.3
Adv. Based	DANN	45.6	59.3	70.1	47.0	58.5	60.9	46.1	43.7	68.5	63.2	51.8	76.8	57.6
	CDAN	46.6	65.9	73.4	55.7	62.7	64.2	51.8	**49.1**	74.5	68.2	**56.9**	**80.7**	62.8
	DDAN	**51.0**	66.0	73.9	57.0	63.1	65.1	52.0	48.4	72.7	65.1	56.6	78.9	62.5
With NAS	AASPC (Ours)	46.53	**68.42**	**75.24**	**58.3**	**66.3**	**67.48**	**56.94**	44.77	**75.33**	**69.26**	51.94	80.33	**63.4**

shown in Table 4.2, the proposed AASPC method outperforms all the baseline methods by improving by 1.9% over state-of-the-art baseline methods (i.e., CDAN) on this dataset, which again demonstrates the effectiveness of the proposed method.

4.4.3 Ablation Study

To investigate the efficacy of key designs of the proposed AASPC method, we conduct ablation study on the Office-31 dataset by comparing with variants of AASPC, including Source Only (no distance calculation and architecture search), AAS (AASPC without population correlation), and PC (AASPC without alignment architecture search). According to the results shown in Table 4.4, the AASPC method outperforms both AAS and PC methods. AAS is inferior to AASPC with a drop of 3.26%, while it performs better

than Source Only by 2.1% in terms of the average accuracy. PC performs better than Source Only by 5.36% on the average accuracy. AASPC further improves over PC by 2.41% and 1.76% on the A→D and A→W tasks in terms of the accuracy, respectively. This experiment verifies the effectiveness of both the AAS and PC components in the AASPC method.

Table 4.4: Ablation Study on the Office-31 dataset with ResNet-50 as the backbone.

Method	A→D	A→W	D→A	D→W	W→A	W→D	Avg
Source Only	83.53	80.50	64.61	98.49	62.69	100.0	81.64
AAS (AASPC w/o PC)	86.35	89.18	64.75	98.24	63.90	100.0	83.74
PC (AASPC w/o AAS)	88.35	91.32	70.36	98.49	69.05	100.0	86.26
AASPC	90.76	93.08	70.36	98.74	69.05	100.0	87.0

4.4.4 Effectiveness of Population Correlation

To demonstrate the effectiveness of the proposed PC, we replace the measurement with other widely used distance functions on the Office-31, Office-Home, and VisDA-2017 datasets. We then compare the performance of PC with these distance functions, including Proxy $\mathscr{A}$-distance, Kullback-Leibler divergence (KL-divergence), Maximum Mean Discrepancies (MMD), CORrelation ALignmen (CORAL), and Central Moment Discrepancy (CMD). For a fair comparison, we only replace the minus of the PC with these distance functions in Eq. (4.2). Specifically, we adopt the ResNet-50 as the backbone, following with the bottleneck layer (consisting of a fully connected layer, a batch normalization layer, a ReLU activation function, and a dropout function) used for generating hidden features and a fully connected layer used for prediction. According to experimental results shown in Tables 4.5, 4.7 and 4.6, we can see that none of the distance

functions can obtain performance improvement compared with no distance function used (i.e., ResNet-50). One possible reason is that the normalization layer used in the bottleneck layer has improved the performance of the ResNet-50 and adapting these distance functions can not improve the performance further. However, the proposed PC can still obtain performance improvement over ResNet-50, which indicates the effectiveness of the proposed PC.

Table 4.5: Comparison of PC with other distance functions on the Office-31 dataset with ResNet-50 as the backbone.

Measurement	A→D	A→W	D→A	D→W	W→A	W→D	Avg
None	83.53	80.50	64.61	98.49	62.69	100.0	81.64
Proxy $\mathscr{A}$-distance	82.73	81.01	64.04	98.11	61.77	100.0	81.28
KL-divergence	83.94	79.75	63.90	97.86	63.51	99.80	81.46
MMD	83.13	79.25	64.11	98.74	63.12	100.0	81.39
CORAL	84.34	80.25	64.61	98.24	62.80	99.80	81.67
CMD	82.93	79.50	64.29	98.62	63.10	100.0	81.41
PC	88.35	91.32	70.36	98.49	69.05	100.0	86.26

4.4.5 Effectiveness of Alignment Architecture Search

To demonstrate the effectiveness of the alignment architecture search (AAS) in the AASPC method, we apply AAS to various distance functions on the Office-31 dataset.

Table 4.6: Comparison of PC with other distance functions on the Office-Home dataset with ResNet-50 as the backbone.

Measurement	Ar→Cl	Ar→Pr	Ar→Rw	Cl→Ar	Cl→Pr	Cl→Rw	Pr→Ar	Pr→Cl	Pr→Rw	Rw→Ar	Rw→Cl	Rw→Pr	Avg
None	43.41	66.55	74.64	56.61	63.98	65.32	53.36	39.36	72.64	64.73	46.30	76.55	60.29
Proxy $\mathscr{A}$-distance	43.21	65.44	74.85	55.09	62.51	65.37	52.33	38.63	72.83	64.57	46.23	76.66	59.81
KL-divergence	44.01	66.75	74.50	55.75	63.42	66.51	52.74	38.14	73.43	65.84	44.79	77.13	60.25
MMD	43.78	66.28	74.48	55.62	64.07	66.19	53.40	38.30	73.15	64.89	45.52	77.43	60.26
CORAL	44.15	65.85	74.16	55.42	63.01	66.83	52.95	39.38	72.53	65.14	45.96	77.07	60.20
CMD	44.40	65.92	74.50	54.68	63.37	67.07	52.78	38.88	72.94	65.64	45.29	77.36	60.24
PC	46.19	66.03	73.7	57.89	63.48	65.80	56.94	44.19	75.58	69.02	51.11	78.89	62.24

Table 4.7: Comparison of PC with other distance functions on the VisDA-2017 dataset with ResNet-50 as the backbone.

Measurement	Synthetic→Real
None	57.68
Proxy $\mathscr{A}$-distance	56.36
KL-divergence	56.27
MMD	58.76
CORAL	56.66
CMD	56.65
PC (Ours)	65.25

Specifically, we modify Algorithm 2 to search alignment architecture for other measurements by replacing the minus of the PC with other distance functions in $\mathscr{L}_{\text{PC}}$. According to experimental results shown in Figure 4.3, AAS can improve the performance of various distance functions on the Office-31 dataset, which demonstrates the effectiveness and generalization ability of the alignment architecture search.

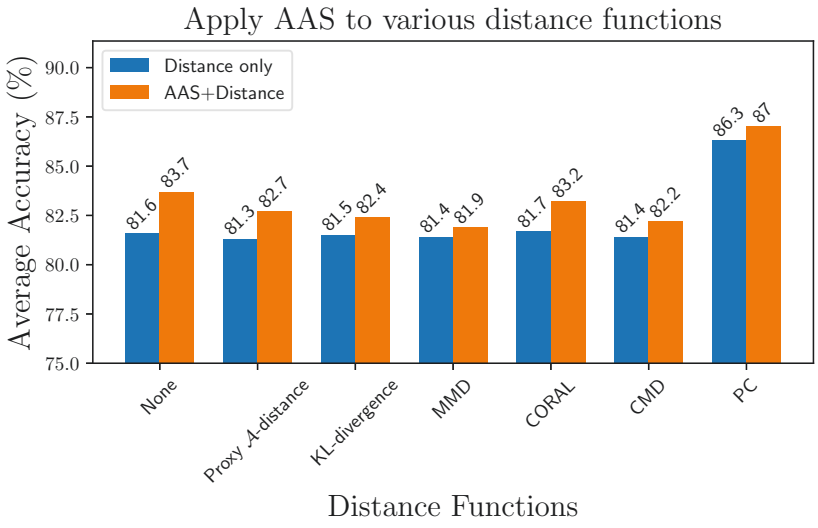


Figure 4.3: Apply AAS to various distance functions.

4.4.6 Hyper-parameter Sensitivity

We investigate the sensitivity with respect to two hyper-parameters: the training batch size and trade-off parameter λ in Eq. (4.2). We set the value of batch size from 32 to 256 and $\lambda \in \{0.1, 1, 10\}$ to obtain the performance change of AASPC. According to Figure 4.4, when the batch size is small, the performance becomes worse. This is because minimizing the loss in Eq. (4.1) may make samples in different classes close to each other, especially when λ is relatively large. The performance of AASPC is stable when the batch size is larger than 128 for both $\lambda = 0.1$ and $\lambda = 1$, which indicates that AASPC is relatively insensitive to a large batch size when λ is not so large, i.e., 0.1 or 1.

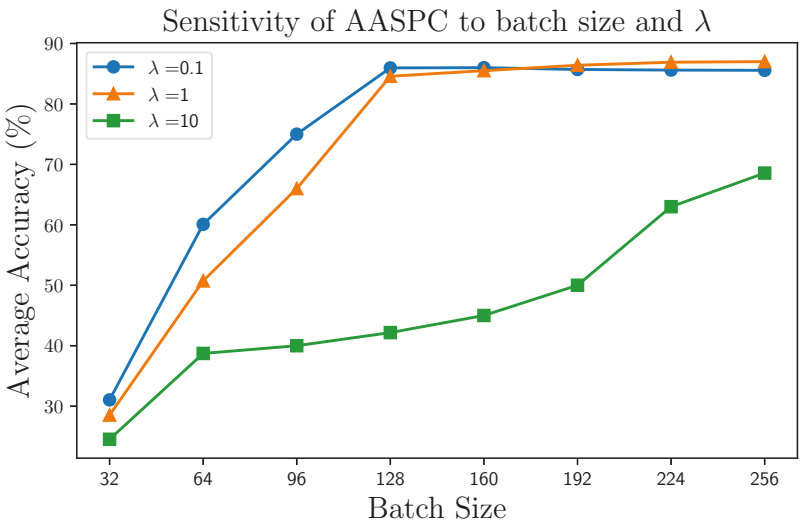


Figure 4.4: Sensitivity of AASPC to batch size and λ on the Office-31 dataset.

4.4.7 Complexity Analysis

In Table 4.8, we compare the training time and performance of AASPC and PC with the source only baseline, which does not compute any distance functions during the

training process. Compared with the source only baseline, the training time per epoch and occupied GPU memory slightly increase for PC and AASPC while the average accuracy dramatically improves. Hence, both PC and AASPC methods introduce negligibly additional computation costs for considerable performance improvement.

Table 4.8: Comparison of training complexity on Office-31 dataset. Training time per epoch and GPU memory consumption are recorded on one single NVIDIA Tesla V100S GPU with batch size 128.

Method	Time per Epoch (s)	GPU memory (M)	Avg Acc. (%)
Source Only	11.4	24865	65.3
PC	12.9	25085	86.3
AASPC	13.6	25185	87.0

4.4.8 Learned Architecture

Figure 4.5 shows the architecture found by AASPC for the transfer task D→W constructed on the Office-31 dataset. The left part of Figure 4.5 shows the search choice within the three cells found by the AASPC method and the right part of Figure 4.5 shows the connections among the three cells, PC and classifier. In Cell 0, the AASPC method chooses the FC layer with the same size as the input and the skip connection is connected to the batch-norm layer. In Cell 1, the choice of FC is the same as Cell 0 but the skip connection starts from the cell input. In Cell 2, the skip connection is the same as Cell 2 but the FC layer is of half size of the input. For connections between cells, the AASPC method chooses to use the output of Cell 0 to calculate the PC and the output of Cell 1 to calculate the classification loss. For a simple transfer task D→W, the searched architecture only has two cells, which indicates that the AASPC method can adaptively

learn architecture depending on the complexity of the DA task. Moreover, the location of the skip connection moves forward in Cell 1 and Cell 2 when compared with Cell 0, which helps reduce the network depth and alleviates the vanishing gradient problem.

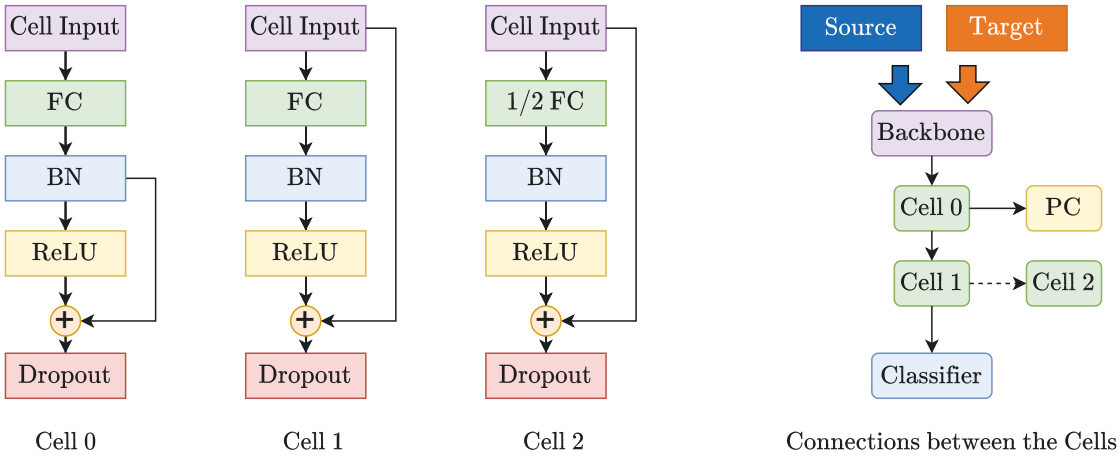
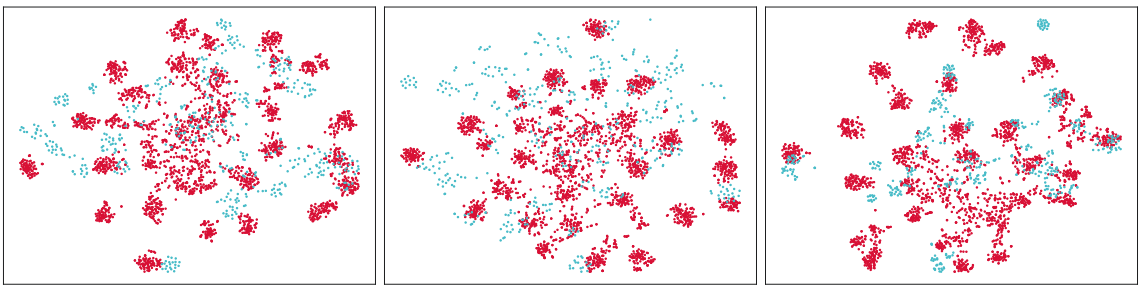


Figure 4.5: Searched architecture for transfer task D→W of the Office-31 dataset. Left: architecture within the three cells. Right: connections between the three cells, PC, and classifier.

4.4.9 Feature Visualization

In Figure 4.6, we visualize the hidden feature representations of the transfer task A→D constructed on the Office-31 dataset learned by source samples only, source and target samples with PC, and source and target samples with AASPC, respectively. According to Figure 4.6, we can see that samples with the representations learned by PC are more distinguishable than those by source only. The representations learned by AASPC are more separable than those by PC, which implies that the proposed AASPC method can learn discriminative and transferable feature representations for DA.



(a) Source Only (b) PC (c) AASPC (Ours)

Figure 4.6: t-SNE visualization of different methods for the transfer task A→D in the Office-31 dataset.

4.5 Summary

In this chapter, we propose a new PC function that can measure domain similarity. We further design the AASPC framework that searches deep alignment architecture for DA tasks. Experiments results on the Office-31, Office-Home, and VisDA-2017 datasets demonstrate the effectiveness of the proposed method. Moreover, the proposed AASPC framework has shown its potential to search alignment architecture for various DA methods. In our future studies, we try to extend the proposed AASPC framework to learn architecture for other DA methods and settings.

CHAPTER 5

LEARNING CONFLICT-NOTICED ARCHITECTURE FOR MULTI-TASK LEARNING

5.1 Introduction

In the last chapter, we investigate how to learn feature alignment architecture that can transfer knowledge across different domains. Moreover, MTL trains a model to perform multiple tasks simultaneously, involving knowledge sharing between multiple tasks. In this chapter, we introduce the conflict-noticing architecture learning method that learns architecture sharing knowledge across different tasks.

MTL [13, 152] aims to improve the generalization performance of a model on multiple learning tasks. Compared with single-task learning, MTL can learn multiple tasks simultaneously to reduce the overall training cost and gain knowledge sharing from

different tasks. Moreover, in some cases, MTL models could make multiple predictions in one forward propagation, which reduces the inference latency. Therefore, MTL has drawn much attention in recent years.

However, learning multiple tasks simultaneously can be challenging because parameter sharing may lead to negative transfer [112] with some empirical evidence in [66, 118, 126]. As empirically analyzed in [19, 26, 147], one main reason for this issue is the conflicting gradients of different tasks with respect to shared model parameters, which will be updated in opposite directions, leading to unsatisfactory performance. To mitigate the gradient conflict, several gradient manipulation methods [18, 75, 79, 139, 147] are proposed to manipulate task gradients by adjusting gradient magnitudes or gradient directions or both of them for all the tasks.

The occurrence of gradient conflict reflects that all the tasks are not strongly related to each other, given the adopted architecture. All the aforementioned works to alleviate the gradient conflict adopt the hard parameter sharing (HPS) architecture that uses a shared encoder for all the tasks with task-specific decoders for each task. Though the HPS architecture has been proven to be effective for many MTL problems and is widely used in MTL, it implicitly requires that all the tasks should be highly related to each other. For some complicated datasets such as the NYUv2 dataset [116], such requirement cannot be satisfied and the HPS architecture does not work well on this dataset [9, 123]. Hence, the gradient conflict issue could be the consequence of the use of an improper architecture for a given MTL problem.

To verify that, we compare the HPS architecture with the Learning to Branch (LTB)

Table 5.1: The ratio of negative cosine similarities (%) between the gradient of different tasks with respect to shared parameters. $\{f_S^1, \ldots, f_S^5\}$ denotes the modules in the sharing architecture while learning architecture. Ratio is calculated by going through all samples in the training set.

Module	f_S^1	f_S^2	f_S^3	f_S^4	f_S^5
HPS	59.26	51.68	42.22	39.36	49.49
LTB	41.08	38.86	39.65	47.05	24.44
CoNAL	0.34	0.19	0.14	0.13	0.00

method [45], which is to learn an architecture for a given MTL problem, on the NYUv2 dataset. We follow [31, 147] to calculate the cosine similarity between gradients of different tasks with respect to shared parameters and put detailed settings of this experiment in Section 5.3.2. According to Table 5.1, the gradient conflict in each module of the learned architecture in LTB is alleviated compared with the fixed HPS architecture. This suggests that architecture learning could be another way to mitigate the gradient conflict.

To the best of our knowledge, none of existing architecture learning methods for MTL considers to mitigate the gradient conflict during the architecture learning process, which is what we aim to do in this work. According to Table 5.1, we can see that the gradient conflict still severely affects the learned architecture in the LTB model. One possible reason is that LTB and other branch-based architecture learning methods include only partially-specific modules, which are first shared by all the tasks and then become specific for one or more tasks, in the search space. Based on its definition, during the architecture learning process, partially-specific modules will be affected by gradients of all the tasks and so weakly-related or even unrelated tasks will affect the learning of both parameters and architectures, leading to a suboptimal architecture which may suffer from gradient

conflict.

To kill two birds with one stone, we propose a Conflict-Noticed Architecture Learning method, which is to handle the gradient conflict during the architecture learning process. Different from existing architecture learning methods [8, 45, 89, 133, 156] for MTL, we are the first to introduce purely-specific modules into the search space of learnable architectures which also have all-shared modules shared by all the tasks. During the architecture learning process, the proposed CoNAL method could adaptively choose to use purely-specific modules when the gradient conflict is detected in the all-shared modules. As demonstrated in Table 5.1, the proposed CoNAL method achieves a very low ratio of the gradient conflict. Moreover, to efficiently handle MTL problems with a large number of tasks, we propose a progressive version of CoNAL called CoNAL-Pro, which can reduce the storage cost and find task subgroups during the architecture learning process. Experiments on Computer Vision (CV), Natural Language Processing (NLP), and Reinforcement Learning (RL) benchmark datasets demonstrate the effectiveness of the proposed methods. The main contributions of this paper are three-fold.

1. We propose the CoNAL method to mitigate the gradient conflict from the perspective of architecture learning.

2. In the CoNAL method, we firstly introduce the purely-specific modules in the search space of multi-task architecture.

3. Extensive experiments on three challenging domains demonstrate the effectiveness of the proposed methods.

5.2 The CoNAL Method

In this section, we introduce the proposed CoNAL method.

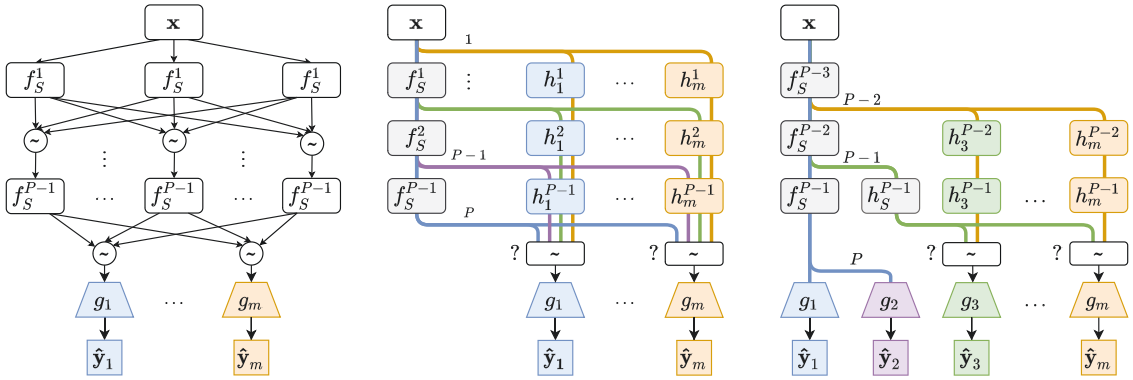


Figure 5.1: Illustration of the search space in various methods. (a), (b) and (c) represent for the search space of the LTB, CoNAL, and CoNAL-Pro methods, respectively. Blocks represent computational modules and edges between blocks denote data flows. Blocks with the grey color stand for all-shared modules and blocks with other colors are for task-specific modules. $\{f_S^1,\ldots,f_S^{P-1}\}$ denotes the all-shared modules for all the tasks. In figure (a), all-shared modules can transform into partially-specific modules at the later stage of learning. In figure (b) and (c), $\{1,\ldots,P\}$ denotes possible branch points in the search space. $\{h_i^1,\ldots,h_i^{P-1}\}$, $\{\alpha_i^1,\ldots,\alpha_i^P\}$, and g_i $(1 \le i \le m)$ denote the task-specific encoder, task-specific architecture parameters, and task-specific decoder, respectively, for task i. $\mathbf{x}$ and $\hat{y}_i$ denote the input data and the prediction for task i. A softmax operation is represented as "~" and "?" means to search within candidate architectures.

5.2.1 Search Space

As an architecture learning method, the CoNAL method defines a search space consisting of modules, each of which could be a fully connected layer or a sophisticated ResNet block/layer depending on the MTL problem under investigation. Different from existing architecture learning methods [8, 45, 89, 133, 156], which learn a tree-structured network architecture with partially-specific modules, all the modules in CoNAL are classified into two types: all-shared module and purely-specific module, which are defined as follows.

Definition 5.2.1 (Partially-specific module)**.** *When a module in the search space is updated by the gradient back-propagated from the loss of all tasks first and later from a fixed set of tasks during the search process, this module is said to be a* partially-specific module*.*

Definition 5.2.2 (All-shared module)**.** *When a module in the search space is always updated by the gradient back-propagated from losses of all the tasks during the search process, this module is said to be an* all-shared module*.*

Definition 5.2.3 (Purely-specific module)**.** *When a module in the search space is only updated by the gradient back-propagated from the loss of a fixed task during the search process, this module is said to be a* purely-specific module *for that task.*

According to Definition 5.2.3, we can see that parameters in a purely-specific module are only updated by the loss of one task, which is different from the partially-specific module defined in Definition 5.2.1. Purely-specific modules are a key ingredient for CoNAL to mitigate gradient conflict. Specifically, when the gradient of a task is detected to be conflicting in the all-shared module, this task could use purely-specific modules to form the encoder without affecting the learning of other tasks.

Formally, the search space of CoNAL is represented as a directed acyclic graph. For m learning tasks $\{\mathcal{T}_i\}_{i=1}^m$, the CoNAL method have a shared encoder network f_S consisting of $(P-1)$ all-shared modules for all the tasks, m task-specific architecture parameters $\{\boldsymbol{\alpha}_t\}_{t=1}^m$ to decide branch points for m tasks, and m task-specific encoder networks $\{h_t\}_{t=1}^m$ consisting of purely-specific modules for the m tasks. Hence, for task t, its model consists

of all-shared modules from f_S, purely-specific modules in h_t, $\boldsymbol{\alpha}_t = (\alpha_t^1, \ldots, \alpha_t^P)$, and a decoder network g_t. Here P is equal to the total number of all possible branch points before or after each module in f_S. As a binary parameter, $\alpha_t^p \in \{0, 1\}$ indicates whether task t branches at the branch point p from f_S to h_t. When α_t^p equals 1, there will be a branch to feed the output of the $(p-1)$-th module in f_S to the p-th module in h_t to form a network for task t, which is illustrated in Figure 5.1(b). Moreover, the sum of entries in $\boldsymbol{\alpha}_t$ should be 1 for any t, indicating that there is only one branch point for each task.

Figure 5.1 illustrates the difference between search spaces of some existing architecture learning works (i.e., LTB [45] and BMTAS [8]), and the CoNAL method. The search space of existing methods only has partially-specific modules. If task i is totally unrelated to other tasks, all the partially-specific modules will be updated by the gradient of the loss of task i, which can be detrimental to the performance of some other tasks. Differently, the CoNAL method could choose h_i for task i, making the learning of other tasks unaffected by task i. The proposed search space in the CoNAL method includes both single-task learning and HPS architectures as two extremes. When all the tasks are unrelated to each other, the network architecture in the CoNAL method could become separate networks by choosing $\{h_t\}$ for each task. For highly related or even identical tasks, the network architecture of the CoNAL method could become the HPS network by choosing f_S only. In most cases, the learned architecture by the CoNAL method is between the two extremes and more complex than them.

5.2.2 Architecture Learning

The CoNAL method is to find an architecture that circumvents gradient conflict for m tasks $\{\mathcal{T}_i\}_{i=1}^m$ by learning architecture parameters $\{\boldsymbol{\alpha}_t\}_{t=1}^m$.

Branch searching If task t branches at the branch point p of f_S to h_t, α_t^p is set to 1 and α_t^i is set to 0 for $1 \le i \le P,\ i \ne p$. Then in this case, the output of the entire encoder network for task t consisting of the first $(p-1)$ all-shared modules in f_S and the last $(P-p)$ purely-specific modules in h_t is $h_t(f_S(\mathbf{x}, p), p)$, where $f_S(\cdot, p)$ denotes the output of the $(p-1)$-th module in f_S and $h_t(\cdot, p)$ denotes the output of h_t starting from the p-th module. We can design some loss to learn $\{\boldsymbol{\alpha}_t\}$ but the discrete nature of $\{\boldsymbol{\alpha}_t\}$ will make stochastic gradient descent methods incapable to learn them. Here we relax $\{\boldsymbol{\alpha}_t\}$ to be continuous and use them to define the probability of branching at each branch point via the softmax function. Hence the output of the entire encoder network for task t is formulated as

$$\mathbf{o}_t(\mathbf{x}, \boldsymbol{\alpha}_t) = \sum_{p=1}^{P} \frac{\exp(\alpha_t^p)}{\sum_{p'} \exp(\alpha_t^{p'})} h_t(f_S(\mathbf{x}_t, p), p),$$

where $\mathbf{x}_t$ denotes the data for task t. Here $\mathbf{o}_t(\mathbf{x}, \boldsymbol{\alpha}_t)$ is just a convex combination of outputs of all possible branching architectures weighted by probabilities based on $\boldsymbol{\alpha}_t$. Then $\mathbf{o}_t(\mathbf{x}, \boldsymbol{\alpha}_t)$ is fed into the decoder g_t to generate the prediction and hence the loss for task t is formulated as

$$\mathscr{L}_t(\boldsymbol{\theta}_t, \boldsymbol{\alpha}_t) = l_t(\mathbf{y}_t, g_t(\mathbf{o}_t(\mathbf{x}, \boldsymbol{\alpha}_t)))$$

where $\boldsymbol{\theta}_t$ includes all the parameters in f_S, h_t, and g_t, $\mathbf{y}_t$ denotes the label of $\mathbf{x}_t$ in task t, and l_t denotes the loss function for task t. Then the total loss over m tasks is formulated

as

$$\mathscr{L}(\boldsymbol{\theta}, \boldsymbol{\alpha}) = \sum_{t=1}^{m} \mathscr{L}_t(\boldsymbol{\theta}_t, \boldsymbol{\alpha}_t), \tag{5.1}$$

where $\boldsymbol{\theta}$ denotes all the model parameters and $\boldsymbol{\alpha} = (\boldsymbol{\alpha}_1, \ldots, \boldsymbol{\alpha}_m)$.

Conflict noticing To mitigate the gradient conflict between tasks, we only update the purely-specific modules when the gradient conflict of any two tasks is detected. That is, we zero out the gradients on the current and successive all-shared modules if the cosine similarity of any two task gradients on this module is negative. Specifically, the conflict-noticed gradient of all-shared module p at each training iteration is formulated as

$$\text{grad}_t = \nabla_{\theta_S} \mathscr{L}_t(\boldsymbol{\theta_t}, \boldsymbol{\alpha_t}) \tag{5.2}$$

$$\nabla_{\theta_S^p} \mathscr{L}(\boldsymbol{\theta}, \boldsymbol{\alpha}) = \begin{cases} \sum_{t=1}^{m} \text{grad}_t^p, & \text{if } \cos(\text{grad}_i^{p'}, \text{grad}_j^{p'}) \geq 0, \forall i, j \ \forall p' \leq p \\ 0, & \text{otherwise} \end{cases} \tag{5.3}$$

where $\cos(\cdot, \cdot)$ computes the cosine similarity between two vectors, θ_S denotes parameters in all-shared modules, θ_S^p denotes parameters in the p-th all-shared module, grad_t denotes the gradient of the loss in task t with respect to θ_S, and grad_t^p denotes the gradient of the loss in task t with respect to θ_S^p. Different from existing methods [75, 147] which manipulate gradients of all the shared parameters as a whole, we split gradients of the shared encoder into fine-grained modules. According to Eq. (5.3), when there exist a pair of tasks (e.g., tasks i and j) such that $\cos(\text{grad}_i^p, \text{grad}_j^p) < 0$, then $\nabla_{\theta_S^p} \mathscr{L}(\boldsymbol{\theta}, \boldsymbol{\alpha}), \ldots, \nabla_{\theta_S^{p-1}} \mathscr{L}(\boldsymbol{\theta}, \boldsymbol{\alpha})$ will be set to zero. In this case, only the all-shared modules before the conflicting all-shared module (i.e., $\{f_S^1, \ldots, f_S^{p-1}\}$) and task-specific encoders

$\{h_t\}$ will be updated with unchanged gradients. Note that the conflict noticing operation needs to include purely-specific modules in the search space. Since the total gradient on the all-shared module is zero when conflict detected, no parameter after the conflict module will be updated without purely-specific module.

Algorithm 3 Learning Conflict-Noticed Architectures

Input: Dataset $\mathscr{D}_{tr}$ and $\mathscr{D}_{val}$
Output: Learned architecture parameter $\boldsymbol{\alpha}$

Uniformly initialized $\boldsymbol{\alpha}_0$, Pre-trained initialized $\boldsymbol{\theta}_0$;
$t := 0$;
while not converged **do**
    Sample a mini-batch from $\mathscr{D}_{val}$;
    Update $\boldsymbol{\alpha}_t$ by minimizing the upper-level subproblem of problem (5.4);
    Sample a mini-batch from $\mathscr{D}_{tr}$;
    Compute all task loss functions and the corresponding gradients according to Eq. (5.2); ;
    Compute conflict-noticed gradient for all-shared modules according to Eq. (5.3);
    Update $\boldsymbol{\theta}_{t+1}$ according to by minimizing the lower-level subproblem of problem (5.4);
    $t := t + 1$;
end while

Architecture determining Here architecture parameters $\boldsymbol{\alpha}$ are viewed as hyper-parameters and we adopt a bi-level formulation to learn both model and architecture parameters as

$$\begin{aligned} &\min_{\boldsymbol{\alpha}} \mathscr{L}_{val}(\boldsymbol{\theta}^*(\boldsymbol{\alpha}), \boldsymbol{\alpha}) \\ &\text{s.t. } \boldsymbol{\theta}^*(\boldsymbol{\alpha}) = \arg\min_{\boldsymbol{\theta}} \mathscr{L}_{tr}(\boldsymbol{\theta}, \boldsymbol{\alpha}), \end{aligned} \tag{5.4}$$

where $\mathscr{L}_{tr}(\cdot,\cdot)$ denotes the total loss defined in Eq. (5.1) on the training dataset and $\mathscr{L}_{val}(\cdot,\cdot)$ denotes the total loss on the validation dataset. We adopt the gradient-based hyperparameter optimization algorithm with the efficient first-order approximation as

in [35, 77] to solve problem (5.4). The architecture learning algorithm for the CoNAL method is summarized in Algorithm 3.

After solving problem (5.4), we can learn the branching architecture for task t by determining the branch point as

$$\boldsymbol{\alpha}_t^* = \arg\max_p(\{\alpha_t^p\}_{p=1}^P),$$

where $\boldsymbol{\alpha}_t^*$ is a P-dimensional one-hot vector with the maximum position being 1.

5.2.3 Retraining

After determining the branching architecture, the entire model is trained from scratch to obtain the final model. One noticeable advantage of our learned architecture is no need to manually-tune the weights between different tasks as in the prior architecture learning methods [45, 124]. This is because our model can balance the training of different tasks via the better architecture design rather than relies on the pre-setting weights to change the gradient magnitudes of different tasks. Inspired by [73], we randomly sample loss weights for task t from a uniform distribution in each iteration and the objective function is formulated as

$$\min_{\boldsymbol{\theta}} \mathscr{L}(\boldsymbol{\theta}, \boldsymbol{\alpha}^*) = \sum_{i=1}^{T}\sum_{t=1}^{m} w_t^i \mathscr{L}_t(\boldsymbol{\theta}_t, \boldsymbol{\alpha}_t^*), \tag{5.5}$$

where w_t^i is a sampled loss weight from the uniform distribution over $[0,1]$ for task t in the i-th iteration after normalizing to satisfy $\sum_{t=1}^m w_t^i = 1$, and T is the total number of iterations.

The retraining process of the CoNAL method can be straightforwardly combined with gradient manipulation strategies, which is further studied in Section 5.4.2.

5.2.4 A Progressive Extension

In the original search space of the proposed CoNAL method, $m+1$ separate networks in the full size need to be trained, where m is the number of tasks. The entire model size during the architecture learning process grows linearly with the number of tasks, which is computationally demanding when m is large. Another limitation of the CoNAL method is that the largest subgroup among multiple tasks could likely dominate the learning of the shared encoder in the training process. If multiple subgroups exist in the MTL problem, tasks in a smaller subgroup would not be effectively learned and utilized.

To effectively handle the aforementioned issues, we propose a progressive version of the CoNAL method called CoNAL-Pro, which can progressively learn the architecture for a large number of tasks with multiple subgroups. Specifically, the CoNAL-Pro method starts the architecture learning process from the last all-share module $p-1$. In the first stage, the search space only contains two branch points: p and $p-1$, for each task. For task t where $1 \leq t \leq m$, it either chooses to use the all-shared module f_S^{p-1} or use purely-specific module h_t^{p-1}. If task t chooses f_S^{p-1}, it will branch at branch point p and be removed from the search space in the following stages.

In this case, only the gradient conflict of the last all-share module needs to be calculated, which highly reduces the memory cost and enables the flexibility to handle a large number of tasks. Tasks branching out at this stage are considered as one subgroup and the number of tasks in the search space reduces after each stage. In the second stage, only tasks that use purely-specific modules are included in the search space, which contains two branch points: $p-1$ and $p-2$. Another shared module h_S^{p-1} is added after

the branch point $p-1$ for the second subgroup and it is shared by all the tasks in current search space. Figure 5.1(c) illustrates the second stage of the CoNAL-Pro method. This process repeats until all the tasks have been allocated to a certain branch. In the last stage, the search space will include purely-specific modules after branch point 1 for the remaining tasks, which indicates that similar to the CoNAL method, the CoNAL-Pro method could learn separate networks for those tasks.

5.3 Experimental Setup

In this section, we introduce experiment datasets in detail and provide all hyper-parameters used in the experiments.

5.3.1 Details of Datasets

CityScapes. The CityScapes dataset [21] consists of high resolution outside street-view images, which contains $2,975$ images for training and 500 images for validation. This dataset contains 19 classes for pixel-wise semantic segmentation, together with ground-truth inverse depth labels. By following [80], we evaluate the performance on the 7-class semantic segmentation and depth estimation tasks. All the images are resized to 128×256.

NYUv2. The NYUv2 dataset [116] consisting of RGB-D indoor scene images has 795 images for training and 654 images for validation. We evaluate the performance on three learning tasks: 13-class semantic segmentation, depth estimation, and surface normal

estimation. By following [80], all the images are resized to 288×384.

PASCAL-Context. The PASCAL-Context dataset [99] is an annotation extension of the PASCAL VOC 2010 challenge and it contains $4,998$ images for training and $5,105$ images for validation. We evaluate the performance on four tasks: 21-class semantic segmentation, 7-class human part segmentation, saliency estimation, and surface normal estimation, where the last two tasks are generated by [93].

Taskonomy. The Taskonomy dataset [149] which contains over 4.5 million indoor images from over $5,000$ buildings with 26 tasks. By following [118], we sample five learning tasks, including 17-class semantic segmentation, depth estimation, keypoint detection, edge detection, and surface normal estimation. We select 5 building images (i.e., "allensville", "collierville", "mifflinburg", "noxapater", and "onaga") from the standard tiny benchmark as the dataset, which contains $13,286$ images for training and $3,794$ images for validation.

CelebA. The CelebA dataset [82] is a large-scale face attributes dataset with 202,599 face images, each of which has 40 attribute annotations. It is split into three parts: 162,770, 19,867, and 19,962 images for training, validation, and testing, respectively. Hence, this dataset contains 40 tasks and each task is a binary classification problem for one attribute. The 9-tasks split includes 5oClockShadow, BlackHair, BlondHair, BrownHair, Goatee, Mustache, NoBeard, RosyCheeks, and WearingHat tasks.

XTREME benchmark. The XTREME benchmark [53] is a large-scale multilingual multi-task benchmark for cross-lingual generalization evaluation, which covers fifty languages and contains nine tasks. We conduct experiments on two tasks, i.e., Named

Entity Recognition (NER) and Part-Of-Speech (POS) tagging, from this benchmark. Following [73, 139], we construct a multilingual problem on each task by choosing the four languages with the largest numbers of data.

Meta-World. The Meta-World environment contains 50 different robotics continuous control and manipulation tasks. For multi-task learning, we choose MT10 challenge for learning 10 manipulation tasks simultaneously. The MT10 evaluation contains 10 tasks: reach, push, pick and place, open door, open drawer, close drawer, press button top-down, insert peg side, open window, and close window, where positions of objects and goal positions are fixed.

5.3.2 Implementation Details

Experiment of gradient conflict For the gradient conflict investigation in Table 5.1, we use the ratio of negative cosine similarities between the gradients of all pairs of tasks at each module for all the data as the metric to measure the magnitude of gradient conflict. We use ImageNet-1k [61] pretrained the Deeplab-ResNet50 [15] as encoders and the ASPP architecture [16] as decoders for the hard parameter sharing model. We follow [45] to setup the search space for the LTB model and train with warm-up and temperature. We train all methods for 200 epochs. Both LTB and CoNAL use the same initialization as HPS for all the modules in the search space. For LTB and HPS, we use the Adam optimizer with a learning rate as 10^{-4} and with weight decay 10^{-5}. While learning architecture, LTB relaxes the architecture parameters to be continuous and use them to define the probability of using different partially-specific modules at each branch

point via the Gumbel-softmax trick. We follow the [45] to warm-up the network for 20 epochs (i.e., only training model parameters) before learning architecture parameters and use the same optimizer to train architecture parameters with a temperature parameter $\tau = 2$ that decay exponentially with respect to the training epoch. For the CoNAL method, we relax architecture parameters to be continuous and use them to define the probability of branching at each branch point via the softmax function. We train model parameters and architecture parameters with the bi-level formulation in Eq. (5.4). We use the Adam optimizer with the learning rate as 10^{-4} and the weight decay 10^{-5} to train model parameters on half of the training data and another Adam optimizer with a learning rate of 5×10^{-5} and the weight decay 10^{-5} for optimizing architecture parameters on the other half of the training data.

Experiments on Multi-Task CV Benchmarks For the multi-task CV datasets used in Section 5.4.1, we employ the Deeplab-ResNet [15] pretrained on the ImageNet-1k dataset [61] with atrous convolutions as encoders and the ASPP architecture [16] as decoders. We use the Deeplab-ResNet50 for the CityScapes and NYUv2 datasets. We use the smaller Deeplab-ResNet18 for the larger PASCAL-Context and Taskonomy datasets for training efficiency. We set branch points before and after each ResNet layer. We use the cross-entropy loss for the semantic segmentation, human part segmentation, and saliency estimation tasks, the cosine similarity loss for the surface normal estimation task, and the L_1 loss for other tasks.

Architecture learning. For the architecture learning process of CoNAL on the four

multi-task CV datasets, we use the Adam optimizer with the learning rate as 10^{-4} with weight decay 10^{-5} for model parameters on half of the training data. We use another Adam optimizer with a learning rate of 5×10^{-5} with weight decay 10^{-5} for optimizing architecture parameters on the other half of the training data. We train for 200 epochs and use the batch size of 2, 2, 16, and 64 for the CityScapes, NYUv2, PASCAL-Context and Taskonomy datasets, respectively. We use the architecture parameters obtained in the last epoch to determine the learned architecture for retraining.

Retraining. For the retraining process of CoNAL, we use the Adam optimizer with the learning rate as 10^{-4} with weight decay 10^{-5} for training on the four multi-task CV datasets. We train for 200 epochs and use the batch size of 8, 8, 32, and 128 for on the CityScapes, NYUv2, PASCAL-Context, and Taskonomy datasets, respectively. We use the model parameters obtained the last epoch for the evaluation.

Settings for baselines. We follow the official implementation provided in the original paper for Cross-stitch [97], MTAN [80], NDDR-CNN [39], AFA [23], RotoGrad [56], MaxRoam [103], TSN [123], MTL-NAS [38], and BMTAS [8]. For methods that do not provide public implementation (i.e., LTB), we implement by ourselves and follow the settings described in the original paper. For each baseline method, we use a similar training setup to the CoNAL method and perform a grid search on the learning rate to report the best result.

Experiments on Multilingual Benchmark For the XTREME benchmark used in Section 5.4.4, we adopt a pretrained multilingual BERT (mBERT) [29] implemented

via the open-source transformers library [141] followed by five fully connected layers as the encoder and one fully connected layer as decoders. We set branch points before and after each fully connected layer in the encoder. The cross-entropy loss is used for these multilingual problems.

Architecture learning. For both NER and POS multilingual problems, the AdamW optimizer [88] with a learning rate as 2×10^{-5} is adopted for training model parameters on half of the training data. Another AdamW optimizer with a learning rate of 10^{-5} is adopted for learning architecture parameters on the other half of the training data. We use a batch size of 8 to train for 100 epochs. We use the architecture parameters obtained in the last epoch to determine the learned architecture for retraining.

Retraining. For both NER and POS multilingual problems, we use the AdamW optimizer with a learning rate as 2×10^{-5} for retraining. We use a batch size of 32 to train for 100 epochs. We use the model parameters obtained in the last epoch for evaluation.

Settings for baselines. We follow the official implementation in the original paper for the Uncertainty Weighting (UW) [58], PCGrad [147], and CAGrad [75] methods. We combine these methods with the HPS architecture and use a similar training setup to the CoNAL method.

Experiments on Multi-Task RL For the Meta-World environment, we use a policy network consisting of 2 module layers and one fully connected layer for the multi-head output. Each module layer contains one fully connected layer with 400 neurons. We set branch points before and after each module layer. We train all methods with 20 million

samples.

Architecture learning. We use the Adam optimizer with a learning rate 3×10^{-4} and a batch size of 4 to train model parameters. We use the Adam optimizer with a learning rate 3×10^{-4} and a batch size of 4 to train architecture parameters. We use the architecture parameters after training to determine the learned architecture for retraining.

Retraining. We use the Adam optimizer with a learning rate 3×10^{-4} and a batch size of 8 to train with 20 million samples. We use the model parameters after training for evaluation.

Settings for baselines. We follow the official implementation in the original paper of methods Soft Modularization [145], PCGrad and CAGrad methods. We combine PCGrad and CAGrad with HPS architecture and use a similar training setup for these methods as the one described for our method. We use equal weights for all task weighting.

Experiments for the CoNAL-Pro Method We use the ResNet-18 network as a shared feature extractor and a fully connected layer with two output units as a task-specific head for each task, which is different from the VGG-16 backbone in LTB. All the images are resized to 64×64. For the 9-task split, we follow [34] to filter the 40 annotated attributes to a set of 9 attributes. The cross-entropy loss is used for all methods.

Architecture learning. We use the Adam optimizer with the learning rate as 5×10^{-4} to train model parameters on half of the training data. We use the Adam optimizer with a learning rate of 2.5×10^{-5} to train architecture parameters on the other half of the train-

ing data. We use a batch size of 16 to train for 10 epochs and the architecture parameters obtained in the last epoch to determine the learned architecture for retraining.

Retraining. We use the Adam optimizer with a learning rate 5×10^{-4} and a batch size of 16 to train for 10 epochs. We use the model parameters obtained in the last epoch for evaluation.

Settings for baselines. The UW, PCGrad and CAGrad methods adopt the HPS architecture. For those methods, we use a similar training setup to the CoNAL method.

5.3.3 Evaluation Metrics

Training speedup and search speedup We compare the average training time for each method with the same batch size on a single NVIDIA Tesla V100 GPU. The training speedup is relative to the STL baseline. The search speedup is relative to the CoNAL method.

Multi-Task CV Benchmarks In the following, we introduce detailed evaluation metrics for all the tasks on the four CV datasets. For the semantic segmentation (**Segmentation**) task, we use pixel-wise labels to calculate the mean intersection over union (**mIoU**) and pixel accuracy (**Pix Acc**) as the evaluation metrics. For the depth estimation (**Depth**) task, we use the absolute error (**Abs Err**) and real error (**Rel Err**) in terms of the L_1 norm as evaluation metrics. For the surface normal estimation (**Surface Normal**) task, we calculate angle distances between the prediction and ground truth of all pixels and use the mean and median of the angle distances as the evaluation

metrics for the NYUv2 and Taskonomy datasets. We use the mean and root mean square error (**RMSE**) of the angle distances as evaluation metrics for the PASCAL-Context dataset. The percentages of pixels' prediction within the angles (**Within** t°) of 11.25°, 22.5°, and 30° to the ground truth are used as another type of evaluation metrics for the surface normal estimation task. For the human part segmentation (**Human Part**) and saliency estimation (**Saliency**) tasks, we use the mIOU and maximal F-measure (**maxF**) as evaluation metrics, respectively. For the keypoint detection (**Keypoint**) and edge detection (**Edge**) tasks, we use the absolute error as the evaluation metric.

Multilingual Benchmark For the experiments on the XTREME benchmark, by following [73] we calculate the F1 score on each language of the multilingual problem. Then, we average the F1 scores across four different languages in the multilingual problem as the performance. We run each method three times with different random seeds and report the average result with the standard deviation.

Multi-Task RL For the RL experiment on the Meta-World MT10 challenge, we evaluate each methods based on the mean success rate for executing tasks defined in the Meta-World environment [148]. By following [145], we evaluate the policy for 10 episodes per task per seed and report the average results with the standard deviation. Note that this is different from [75], which reports the highest average test performance of a method over 10 random seeds during the entire training stage.

Table 5.2: Performance on the CityScapes validation dataset, where the performance difference between each method and STL is reported. ↑ (↓) indicates the higher (lower) the result, the better the performance. The green color indicates that the corresponding method performs better than the STL method and the red color is defined oppositely. The number of parameters (abbreviated as Parms.) is calculated in MB.

Method	Segmentation		Depth		Δ_I ↑	Parms. (M)↓
	mIoU ↑	Pix Acc ↑	Abs Err ↓	Rel Err↓		
STL	68.13	91.28	0.0133	45.039	0	79.27
HPS	−0.73	−0.36	+0.0009	+0.3872	−2.3334	55.76
Cross-stitch	−0.12	+0.01	+0.0002	−0.6144	−0.1335	79.26
MTAN	+0.84	+0.31	+0.0003	−1.2882	+0.4866	72.04
NDDR-CNN	−0.11	−0.03	+0.0004	−0.1728	−0.7627	101.58
AFA	+0.79	+0.24	+0.0025	+2.1253	−5.5905	87.09
RotoGrad	+0.04	+0.09	+0.0002	−0.0612	−0.3034	57.86
MaxRoam	−1.41	+0.05	+0.0009	−2.7490	−0.7298	55.76
TSN	+1.62	+0.57	−0.0005	+0.8070	+1.2442	50.58
MTL-NAS	−2.78	−1.08	+0.0011	+2.8903	−5.0490	87.26
BMTAS	+1.44	+0.54	−0.0011	−0.7830	+3.1907	79.04
LTB	+1.58	+0.54	−0.0008	+1.6607	+1.2708	70.73
CoNAL	+1.54	+0.51	−0.0010	−2.3906	**+3.8870**	77.82

5.4 Experiments

In this section, we empirically evaluate the proposed CoNAL method. Details on the experimental setup are put in Section 5.3.

5.4.1 Experiments on Multi-Task CV Benchmarks

To demonstrate the effectiveness of the proposed CoNAL method, we conduct experiments on four CV benchmark datasets: **CityScapes** [21], **NYUv2** [116], **PASCAL-Context** [99], and **Taskonomy** [149]. The baseline methods in comparison include the Single-Task Learning (**STL**) that trains each task separately, the **HPS** architecture, manual architecture designed methods including **Cross-stitch** [97], **MTAN** [80], **NDDR-CNN** [39], **AFA** [23], and **RotoGrad** [56], and architecture learning methods such as **MaxRoam**

[103], **TSN** [123], **MTL-NAS** [38], **BMTAS** [8], and **LTB**. For fair comparison, we use the same backbone (with details in Section 5.3.2) for all the models in comparison.

In the four datasets, each task has multiple metrics for a thorough evaluation, where definitions of those metrics are put in Section 5.3.3. To better show the comparison among all the methods in comparison, we report the overall relative performance of each method over the STL baseline as

$$\Delta_I = 100\% \times \frac{1}{m}\sum_{t=1}^{m}\frac{1}{m_t}\sum_{j=1}^{m_t}\frac{(-1)^{p_{t,j}}(\mathrm{M}_{t,j}-\mathrm{STL}_{t,j})}{\mathrm{STL}_{t,j}},$$

where for a method M, $\mathrm{M}_{t,j}$ denotes its performance in terms of the jth evaluation metric for task t, $\mathrm{STL}_{t,j}$ is defined similarly, $p_{t,j}$ equals 1 if a lower value represents a better performance in terms of the jth metric in task t and 0 otherwise, and m_t denotes the number of evaluation metrics in task t.

Table 5.3: Performance on the NYUv2 dataset, where the performance difference between each method and STL is reported. ↑ (↓) indicates the higher (lower) the result, the better the performance. The green color indicates that the corresponding method performs better than the STL method and the red color is defined oppositely. The number of parameters (i.e., Parms.) is calculated in MB.

Method	Segmentation		Depth		Surface Normal					$\Delta_I \uparrow$	Parms. (M)↓
	mIoU ↑	Pix Acc ↑	Abs Err ↓	Rel Err↓	Angle Distance		Within t° ↑				
					Mean ↓	Median ↓	11.25	22.5	30		
STL	53.98	75.38	0.3945	0.1631	22.25	15.63	38.12	64.38	74.81	0	118.91
HPS	+0.50	+0.44	−0.0106	−0.0083	+1.25	+1.43	−2.81	−3.28	−2.67	-0.5088	71.89
CrossStitch	−0.52	+0.11	−0.0141	−0.0076	+0.76	+0.70	−1.11	−1.96	−1.79	+0.1493	118.89
MTAN	+0.76	+0.40	−0.0149	−0.0082	+0.72	+0.67	−1.21	−1.75	−1.49	+0.7658	92.35
NDDR-CNN	−0.14	−0.15	−0.0074	−0.0071	+0.35	+0.44	−0.45	−0.95	−0.89	+0.4106	169.10
AFA	−2.44	−1.47	+0.0085	+0.0061	+1.98	+1.77	−3.05	−4.16	−3.85	-4.7187	136.88
RotoGrad	+0.11	−0.13	−0.0145	−0.0061	+0.80	+0.87	−2.14	−2.51	−1.89	-0.1745	75.03
MaxRoam	+0.55	+0.23	−0.0066	+0.0069	−0.35	−0.65	−1.54	−1.50	−2.58	-0.4803	71.89
TSN	+0.35	+0.61	−0.0172	−0.0068	+1.51	+1.87	−3.68	−4.33	−3.40	-0.9746	50.58
MTL-NAS	−0.06	−0.17	−0.0098	−0.0101	+0.16	+0.38	+0.03	−0.17	−2.23	+0.9666	183.40
BMTAS	−0.14	+0.04	−0.0055	−0.0016	+0.00	−0.11	+0.49	−0.03	−0.12	+0.4793	116.04
LTB	−0.56	+0.05	−0.0040	−0.0095	−0.01	+0.08	−0.37	−0.29	−0.07	+0.8511	86.85
CoNAL	+0.08	+0.27	−0.0095	−0.0069	−0.26	−0.43	+1.20	+0.69	+0.34	**+1.7586**	93.96

Tables 5.2 and 5.3 as well as Tables 5.9 and 5.10 in Section 5.4.7 show the performance of all methods in comparison on the four CV benchmark datasets. Compared with the

STL counterpart, the proposed CoNAL method improves the performance of all tasks in terms of all the evaluation metrics, while having smaller numbers of parameters in the learned architectures. This indicates that the CoNAL method circumvent negative transfer on those four CV datasets.

Although some manually designed architecture methods (e.g., MTAN) can also circumvent negative transfer on the CityScapes dataset, those methods perform worse than STL in some tasks of the other three larger datasets. For example, on the NYUv2 dataset, existing architecture learning methods (e.g., MTL-NAS) mitigate the negative transfer when compared with HPS but still exhibit inferior performance to the STL method on the surface norm prediction task. Moreover, the proposed CoNAL method achieves the best Δ_I on the four datasets when compared with baseline methods, which demonstrates the effectiveness of the proposed CoNAL method. Figure 5.2 visualizes the model size (in terms of the number of parameters) and the performance (in terms of Δ_I) of various MTL methods. The proposed CoNAL method learns an architecture with a medium model size and the best performance.

5.4.2 Combination and Comparison with Gradient Manipulation Methods

The retraining process of the CoNAL method can straightforwardly incorporate various gradient manipulation methods based on Eq. (5.5). To demonstrate that the performance of the CoNAL method can be improved even further, we combine the proposed CoNAL model with the PCGrad [147] and CAGrad [75] methods, which manipulate task gra-

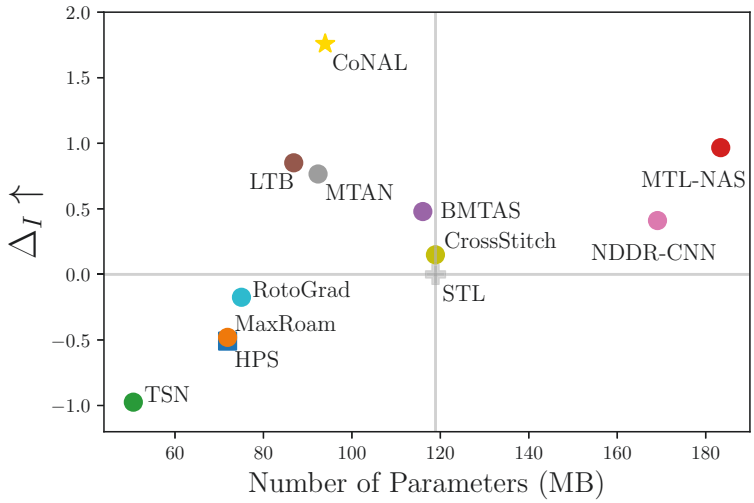


Figure 5.2: Performance and model size of various MTL methods on the NYUv2 dataset.

Table 5.4: Combination with gradient manipulation methods on the NYUv2 dataset. The training speedup is computed over the STL model.

Method	Train Speedup ↑	Δ_I ↑
STL	1.0x	0
HPS	1.69x	-0.5088
HPS-PCGrad	0.78x	-0.1205
HPS-CAGrad	0.61x	-0.1817
CoNAL	1.18x	+1.7586
CoNAL-PCGrad	0.42x	**+2.5271**
CoNAL-CAGrad	0.38x	+1.8932

dients to alleviate the gradient conflict. According to experimental results shown in Table 5.4, combining with PCGrad and CAGrad can further improve the performance of the proposed CoNAL model. Compared with the HPS-PCGrad method that improves about 0.4 over HPS in terms of Δ_I, CoNAL-PCGrad improves about 0.8 over CoNAL, which indicates that the architecture learned by the CoNAL method is more preferred than HPS while combining with the PCGrad method. Moreover, as the PCGrad and CAGrad methods are built on the HPS architecture, we can see that the CoNAL method

performs better than the PCGrad and CAGrad methods, which are just HPS-PCGrad and HPS-CAGrad in Table 5.4, and this result indicates that learning a suitable architecture could be more important to the performance improvement.

5.4.3 Ablation Study

We provide the ablation study for CoNAL in Table 5.5. For "w/o conflict-notice", we simply add purely-specific modules into the search space without the gradient noticing operation. For "w/o conflict-notice and pure", we replace purely-specific modules with partially-specific modules in the search space and does not conduct the gradient noticing operation. Compared with the original CoNAL method, the performance of those two variants degrades, which verifies the usefulness of the pure-specific modules and the gradient noticing operation.

Table 5.5: Ablation study of the CoNAL method on the NYUv2 dataset. Search speedup represents the relative time of the architecture learning computed over the CoNAL. The number of parameters (abbreviated as Parms.) is the model size of the learned architecture.

Method	$\Delta_I \uparrow$	Search Speedup↑	Parms. (M)↓
CoNAL	+1.7586	1.0x	93.96
-w/o conflict-notice	+1.1647	1.44x	86.85
-w/o conflict-notice and pure	-0.6773	1.46x	71.89
-w/o module splitting	+0.9729	1.07x	95.40
-w/o random loss weighting	+1.6494	1.00x	93.96

For "w/o module splitting", we replace all-shared and purely-specific modules with the entire all-shared and purely-specific encoders in the search space of CoNAL. This variant has a larger model size and worse performance than the CoNAL method, which verifies the usefulness of fine-grained modules. For "w/o random loss weighting", we

replace randomly sampled task loss weights with the same fixed weights as in LTB. This variant has slightly worse performance than the CoNAL method, which verifies that the random loss weighting can improve the performance a bit, and this strategy can reduce the tedious cost to tune loss weights.

5.4.4 Experiments on Multilingual Benchmark

Table 5.6: Performance on the XTREME benchmark in terms of the F1 score. POS and NER denote two multilingual problems from the XTREME benchmark. 'en', 'zh', 'te', 'vi', 'de', and 'es' denote English, Mandarin, Telugu, Vietnamese, German, and Spanish, respectively. Results are averaged across three independent runs.

Method	POS (F1 Score)				
	en	zh	te	vi	Avg. F1 Score ↑
STL	94.91 ± 0.09	88.63 ± 0.06	91.28 ± 0.47	86.74 ± 0.09	90.39 ± 0.14
HPS	94.89 ± 0.08	88.44 ± 0.04	90.95 ± 0.37	86.89 ± 0.23	90.29 ± 0.13
UW	94.54 ± 0.48	88.60 ± 0.21	90.82 ± 0.20	87.50 ± 0.27	90.36 ± 0.20
PCGrad	94.71 ± 0.27	88.46 ± 0.30	90.71 ± 0.06	86.82 ± 0.23	90.18 ± 0.05
CAGrad	94.69 ± 0.25	88.73 ± 0.12	91.30 ± 0.88	86.86 ± 0.20	90.39 ± 0.24
CoNAL	95.01 ± 0.08	88.78 ± 0.09	91.99 ± 0.03	86.89 ± 0.16	$\mathbf{90.67} \pm 0.04$
Method	NER (F1 Score)				
	en	zh	de	es	Avg. F1 Score ↑
STL	84.70 ± 0.12	82.56 ± 0.22	90.23 ± 0.23	92.62 ± 0.17	87.53 ± 0.08
HPS	85.10 ± 0.12	82.69 ± 0.23	90.55 ± 0.05	92.70 ± 0.04	87.76 ± 0.03
UW	84.36 ± 0.17	84.35 ± 0.05	90.37 ± 0.17	92.34 ± 0.23	87.85 ± 0.05
PCGrad	84.69 ± 0.13	82.44 ± 0.16	90.38 ± 0.24	92.70 ± 0.02	87.55 ± 0.09
CAGrad	84.50 ± 0.25	83.19 ± 0.59	90.59 ± 0.13	92.99 ± 0.20	87.82 ± 0.13
CoNAL	84.94 ± 0.16	83.10 ± 0.18	90.70 ± 0.16	93.04 ± 0.18	$\mathbf{87.95} \pm 0.10$

To show that the proposed CoNAL method works for NLP problems, we conduct experiments on two multilingual problems: Named Entity Recognition (NER) and Part-Of-Speech (POS) from the XTREME benchmark [53]. Different from heterogeneous MTL problems in CV datasets where different tasks share the same input data but are of different types with respective loss functions, each language/task in multilingual

Table 5.7: Comparison on mean success rates for MT10 tasks. Results are calculated for three independent runs with different seeds.

Method	Train Speedup ↑	Mean Success Rate ↑
Single-task policy	1.0x	0.78±0.042
Multi-task SAC	7.50x	0.44±0.060
Multi-head SAC	6.98x	0.58±0.069
Soft Modularization	5.36x	0.68±0.088
PCGrad	3.09x	0.65±0.092
CAGrad	3.28x	0.73±0.068
CoNAL	6.25x	**0.76**±0.049

problems has its own input data but is of the same type with the same loss function (i.e., the cross-entropy loss for classification tasks), which is just the homogeneous MTL problem [152]. The evaluation measure is the F1 score.

We also compare with the uncertainty weighting **(UW)** [58] method, which is to learn loss weighting for different tasks. Experimental results in Table 5.6 show that the CoNAL method performs better than STL on all the tasks, which demonstrates the effectiveness of the CoNAL method on NLP problems. Moreover, the CoNAL method achieves the best performance in terms of the average F1 score.

5.4.5 Experiments on Multi-Task RL

To further exam the proposed CoNAL method for RL tasks, we evaluate it on the MT10 challenge from the Meta-World environment [148]. By following [117, 145], we train the policy with Soft Actor-Critic (SAC) [48] and compare with multi-task SAC (i.e., SAC with a shared model), multi-head SAC (i.e., SAC with a shared policy network and task-specific head), Soft Modularization [145], PCGrad [147] and CAGrad [75].

According to the results shown in Table 5.7, the proposed CoNAL method outperforms

Table 5.8: Results on the CelebA 9-task dataset. The mean and standard deviation of the total test error are calculated over three independent runs.

Method	Parms. (M) ↓	Train Speedup ↑	Total Test Error ↓
STL	100.56	1.0x	49.59±0.12
HPS	11.21	8.84x	49.78±0.26
UW	11.21	8.34x	49.47±0.86
PCGrad	11.21	2.59x	48.66±0.48
CAGrad	11.21	2.44x	48.75±0.71
TAG-2	22.37	1.29x	49.23±0.05
LTB	77.39	2.10x	49.47±0.45
CoNAL	57.86	2.51x	49.02±0.45
CoNAL-Pro	30.58	3.18x	**48.63**±0.40

most baseline methods and the significant t-test with 95% confidence shows that the CoNAL method performs better than PCGrad, CAGrad and Soft Modularization. Though the single-task policy performs slightly better on one task (i.e., the pick-place task), the CoNAL method achieves comparable performance with the single-task policy in terms of the mean success rate and has a much faster training speed.

5.4.6 Experiments for the CoNAL-Pro Method

To evaluate the proposed CoNAL-Pro method, we conduct experiments on the CelebA dataset [82]. We follow [34] to filter the 40 tasks down to 9 tasks, leading to CelebA 9-task dataset, which has a larger number of tasks with multiple task subgroups.

On the CelebA 9-task dataset, we compare the proposed CoNAL and CoNAL-Pro methods with the HPS method, the **TAG-2** method [34] that performs the two-split task grouping, **LTB** [45], PCGrad, and CAGrad. According to experimental results shown in Table 5.8, the CoNAL-Pro methods has the lowest total test error, which demonstrates

the effectiveness of the proposed method. The model architecture learned by the CoNAL-Pro method has fewer parameters but better performance than the LTB method as it learns more subgroups. This demonstrates that the CoNAL-Pro method can obtain an architecture that can learn task grouping for better knowledge sharing among tasks. According to grouping results of the CoNAL-Pro method in Section 5.4.10, we can see that some tasks such as BrownHair and WearingHat are grouped into one subgroup and other tasks such as BlondHair and NoBread are in another subgroup, which matches our intuition.

Table 5.9: Performance of various models on the PASCAL-Context dataset, where the performance difference between each method and STL is reported. ↑ (↓) indicates the higher (lower) the result, the better the performance. The green color indicates that the corresponding method performs better than the STL method, and the red color is defined oppositely. The number of parameters (abbreviated as Parms.) is calculated in MB.

Method	Segmentation	Human Part	Saliency		Surface Normal					Δ_I ↑	Parms (M)↓
					Angle Distance		Within t° ↑				
	mIoU↑	mIoU↑	mIoU↑	maxF↑	Mean ↓	RMSE ↓	11.25	22.5	30		
STL	65.16	59.08	64.94	76.91	16.44	24.97	47.56	80.29	89.81	0	63.60
HPS	−0.07	+0.05	−0.26	+0.53	+1.00	+0.98	−4.67	−3.58	−2.20	-1.3038	34.79
Cross-stitch	−0.19	−0.45	−0.48	+0.16	+0.12	+0.35	−1.50	−2.18	−0.82	-0.7758	79.46
MTAN	−0.60	+0.00	−0.37	+0.33	+0.70	+0.79	−3.72	−2.81	−1.82	-1.2867	36.61
NDDR-CNN	+0.12	+0.10	+0.15	+0.56	−0.02	+0.14	−0.64	−0.44	−0.37	+0.0706	69.25
AFA	+2.10	+1.61	−1.87	−3.40	+1.24	+1.18	+6.54	+4.92	+3.10	+1.1272	199.14
TSN	+1.61	−1.17	+0.16	+0.05	+0.23	+0.19	−4.57	−3.36	−1.80	-0.7368	26.85
BMTAS	−0.16	−0.11	−0.28	+0.41	−0.33	−0.04	+0.52	+0.22	+0.11	+0.0935	45.28
CoNAL	+0.80	+0.78	+0.44	+0.87	−0.43	−0.11	+0.78	+0.42	+0.21	+1.1388	63.15

5.4.7 Experimental Results on PASCAL-Context and Taskonomy Datasets

The results on the PASCAL-Context and Taskonomy datasets are shown in Tables 5.9 and 5.10. The proposed CoNAL method achieves the best Δ_I on these two datasets when compared with baseline methods, which demonstrates the effectiveness of the

Table 5.10: Performance of various models on the Taskonomy dataset, where the performance difference between each method and STL is reported. ↑ (↓) indicates the higher (lower) the result, the better the performance. The green color indicates that the corresponding method performs better than the STL method, and the red color is defined oppositely. The number of parameters (abbreviated as Parms.) is calculated in MB.

Method	Segmentation		Depth		Keypoint	Edge	Surface Normal					Δ_I ↑	Parms (M)↓
							Angle Distance		Within t° ↑				
	mIoU ↑	Pix Acc ↑	Abs Err ↓	Rel Err↓	Abs Err ↓	Abs Err ↓	Mean ↓	Median ↓	11.25	22.5	30		
STL	65.49	97.56	0.0069	0.0115	0.1303	0.1341	10.62	4.26	73.46	85.4	90.17	0	79.50
HPS	+0.26	+0.12	+0.0014	+0.0020	−0.0215	+0.0012	+0.46	+0.38	−1.73	−0.28	−0.44	-2.7039	34.79
Cross-stitch	+0.80	+0.74	−0.0002	−0.0007	−0.0300	−0.0011	−1.73	−0.24	+0.63	+1.19	+0.65	+4.1831	79.46
MTAN	+0.27	+0.76	+0.0014	+0.0021	+0.0048	+0.0119	−1.22	+0.11	+2.51	+3.70	+2.76	+0.6377	36.61
NDDR-CNN	+0.57	+0.76	+0.0001	+0.0030	−0.0333	−0.0035	−2.09	−0.54	+0.69	+1.23	+0.68	+3.9309	88.32
AFA	+0.77	+0.64	+0.0019	+0.0029	−0.0665	−0.0516	−1.31	+0.10	+0.51	+1.14	+0.62	+3.0849	242.1
LTB	+0.23	+0.08	−0.0012	−0.0022	−0.0238	+0.0007	−0.69	−0.76	+1.01	+1.04	+0.48	+5.1795	43.19
CoNAL	+0.31	+0.07	−0.0010	−0.0019	−0.0240	−0.0004	−0.80	−0.85	+1.26	+1.19	+0.58	+5.4030	54.36

proposed CoNAL method. Note that some baseline methods (e.g., LTB) did not report their performance on some datasets (e.g., PASCAL-Context dataset) in their original paper and hence we did not include them in comparison. Detail of the hyper-parameters used in the experiments are provided in Section 5.3.2.

5.4.8 Effectiveness of the Architecture Learning Algorithm in CoNAL

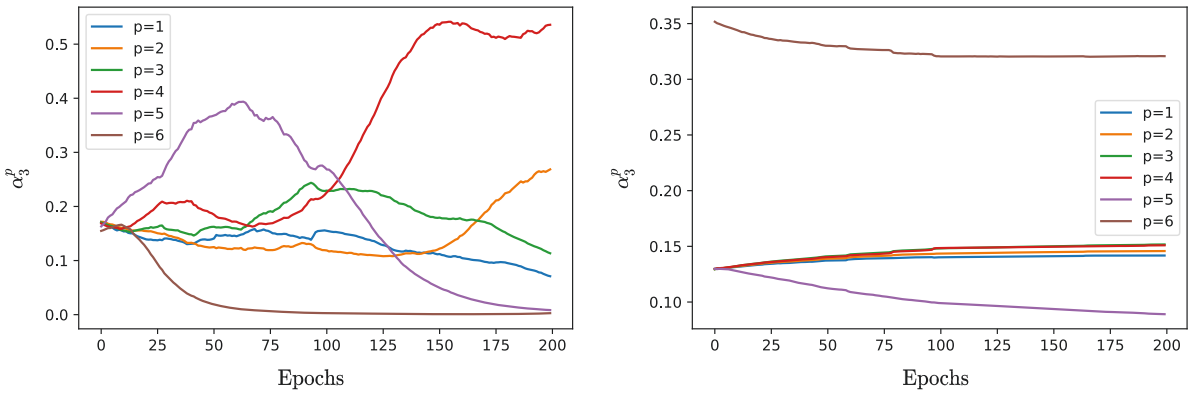


Figure 5.3: The change of architecture parameters $\{\alpha_3^p\}_{p=1}^P$ learned in the CoNAL method (Left) and the CoNAL$_{sl}$ method (Right), respectively, over epochs for the surface normal estimation task on the NYUv2 dataset.

Table 5.11: Comparison under similar model capacities on the NYUv2 dataset.

Method	Parms (M) ↓	Δ_I ↑
STL	118.91	0
CoNAL+	118.53	**+2.6966**
HPS+	94.23	+1.1551
CoNAL	93.94	+1.7586

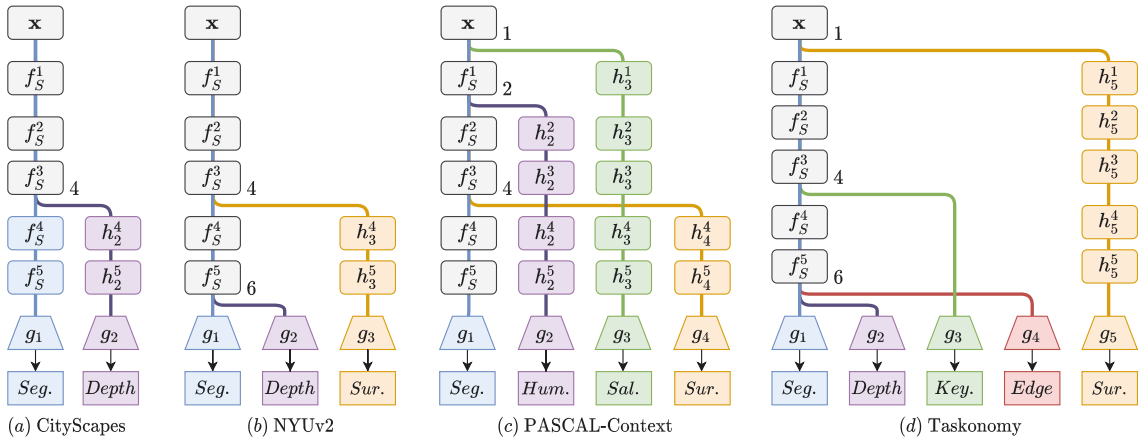
To show the usefulness of problem (5.4) in the architecture learning process, we compare with a variant of CoNAL called CoNAL$_{sl}$ that use a single-level optimization problem to utilize all the data, including both training and validation data, to learn both model parameters and architecture parameters. That is, the CoNAL$_{sl}$ method minimizes $\mathscr{L}_{tr}(\boldsymbol{\theta}, \boldsymbol{\alpha}) + \mathscr{L}_{val}(\boldsymbol{\theta}, \boldsymbol{\alpha})$ with respect to both $\boldsymbol{\theta}$ and $\boldsymbol{\alpha}$. Figure 5.3 shows the change of architecture parameters $\{\alpha_3^p\}_{p=1}^P$ for the surface normal estimation task on the NYUv2 dataset for the CoNAL and CoNAL$_{sl}$ methods. According to the results, in the first 100 epochs for the CoNAL method, α_3^5 has the largest value, which means task 3 tends to split at branch point 5. Then, α_3^4 overpasses α_3^5, which means that task 3 splits one module earlier. This inspires us that the architecture learning process could proceed in a progressive way, which motivates the proposal of the CoNAL-Pro method in Section 5.2.4. Differently, in the CoNAL$_{sl}$ method, α_3^6 has the largest value and has little change during the architecture learning process. In this case, the CoNAL$_{sl}$ method is more likely to update model parameters of all-shared modules instead of choosing a better architecture to reduce the overall loss, which indicates that the bi-level formulation is more preferred.

5.4.9 Comparison under Similar Model Capacities

To compare the performance under comparable model capacity, we try to match the number of parameters of different models. For the CoNAL+ and HPS+ models, we enlarge their network capacities by adding additional residual blocks. Specifically, we add three additional residual blocks after the fourth module for the HPS+ model to match the model size of the CoNAL model. For the CoNAL+ model, we add four additional residual blocks after the fourth module (i.e., branch point 5) to match the model size of the STL model. According to the results shown in Table 5.11, we can see that with a similar network capacity to the STL model, CoNAL+ performs even better with +2.6966 for Δ_I. Moreover, with a similar network capacity to the CoNAL model, HPS+ performs better than HPS, whose performance is reported in Table 5.3, but inferior to CoNAL. This experiment shows that the good performance of CoNAL is due to not only the increased model capacity but also the better architecture.

5.4.10 Analysis of Learned Architectures

The learned architectures in all the experiments are shown in Figure 5.4 and Tables 5.12, 5.13, and 5.14. For the CityScapes dataset, two tasks split at branch point 4. For the NYUv2 dataset, the architecture found by CoNAL splits the surface normal estimation task at branch point 4, which is after the third module in the shared encoder. The semantic segmentation task and depth estimation task share all modules in the encoder. For the PASCAL-Context dataset, the surface normal estimation task is also split from the shared encoder at branch point 4, which is similar to the architecture on the NYUv2



(a) CityScapes (b) NYUv2 (c) PASCAL-Context (d) Taskonomy

Figure 5.4: The learned architecture of the proposed CoNAL method on four CV benchmark datasets. The number besides the branch indicates the index of the branch point in the search space. "Seg.", "Depth", "Sur.", "Hum.", "Sal.", "Key.", and "Edge" denote the semantic segmentation, depth estimation, surface normal estimation, human part segmentation, saliency estimation, keypoint detection, and edge detection tasks, respectively.

dataset. The saliency estimation task is split at branch point 1, which means it uses a separate network. For the Taskonomy dataset, the surface normal estimation task is split at branch point 1 to use a separate network and the keypoint detection task is split at branch point 4. All other tasks share all the modules in the shared encoder.

Table 5.12: The learned architecture of the proposed CoNAL method on the POS and NER problems from the XTREME benchmark. The total number of branch points P is set to 6. 'en', 'zh', 'te', 'vi', 'de', and 'es' denote English, Mandarin, Telugu, Vietnamese, German, and Spanish, respectively.

Branch Point	$p = 1$	$p = 2$	$p = 3$	$p = 4$
POS	vi		zh	en, te
NER	es	en, de	zh	

In the XTREME benchmark, the Mandarin tasks share the first two modules and split at branch point 3 for both POS and NER problems. Vietname and Spanish tasks split at branch point 1, which indicates that they use separate networks.

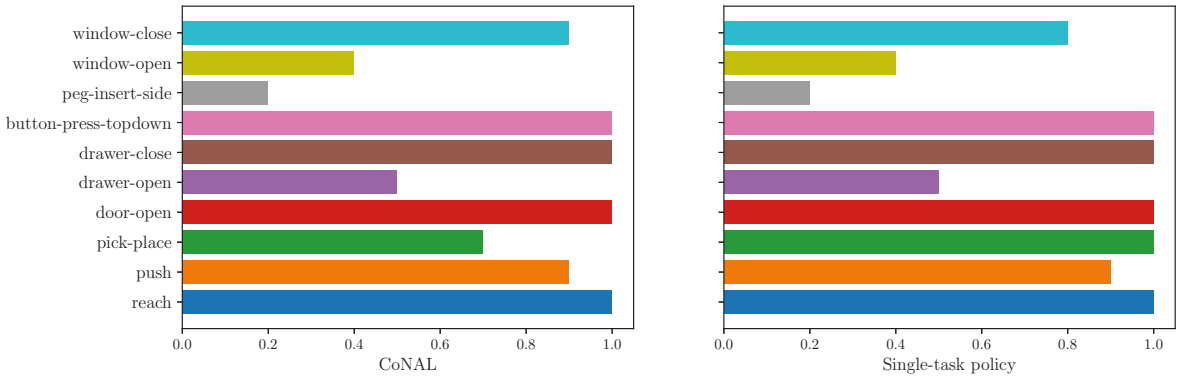


Figure 5.5: The mean success rate of each tasks in Meta-World MT10.

For Meta-World MT10 challenge in the RL experiment, tasks such as window-open and window-close share all modules while complex tasks such as button-press-topdown split at branch point 1. Figure 5.5 shows the performance of the CoNAL method and the single-task policy on each task. Though the single-task policy performs slightly better on the pick-place task, the CoNAL method achieves comparable performance with the single-task policy in terms of the mean success rate.

Table 5.13: The learned architecture of the proposed CoNAL method on Meta-World MT10 challenge. The total number of branch points P is set to 2.

Branch Point	Name of task
$p = 1$	reach, pick-place, drawer-close, button-press-topdown
$p = 2$	push, door-open, drawer-open, peg-insert-side, window-open, window-close

For the CelebA 9-task dataset, in the CoNAL-Pro method, some tasks such as Brown-Hair, Goatee and WearingHat are grouped into one subgroup which is at branch 4. Other tasks such as BlondHair and NoBread are in different subgroups at different branches.

Table 5.14: The learned architecture of the proposed CoNAL-Pro method on the CelebA dataset, where the total number of branch points P is set to 6.

Branch Point	9-task
$p = 0$	NoBeard
$p = 2$	BlondHair, RosyCheeks
$p = 4$	BrownHair, Goatee, WearingHat
$p = 5$	5oClockShadow, BlackHair, Mustache

5.4.11 Experiments on Synthetic Datasets

To demonstrate that the proposed CoNAL method can circumvent negative transfer, we design a synthetic dataset with controllable task relatedness. Inspired by [18, 45], we construct a regression task $T_{r,t}$ by generating target value as

$$\mathbf{y}_{r,t} = \mathscr{A}_r[(\mathbf{x} + \delta_t) * Z/\varphi],$$

where $\mathscr{A}_r \in \{\text{sinc}, \text{bent}, \text{square}\}$ stands for different element-wise activation functions that control the task relatedness, $\mathbf{x} \in \mathbb{R}^{100\times 200}$ denotes the 200-dimensional input data for all the tasks with each feature randomly sampled from a normal distribution $\mathcal{N}(0,10)$, $\delta_t \in \mathbb{R}^{100\times 200}$ is the task-specific noise randomly sampled from $\mathcal{N}(0,2)$, $Z \in \mathbb{R}^{200}$ is a constant vector with its entries randomly sampled from a normal distribution $\mathcal{N}(0,1)$, and φ equals the input dimension 200 for normalization. Tasks generated by the same activation function are considered highly related since they only differ in the noises and tasks generated by different activation functions are considered more unrelated.

We compare the proposed CoNAL method with the LTB and BMTAS methods which have partially-specific modules and behave similarly in this dataset. The shared, partially-specific and purely-specific encoders, i.e., f_S and h_t, have two modules, each

of which is a fully connected layer with five neurons followed by the ReLU activation function. The search space has three branch points for each task and they are located before the first module, after the first module, and after the second module, respectively. Another fully connected layer with one neuron is used as a task-specific decoder for each task. The mean square loss is used as the loss function, which is optimized by the SGD optimizer. Both model and architecture parameters are trained for 30 epochs with a learning rate 0.02.

5.4.11.1 Purely-Specific Modules Really Help

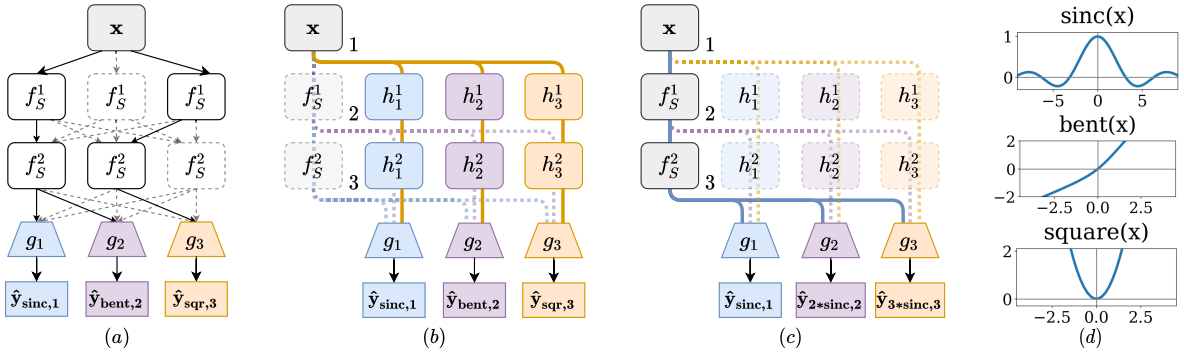


Figure 5.6: The comparison of learned architecture on the synthetic dataset. $\{1,2,3\}$ denotes available branch points in the search space. $\{f_S^1, f_S^2\}$ denotes the shared encoder for all the tasks. $\{h_1^1, \ldots, h_3^2\}$ and $\{g_1, g_2, g_3\}$ denote the task-specific encoders and decoders, respectively, for the three tasks. Solid lines indicate the final learned architecture and dotted lines indicate branches that are not selected.

In the first experiment, we generate three dissimilar tasks with the three activation functions, i.e., sinc, bent, and square. Figure 5.6(a) shows the learned architecture by the LTB and BMTAS methods, where all the modules in the search space are partially-specific modules. In the learned architecture, the first two tasks corresponding to the sinc and bent activation functions share the same encoder while the last task is learned separately. As shown in Figure 5.6(b), the learned architecture by the CoNAL method

only has three independent task-specific encoders for the three tasks, which indicates that it is better to learn these three tasks separately, which matches our expectation as the three tasks are dissimilar. This result shows that purely-specific modules can help identify the expected task relationship on this dataset even when negative transfer possibly occurs.

5.4.11.2 CoNAL Helps Find Compact Architecture

In the second experiment, we generate three tasks by using the same activation function (i.e., sinc) but with different scale multipliers 1, 2, and 3 (i.e., multiplying y_{sinc} by 1, 2, and 3, respectively). In this case, these three tasks are very similar. According to Figure 5.6(c), the learned architecture by the CoNAL method uses the shared encoder for all the tasks and this result indicates these three tasks are highly related and better learned together, which matches our intuition.

5.5 Summary

In this chapter, we propose the CoNAL method to learn multi-task network architectures to alleviate the gradient conflict issue. We first introduce purely-specific modules to the design of the search space and propose a conflict-noticed algorithm to mitigate gradient conflict. We further propose an extension of the CoNAL method to enable the learning on many tasks and the identification of multiple subgroups. We validate the CoNAL method on multiple MTL benchmarks across three challenging domains.

CHAPTER 6

DEEP SAFE MULTI-TASK LEARNING

6.1 Introduction

In the last chapter, CoNAL handles the gradient conflict during the architecture learning process to alleviate gradient conflict. However, it cannot guarantee to improve the performance of all the tasks compared with single-task learning. We introduce the deep safe multi-task learning method in this chapter to address this issue.

MTL has been widely used in many Computer Vision (CV) applications, such as human action recognition [74], face attribute estimation [49], age estimation [154], and dense prediction tasks [135]. Although MTL has demonstrated its usefulness in many applications, MTL cannot guarantee to improve the performance of all the tasks compared with single-task learning. Specifically, as empirically observed in [46, 66, 118, 124, 126], when learning multiple tasks together, many existing MTL models can achieve better

performance on some tasks than their single-task counterparts but underperform on the other tasks. Such phenomenon is called the *negative sharing* phenomenon in this chapter, which is similar to the 'negative transfer' phenomenon [138] in transfer learning [144] but with some differences as discussed later. One reason for the occurrence of *negative sharing* is that there are partially related or even unrelated tasks among tasks under the investigation, making jointly learning those tasks impair the performance of some tasks.

To the best of our knowledge, there is little work to study the *negative sharing* problem for MTL. To fill this gap, in this chapter, we firstly give a formal definition for *negative sharing* that could occur in MTL. Then we formally define an ideal and also basic situation for MTL called *safe multi-task learning*, where the generalization performance of an MTL model is no worse than its single-task counterpart on each task. That is, there is no *negative sharing* occurred. According to the definition of MTL [13, 152], we can see that every MTL model is required to achieve *safe multi-task learning*. Otherwise, single-task learning is more preferred than MTL, since an unsafe MTL model may bring the risk of worsening the generalization performance of some or even all the tasks. As true data distributions in multiple tasks are usually unknown so that *safe multi-task learning* is hardly to measure, we formally define *empirically safe multi-task learning* and *probably safe multi-task learning*, which are measurable.

To achieve *empirically*/*probably safe multi-task learning*, we propose a Deep Safe Multi-Task Learning (DSMTL) model whose architecture consists of a public encoder shared by all the tasks and a private encoder for each task. The public encoder and a

private encoder of a task are combined via a gating mechanism to form the entire encoder for that task. To train the DSMTL model, we propose two learning strategies: individual learning (denoted by DSMTL-IL) and joint learning (denoted by DSMTL-JL), which learn model parameters separately and jointly, respectively. For those two strategies, we provide theoretical analyses to show that they can achieve some versions of both *empirically safe multi-task learning* and *probably safe multi-task learning*.

To improve the scalability of the DSMTL model with respect to the number of tasks, we propose an extension called DSMTL with Architecture Learning (DSMTL-AL), which leverages neural architecture search to learn a more compact architecture with fine-grained modular splitting. Specifically, we allow the DSMTL-AL model to learn where to switch to the private encoder while forwarding in the public encoder. In this way, the DSMTL-AL model can save the first few modules in the private encoders and hence improve the scalability.

Extensive experiments on benchmark datasets, including CityScapes, NYUv2, PASCAL-Context, and Taskonomy, demonstrate the effectiveness of the proposed DSMTL-IL, DSMTL-JL, and DSMTL-AL methods.

The main contributions of this chapter are summarized as follows.

- We provide formal definitions for MTL, including *negative sharing*, *safe multi-task learning*, *empirically safe multi-task learning*, and *probably safe multi-task learning*.

- We propose the simple and effective DSMTL model with two learning strategies, which is guaranteed to achieve some versions of *empirically*/*probably safe multi-*

task learning.

- We propose the DSMTL-AL method, which is an extension of the DSMTL methods, to learn a compact architecture with good scalability.
- Extensive experiments demonstrate that empirically the proposed methods can achieve *safe multi-task learning* and that they outperform state-of-the-art baseline models.

6.2 Related Work

MTL has been extensively studied in recent years [33, 44, 63, 153, 155]. How to design a good network architecture for MTL is an important issue. The most widely used architecture is the multi-head hard sharing architecture [13, 81, 113], which shares the first several layers among all the tasks and allows the subsequent layers to be specific to different tasks. Then, to better handle task relationships, different MTL architectures have been proposed. For example, [97] proposes a cross-stitch network to learn to linearly combine hidden representations of different tasks. [91] proposes a multi-gate mixture-of-experts model which adopts the mixture-of-experts model by sharing expert submodels across all tasks, while having a gating network trained to optimize each task. [80] proposes a Multi-Task Attention Network (MTAN), which consists of a shared network and an attention module for each task so that both shared and private feature representations can be learned via the attention mechanism. [39] proposes a Neural Discriminative Dimensionality Reduction (NDDR) layer to enable automatic

feature fusing at every layer for different tasks. [124] proposes an Adaptive Sharing (AdaShare) method to learn the sharing pattern through a policy that selectively chooses which layers to be executed for each task. [23] proposes an Adaptive Feature Aggregation (AFA) layer, where a dynamic aggregation mechanism is designed to allow each task to adaptively determine the degree of the knowledge sharing between tasks. PS-MCNN [12] adopts both shared network and task-specific network by performing the concatenation operation after each block to learn shared and task-specific representations. PLE [126] separates shared components and task-specific components explicitly and adopts a progressive routing mechanism to extract semantic knowledge gradually for MTL. Some routing-based methods are proposed, including Multi-Agent Reinforcement Learning (MARL) [111] that allows the MTL network to dynamically self-organize its architecture in response to the input, Stochastic Filter Groups (SFG) [7] that assigns convolution kernels in each layer to the "specialist" or "generalist" group, and Task Routing Layer (TRL) [119] that allows for a single model to fit to many tasks within its parameter space with task-specific masking.

Instead of hand-crafting architectures for MTL, there are some works to leverage techniques in NAS [77] to automatically search MTL architectures with good performance. For example, [89] dynamically widens a multi-layer network to create a tree-like deep architecture, where similar tasks reside in the same branch. [72] proposes an evolutionary architecture search algorithm to search blueprints and modules that are assembled into an MTL network. [38] searches inter-task layers for better feature fusion across tasks. [45] proposes a differentiable architecture search algorithm to learn branching blocks to

construct a tree-structured neural network for MTL. [8] automatically determines the branching architecture for the encoder in a multi-task neural network under resource constraints. [46] aims to learn where to share or branch within a network for multiple tasks. [123] designs a task switching network that can learn to switch between tasks with a constant number of parameters which is independent of the number of tasks. All the aforementioned works do not study how to achieve *safe multi-task learning*, which is the focus of this chapter.

The safeness of machine learning methods has drawn attention in recent years [42, 70, 71, 127, 128, 129]. [71] proposes a safe semi-supervised support vector machine that performs no worse than the supervised counterpart, leading to the safeness in the use of unlabeled data. [70] addresses the safe weakly supervised learning problem by integrating multiple weakly supervised learners, which is guaranteed to derive a safe prediction under a mild condition. [42] proposes a safe deep semi-supervised learning method to alleviate the harm caused by class distribution mismatch. Moreover, there are some works [127, 128, 129] to address the safeness in multi-view clustering. [129] proposes reliable multi-view clustering, which empirically performs no worse than its single-view counterpart and proves that its performance will not significantly degrade under some assumptions. [128] proposes deep safe multi-view clustering to reduce the risk of performance degradation caused by view increasing and hence to guarantee to achieve the safeness in multi-view clustering. [127] proposes a bi-level optimization framework to achieve safe incomplete multi-view clustering. Different from the aforementioned works that address the safeness in semi-supervised learning, weakly

supervised learning, and multi-view clustering, our work focuses on the safeness in multi-task learning.

6.3 Definitions

In this section, we formally introduce some definitions to measure the safeness in MTL.

We first define the *negative sharing* phenomena.

Definition 6.3.1 (Negative Sharing)**.** *For an MTL model which is trained on multiple learning tasks jointly, if its generalization performance on some tasks is inferior to the generalization performance of the corresponding single-task counterpart that is trained on each task separately, then* negative sharing *occurs.*

Remark 1. Negative sharing *could occur when some tasks are partially or totally unrelated to other tasks. In this case, manually enforcing all the tasks to have some forms of sharing will impair the performance of some or even all the tasks. In Definition 6.3.1, the MTL model and its single-task counterpart usually have similar architectures, since totally different architectures could bring additional confounding factors. Moreover,* negative sharing *is similar to negative transfer [138] in transfer learning [144]. However, knowledge transfer in transfer learning is directed as it is from a source domain to a target domain, while knowledge sharing in MTL is among all the tasks, making it usually undirected. From this perspective,* negative sharing *is different from negative transfer.*

Definition 6.3.2 (Safe Multi-Task Learning)**.** *When* negative sharing *does not occur for an MTL model on a dataset, this MTL model is said to achieve* safe multi-task learning

on this dataset.

Remark 2. Safe multi-task learning *is an ideal situation for an MTL model to achieve. However, the generalization performance is hard to evaluate during the learning process, so it is hard to determine whether an MTL model can achieve* safe multi-task learning.

As the empirical/training loss is easy to compute during the learning process, we present the following definition based on the empirical loss to measure the empirical safeness of MTL models.

Definition 6.3.3 (Empirically Safe Multi-Task Learning)**.**
(1) *If the empirical loss of an MTL model on each task is no larger than that of its single-task counterpart, this MTL model is said to achieve* empirically individual safe multi-task learning.
(2) *If the average of empirical losses of an MTL model on all the tasks is no larger than that of its single-task counterpart, this MTL model is said to achieve* empirically average safe multi-task learning.

Remark 3. *In Definition 6.3.3, we define two versions of* empirically safe multi-task learning. *It is easy to see that an MTL model satisfying* empirically individual safe multi-task learning *can achieve* empirically average safe multi-task learning *but not vice versa, which indicates that* empirically individual safe multi-task learning *is weaker than* empirically individual safe multi-task learning. *Even though* empirically safe multi-task learning *is easy to measure based on the empirical loss of each task, an MTL model that achieves* empirically safe multi-task learning *cannot have guarantee to achieve* safe

multi-task learning*, since there is a gap between the empirical loss and the expected loss that is to measure the generalization performance. Hence,* empirically safe multi-task learning *is a loose version of* safe multi-task learning.

With m learning tasks in an MTL problem, $\mathscr{E}_t$ and $\mathscr{E}_t^{\mathrm{STL}}$ denote the expected losses of an MTL model and its single-task counterpart on task t, respectively. The corresponding average expected losses are denoted by $\mathscr{E}$ and $\mathscr{E}^{\mathrm{STL}}$, respectively. n denotes the average number of samples in all the tasks. With the above notations, we can define *probably safe multi-task learning* as follows.

Definition 6.3.4 (Probably Safe Multi-Task Learning)**.**

(1) *For an MTL model trained on m tasks, if for* $0 < \delta < 1$*, there exist m constants* $\epsilon_t \geq 0$ *such that* $\mathscr{E}_t + \epsilon_t \leq \mathscr{E}_t^{STL} + \rho_n^t$ *holds with at least probability* $1-\delta$ *for any* $t \in [1, m]$*, where* ρ_n^t *is a function of n satisfying* $\lim_{n\to+\infty} \rho_n^t = 0$*, then this MTL model is said to achieve* probably individual safe multi-task learning.

(2) *If for* $0 < \delta < 1$*, there exists a constant* $\epsilon \geq 0$ *such that* $\mathscr{E} + \epsilon \leq \mathscr{E}^{STL} + \rho_n$ *holds with at least probability* $1-\delta$*, where* ρ_n *is a function of n satisfying* $\lim_{n\to+\infty} \rho_n = 0$*, then this MTL model is said to achieve* probably average safe multi-task learning.

Remark 4. *Different from* empirically safe multi-task learning *which is measured based on empirical losses,* probably safe multi-task learning *is based on expected losses, making it a tighter approximation of* safe multi-task learning *than* empirically safe multi-task learning. *Compared with* safe multi-task learning*,* probably safe multi-task learning *is easy to be measured based on some analysis tool as verified in Section 6.5. According to*

Definition 6.3.4, it is easy to see when n is large enough, probably individual safe multi-task learning *could become* safe multi-task learning *in a large probability. Between the two versions of* probably safe multi-task learning*, similar to* empirically safe multi-task learning*,* probably average safe multi-task learning *is weaker.*

As discussed above, *empirically safe multi-task learning* and *probably safe multi-task learning* are two measures for an MTL model to achieve but few works can guarantee to achieve that. In the next section, we will propose the DSMTL model with such guarantees under mild conditions.

6.4 DSMTL

In this section, we present the proposed DSMTL model. Beside the network architecture, we introduce two strategies to learn model parameters, leading to two variants (i.e., DSMTL-IL and DSMTL-JL).

6.4.1 The Architecture

As shown in Figure 6.1, the architecture of the DSMTL model can be divided into four parts: a public encoder f_S shared by all the tasks, m private encoders $\{f_t\}_{t=1}^m$ for m tasks, m gates $\{g_t\}_{t=1}^m$ for m tasks, and m private decoders $\{h_t\}_{t=1}^m$ for m tasks. For task t, its model consists of the public encoder f_S, the private encoder f_t, the gate g_t, and the private decoder h_t, where f_S and f_t are combined by g_t. Specifically, given a data sample $\mathbf{x}$, the gate g_t in task t receives two inputs: $f_S(\mathbf{x})$ and $f_t(\mathbf{x})$, and outputs $g_t(f_S(\mathbf{x}), f_t(\mathbf{x}))$,

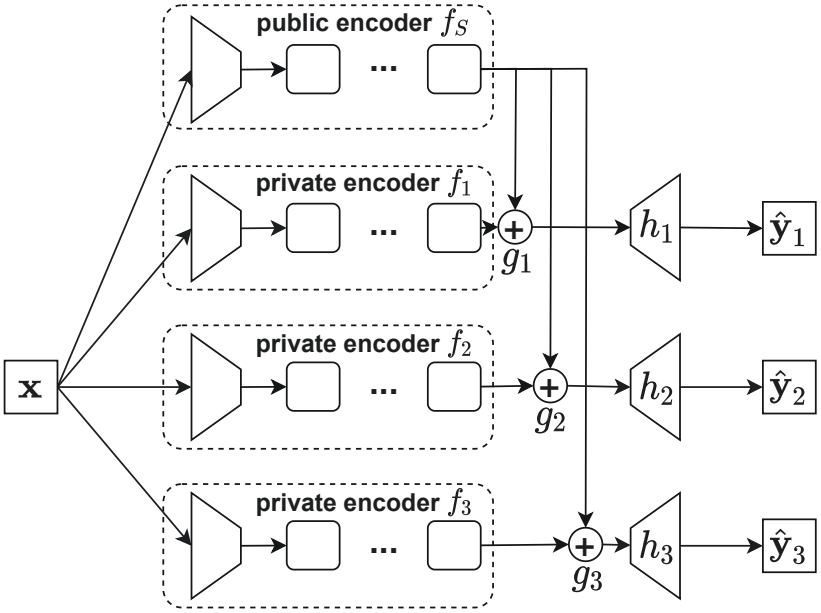


Figure 6.1: An illustration for the architecture of the DSMTL model with three tasks (i.e., $m = 3$). Here without loss of generality, we assume different tasks share the input data. For task t, an input $\mathbf{x}$ is first fed into both the public encoder f_S and private encoder f_t, then it goes through the gate g_t to obtain the combined feature representation, and finally it is through the private decoder h_t to obtain the output $\hat{\mathbf{y}}_t$.

which is fed into h_t to obtain the final prediction $h_t(g_t(f_S(\mathbf{x}), f_t(\mathbf{x})))$, which is used to define a loss for $\mathbf{x}$. Here the public encoder f_S and private encoders $\{f_t\}_{t=1}^m$ usually have the same network structure. The private decoders are designed to be task-specific as different tasks may have different types of loss functions.

Here the gate g_t is to determine the contributions of f_S and f_t. Ideally, when task t is unrelated to other tasks, g_t should choose f_t only. On another extreme where all the tasks have the same data distribution, all the tasks should use the same model and hence g_t should choose f_S only. In cases between those two extremes, g_t can combine f_S and f_t in proportion. To achieve the aforementioned effects, we use a simple convex

combination function for g_t as

$$g_t(f_S(\mathbf{x}), f_t(\mathbf{x})) = \alpha_t f_S(\mathbf{x}) + (1-\alpha_t) f_t(\mathbf{x}), \tag{6.1}$$

where $\alpha_t \in \mathscr{M} = [0,1]$ defines the weight of $f_S(\mathbf{x})$ for task t and it is a learnable parameter. When α_t equals 0, only the private encoder f_t will be used, which corresponds to the unrelated case. When α_t equals 1, only the public encoder f_S will be used, which corresponds to the case that all the tasks follow identical or similar distributions. When α_t is between 0 and 1, f_S and f_t are combined with proportions α_t and $1-\alpha_t$, respectively, and α_t can be adaptively learned to minimize the training loss on task t.

The average empirical loss of all the tasks is defined as

$$\frac{1}{mn} \sum_{t=1}^{m} \sum_{i=1}^{n} \mathscr{L}_t(\mathbf{y}_t^i, h_t(g_t(f_S(\mathbf{x}_t^i), f_t(\mathbf{x}_t^i)))), \tag{6.2}$$

where $\mathbf{x}_t^i$ denotes the ith data point in task t, $\mathbf{y}_t^i$ denotes the label of $\mathbf{x}_t^i$ in task t, without loss of generality, different tasks are assumed to have the same number of data samples, which is denoted by n, and $\mathscr{L}_t$ denotes the loss function for task t (e.g., the pixel-wise cross-entropy loss for the semantic segmentation task, the L_1 loss for the depth estimation task, and the element-wise dot product loss for the surface normal prediction task). The DSMTL model is to minimize the average empirical loss in Eq. (6.2) to learn its model parameters. In the following sections, we provide two learning strategies (e.g., individual learning and joint learning) to learn model parameters, leading to two variants of DSMTL, including DSMTL with Individual Learning (DSMTL-IL) and DSMTL with Joint Learning (DSMTL-JL).

6.4.2 DSMTL-IL

The set of all the parameters in the DSMTL model is denoted by Θ. We divide Θ into Θ_S and Θ_H, where Θ_S includes all the parameters in $\{f_t\}_{t=1}^m$ and $\{h_t\}_{t=1}^m$, and Θ_H includes the parameters in f_S and $\{g_t\}_{t=1}^m$. The individual learning strategy consists of two stages. The first stage is to optimize Θ_H by fixing each α_t to 0. Then by fixing the learned Θ_H in the first stage, the second stage is to learn Θ_H. As the single-task model for each task consists of an encoder and a decoder whose structures are identical to f_t and h_t, respectively, the first stage is equivalent to learning a single-task model for each task, and after that, the second stage can learn the shared encoder f_S and the gate $\{g_t\}_{t=1}^m$. Formally, the objective function of the DSMTL-IL model is formulated as

$$\min_{\Theta_H}\left[\min_{\Theta_S}\frac{1}{mn}\sum_{t=1}^{m}\sum_{i=1}^{n}\mathscr{L}_t(\mathbf{y}_t^i,h_t(g_t(f_S(\mathbf{x}_t^i),f_t(\mathbf{x}_t^i))))\right]. \tag{6.3}$$

The DSMTL-IL model can be proved to achieve both *empirically individual safe multi-task learning* and *probably individual safe multi-task learning* as shown in Section 6.5.2 due to its two-stage optimization process.

6.4.3 DSMTL-JL

Different from the DSMTL-IL model which optimizes two partitions of model parameters sequentially, the DSMTL-JL model adopts a joint learning strategy to learn Θ together. Formally, the objective function of the DSMTL-JL model is formulated as

$$\min_{\Theta}\frac{1}{mn}\sum_{t=1}^{m}\sum_{i=1}^{n}\mathscr{L}_t(\mathbf{y}_t^i,h_t(g_t(f_S(\mathbf{x}_t^i),f_t(\mathbf{x}_t^i)))). \tag{6.4}$$

The joint learning strategy allows the DSMTL model to learn all the model parameters, which is more flexible for deep neural networks in an end-to-end learning manner. Different from the DSMTL-IL model, the DSMTL-JL model can achieve both *empirically average safe multi-task learning* and *probably average safe multi-task learning* as proven in Section 6.5.3.

6.5 Analyses

In this section, we provide theoretical analyses to analyze the safeness of both the DSMTL-IL and DSMTL-JL methods.

6.5.1 Preliminary

With m tasks, the probability measure for the data distribution in task t is denoted by μ_t and the data in all the tasks take the form of $(\bar{\mathbf{X}}, \bar{\mathbf{Y}}) \sim \prod_{t=1}^{m}(\mu_t)^n$, where $\mathbf{X}_t = (\mathbf{x}_t^1, \ldots, \mathbf{x}_t^n)$ denotes the data in task t, $\bar{\mathbf{X}} = (\mathbf{X}_1, \ldots, \mathbf{X}_m)$, and $\bar{\mathbf{Y}}$ denotes labels for $\bar{\mathbf{X}}$. Here we consider the encoders $f_1, \ldots, f_m, f_S : \mathscr{X} \to \mathbb{R}^q$ as mapping functions chosen from a hypothesis class $\mathscr{F}$ and the decoders $h_1, \ldots, h_m$ as mapping functions chosen from a hypothesis class $\mathscr{H}$. To facilitate the analysis, we introduce following assumption.

Assumption 1. *Assume that* (i) $\mathscr{L}_t(\cdot,\cdot) \in [0,1]$ *for* $t = 1, \ldots, m$ *is* 1*-Lipschitz w.r.t the second argument;* (ii) *The hypothesis class* $\mathscr{F}$ *is uniformly bounded;* (iii) *The functions in hypothesis class* $\mathscr{H}$ *are Lipschitz continuous;* (iv) $0 \in \mathscr{F}$ *and* $h(0) = 0$ *holds for all the*

functions h in $\mathscr{H}$.[1]

To analyze the safeness of the DSMTL variants, we compare with the corresponding Single-Task Learning (STL) model, which consists of an encoder and a decoder with identical structures to f_t and h_t, respectively, for task t. We also compare with the widely-used Hard Parameter Sharing (HPS) model, which consists of a shared encoder and task-specific decoders with the network structures identical to f_S and $\{h_t\}_{t=1}^m$, respectively. Then the empirical loss of the STL model on task t is formulated as $L_t^{\text{STL}} = \frac{1}{n}\sum_{i=1}^n \mathscr{L}_t(\mathbf{y}_t^i, h_t^{\text{STL}}(f_t^{\text{STL}}(\mathbf{x}_t^i)))$ and the expected loss of the STL model on task t is formulated as $\mathscr{E}_t^{\text{STL}} = \mathbb{E}_{(\mathbf{x},\mathbf{y})\sim\mu_t}[\mathscr{L}_t(\mathbf{y}, h_t^{\text{STL}}(f_t^{\text{STL}}(\mathbf{x})))]$, where f_t^{STL} and h_t^{STL} have identical network structures to f_t and h_t in DSMTL, respectively. The average empirical loss of the STL model is computed as $L^{\text{STL}} = \frac{1}{m}\sum_{t=1}^m L_t^{\text{STL}}$, and the average expected loss of the STL model is as $\mathscr{E}^{\text{STL}} = \frac{1}{m}\sum_{t=1}^m \mathscr{E}_t^{\text{STL}}$. Similarly, the average empirical loss of the HPS model is formulated as

$$L^{\text{HPS}} = \frac{1}{mn}\sum_{t=1}^m\sum_{i=1}^n \mathscr{L}_t(\mathbf{y}_t^i, h_t^{\text{HPS}}(f_S^{\text{HPS}}(\mathbf{x}_t^i))),$$

where h_t^{HPS} and f_S^{HPS} have identical network structures to h_t and f_S in DSMTL, respectively.

The expected loss of DSMTL on task t is formulated as

$$\mathscr{E}_t = \mathbb{E}_{(\mathbf{x},\mathbf{y})\sim\mu_t}[\mathscr{L}_t(\mathbf{y}, h_t(g_t(f_S(\mathbf{x}), f_t(\mathbf{x}))].$$

Then the average expected loss of the DSMTL model is computed as $\mathscr{E} = \frac{1}{m}\sum_{t=1}^m \mathscr{E}_t$.

[1]The last assumption in Assumption 1 is not essential but it can help give simpler theoretical results as verified by the proofs in the appendix.

6.5.2 Analysis on DSMTL-IL

In DSMTL-IL, since α_t is set to zero for each task in the first stage, the objective function of the first stage is equivalent to the following problem as

$$\min_{\Theta_S} \frac{1}{mn} \sum_{t=1}^{m} \sum_{i=1}^{n} \mathscr{L}_t(\mathbf{y}_t^i, h_t(g_t^0(\emptyset, f_t(\mathbf{x}_t^i)))), \tag{6.5}$$

where g_t^0 denotes the gate of task t with α_t as 0 and $\emptyset$ denotes a null network. Thus the first stage is to train the STL model with the empirical loss L_t^{STL} for task t. As g_t is a learnable gate, after sufficient training in the second stage, the empirical loss of the DSMTL-IL model on each task is no larger than that of the first stage. Based on this observation, we have the following theorem.[2]

Theorem 6.5.1. *Let L^* be the optimal value of problem* (6.3) *and L_t^* the corresponding empirical loss for task t. Then we have $L_t^* \leq L_t^{STL}$ for all $1 \leq t \leq m$.*

Theorem 6.5.1 shows that the DSMTL-IL model can achieve *empirically individual safe multi-task learning* in Definition 6.3.3. Moreover, in the following theorem, we show that it also achieves *probably individual safe multi-task learning*.

Theorem 6.5.2. *Suppose Assumption 1 is satisfied. Let L^* be the optimal value of problem* (6.3) *and the corresponding solution is denoted by $\hat{f}_S$, $\{\hat{f}_t\}$, $\{\hat{h}_t\}$, $\{\hat{g}_t\}$. Let $\hat{\mathscr{E}}_t = \mathscr{E}_t(\hat{f}_S, \hat{f}_t, \hat{h}_t, \hat{g}_t)$. Then for $(\mathbf{X}_t, \mathbf{Y}_t) \sim \mu_t^n$, with probability at least $1-\delta$, we have*

$$\hat{\mathscr{E}}_t + \epsilon_t \leq \mathscr{E}_t^{STL} + \rho_{n,t},$$

where ϵ_t is formulated as $\epsilon_t = L_t^{STL} - L_t^*$, $\rho_{n,t}$ is defined as $\rho_{n,t} = \frac{C_1 G(\mathscr{F}'(\mathbf{X}_t))}{n} + \frac{C_2 Q}{\sqrt{n}} + \sqrt{\frac{18 \ln \frac{2}{\delta}}{n}}$, $\mathscr{F}'(\mathbf{X}_t) = \{(f(\mathbf{x}_t^i)) : f \in \mathscr{F}\}$, C_1 and C_2 are two constants, $G(\cdot)$ denotes the Gaussian

[2]The proofs for all the theorems are put in the appendix.

average [2], and Q is defined as $Q = \sup_{z \neq \tilde{z} \in \mathbb{R}^{nq}} \mathbb{E} \sup_{h \in \mathscr{H}} \frac{\langle \gamma, h(z_i) - h(\tilde{z}_i) \rangle}{\|z - \tilde{z}\|}$ *with γ as a vector of independent standard normal variables.*

Remark 5. *For many classes of interest, the Gaussian average $G(\mathscr{F}'(\mathbf{X}_t))$ is $O(\sqrt{n})$ according to [94]. For reasonable classes $\mathscr{H}$, one can find a bound on Q, which is independent of n [95]. Therefore, $\rho_{n,t}$ is $O(1/\sqrt{n})$ for $1 \leq t \leq m$ and hence $\rho_{n,t}$ satisfies $\lim_{n \to +\infty} \rho_{n,t} = 0$. Moreover, according to Theorem 6.5.1, we have $\epsilon_t \geq 0$. Thus, Theorem 6.5.2 proves that the proposed DSMTL-IL model can achieve* probably individual safe multi-task learning *in Definition 6.3.4.*

6.5.3 Analysis on DSMTL-JL

In this section, we analyze the safeness and excess risk bound of the DSMTL-JL model.

For the safeness of the DSMTL-JL model, we have the following theorems.

Theorem 6.5.3. *Let L^* be the optimal value of problem* (6.4)*. Then we have L^* is no higher than the minimun of L^{STL} and L^{HPS}, i.e., $L^* \leq \min\{L^{STL}, L^{HPS}\}$.*

Remark 6. *Theorem 6.5.3 shows that the DSMTL-JL model can achieve* empirically average safe multi-task learning *in Definition 6.3.3. Moreover, it also implies that it can achieve a lower average empirical loss compared with the corresponding HPS model. To see that, the HPS model for task t can be represented as $h_t(g_t^1(f_S(\mathbf{x}), \emptyset))$, where g_t^1 denotes the gate of task t with α_t as 1. As g_t^1 is a feasible solution for the DSMTL-JL model, it is easy to see that the empirical loss of the DSMTL-JL model after sufficient training is*

lower than that of the HPS model, which could be one reason why the DSMTL-JL model outperforms the HPS model as shown in experiments.

Theorem 6.5.4. *Suppose Assumption 1 is satisfied. Let L^* be the optimal value of problem* (6.4) *and the corresponding solution is denoted by $\hat{f}_S$, $\{\hat{f}_t\}$, $\{\hat{h}_t\}$, $\{\hat{g}_t\}$. Let $\hat{\mathscr{E}} = \mathscr{E}(\hat{f}_S, \{\hat{f}_t\}, \{\hat{h}_t\}, \{\hat{g}_t\})$. Then for $(\bar{\mathbf{X}}, \bar{\mathbf{Y}}) \sim \prod_{t=1}^{m} (\mu_t)^n$, with probability at least $1-\delta$, we have*

$$\hat{\mathscr{E}} + \epsilon \leq \mathscr{E}^{STL} + \rho_n,$$

where $\rho_n = \frac{C_1 G(\mathscr{F}(\bar{\mathbf{X}}))}{nm} + \frac{C_2 Q}{\sqrt{n}} + \sqrt{\frac{18 \ln \frac{2}{\delta}}{mn}}$, $\mathscr{F}(\bar{\mathbf{X}}) = \{(f_1(\mathbf{x}_t^i), \ldots, f_m(\mathbf{x}_t^i)) : f_t \in \mathscr{F}\}$, C_1 and C_2 are two constants, ϵ is formulated as $\epsilon = L^{STL} - L^*$, and Q is defined in Theorem 6.5.2.

Remark 7. *According to [94], the Gaussian average $G(\mathscr{F}(\bar{\mathbf{X}}))$ is $O(\sqrt{mn})$ for many classes of interest. Therefore, ρ_n is $O(\frac{1}{\sqrt{n}})$ and it satisfies $\lim_{n \to +\infty} \rho_n = 0$. According to Theorem 6.5.3, $\epsilon \geq 0$. Thus, Theorem 6.5.4 implies that the proposed DSMTL-JL model can achieve* probably average safe multi-task learning *in Definition 6.3.4.*

To analyze the excess risk bound for the DSMTL-JL model, we define the minimal expected risk as

$$\mathscr{E}^* = \min_{f_S, f_t \in \mathscr{F}, h_t \in \mathscr{H}, \alpha_t \in \mathscr{M}} \mathscr{E}.$$

Then we have the following result.

Theorem 6.5.5. *Suppose Assumption 1 is satisfied. The solution of the optimization problem in Eq.* (6.4) *is denoted by $\hat{f}_S$, $\{\hat{f}_t\}$, $\{\hat{h}_t\}$, $\{\hat{g}_t\}$. Let $\hat{\mathscr{E}} = \mathscr{E}(\hat{f}_S, \{\hat{f}_t\}, \{\hat{h}_t\}, \{\hat{g}_t\})$. Then for $(\bar{\mathbf{X}}, \bar{\mathbf{Y}}) \sim \prod_{t=1}^{m} (\mu_t)^n$, with probability at least $1-\delta$, we have*

$$\hat{\mathscr{E}} - \mathscr{E}^* \leq \frac{C_1 G(\mathscr{F}(\bar{\mathbf{X}}))}{nm} + \frac{C_2 Q}{\sqrt{n}} + \sqrt{\frac{8 \ln \frac{4}{\delta}}{mn}}, \tag{6.6}$$

where $\mathscr{F}(\bar{\mathbf{X}})$ is defined in Theorem 6.5.4, C_1 and C_2 are two constants, and Q is defined in Theorem 6.5.2.

Theorem 6.5.5 provides an upper bound on the error of this DSMTL-JL model. In the bound (6.6), the first term of the right-hand side can be regarded as the cost to estimate all the feature mappings $\{f_t\}$ and f_S, and it decreases with respect to the number of tasks. The order of this term is $O(\frac{1}{\sqrt{mn}})$. The second term of the right-hand side corresponds to the cost to estimate task-specific functions $\{g_t\}$ and $\{h_t\}$, and it is of order $O(\frac{1}{\sqrt{n}})$. The third term defines the confidence of the bound. The convergence rate of this bound is as tight as typical generalization bounds [95] for MTL.

6.6 Architecture Learning for DSMTL

A limitation of the DSMTL model is that its model size grows linearly with respect to the number of tasks, which makes its scalability not so good. To address this issue, we propose the DSMTL-AL model to not only achieve comparable or even better performance than the DSMTL-IL and DSMTL-JL models but also learn a more compact architecture via techniques in neural architecture search [77].

Inspired by the architecture of the DSMTL model introduced in Section 6.4.1, the supernet in the DSMTL-AL method has a public encoder f_S, m private encoders $\{f_t\}_{t=1}^m$, and m private decoders $\{h_t\}_{t=1}^m$. Instead of treating the public and private encoders as a whole, we divide them into modules, which could be a fully connected layer or a sophisticated ResNet block/layer, depending on the MTL problem under investigation.

Without loss of generality, we assume that both the public and private encoders have the same number of modules, i.e., $(P-1)$.

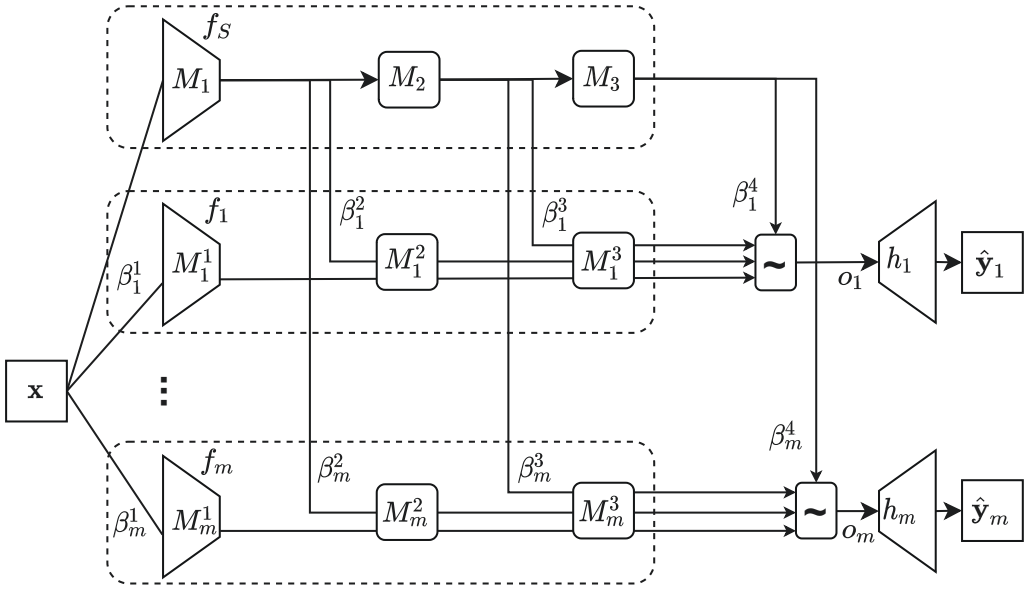


Figure 6.2: An illustration for the architecture learning process in the DSMTL-AL model, where without loss of generality, different tasks are assumed to share the same input data. For simplicity, we assume that there are three modules (i.e., $P=4$) in each encoder. "$\sim$" represents the convex combination. In the retraining process, only the branch with the largest architecture parameter whose index is denoted by p_t^* is preserved and the first (p_t^*-1) private modules will be removed.

As illustrated in Figure 6.2, the DSMTL-AL method is to learn a branching architecture to combine public modules in the public encoder and private modules in the corresponding private encoder for each task to reduce the model size. The search space for the architecture in the DSMTL-AL method consists of m architecture parameters $\{\boldsymbol{\beta}_t\}_{t=1}^m$ to decide branch positions for m tasks, where $\boldsymbol{\beta}_t = (\beta_t^1, \ldots, \beta_t^P)$. As a binary parameter, $\beta_t^p \in \{0,1\}$ indicates whether task t branches at the branch position p from f_S to f_t. Specifically, when β_t^p equals 1, there will be a branch to feed the output of the $(p-1)$-th module in f_S to the p-th module in f_t to form a combined encoder for task t. Moreover,

the sum of entries in $\boldsymbol{\beta}_t$ should be 1 for any t, indicating that there is only one branch position for each task. In this sense, only part of the private/public modules in all the encoders will be used for each task. Hence the model size is smaller than the entire supernet, which is used in the DSMTL-IL and DSMTL-JL models.

The search space in the DSMTL-AL method includes both STL and HPS architectures as two extremes. When all the tasks are unrelated to each other, the architecture in the DSMTL-AL method could become the STL architecture by choosing $\{f_t\}$ for each task (i.e., $\beta_t^1 = 1$ for $t = 1,\ldots,m$). For highly related or even identical tasks, the architecture of the DSMTL-AL method could become the HPS architecture by choosing f_S only (i.e., $\beta_t^P = 1$ for $t = 1,\ldots,m$).

The DSMTL-AL method is to find the best branching architecture in the search space for all the m tasks by learning architecture parameters $\{\boldsymbol{\beta}_t\}_{t=1}^m$. If task t branches at the branch position p of f_S to connect to the corresponding next module in f_t, β_t^p is set to 1 and all the β_t^i's ($i \neq p$) are set to 0. In this case, the output of the combined encoder for task t consisting of the first $(p-1)$ public modules in f_S and the last $(P-p)$ private modules in f_t is $f_t(f_S(\mathbf{x}, p-1), p, P-1)$, where $f_S(\cdot, p)$ denotes the output of the p-th module in f_S and $f_t(\cdot, p, q)$ denotes the output of the q-th module in f_t starting from the p-th module. Here $f_S(\mathbf{x}, 0)$ is defined to be $\mathbf{x}$, corresponding to the input to the first module, and $f_t(\mathbf{x}, P, P-1)$ is defined as $\mathbf{x}$. Since $\{\boldsymbol{\beta}_t\}$ are binary variables, this discrete nature makes stochastic gradient descent methods incapable of learning them. Here we relax $\{\boldsymbol{\beta}_t\}$ to be continuous and define them as the probability of branching at each branch position. Specifically, the output of the combined encoder for task t is formulated

as

$$\mathbf{o}_t(\mathbf{x}, \boldsymbol{\beta}_t) = \sum_{p=1}^{P} \beta_t^p f_t(f_S(\mathbf{x}, p-1), p, P-1),$$

where $\boldsymbol{\beta}_t$ is in the $(P-1)$-dimensional simplex set denoted by $\mathscr{S}_P$, satisfying that $\beta_t^p \geq 0$ and $\sum_{p=1}^{P} \beta_t^p = 1$. Here $\mathbf{o}_t(\mathbf{x}, \boldsymbol{\beta}_t)$ is a convex combination of outputs of all possible branching architectures weighted by probabilities based on $\boldsymbol{\beta}_t$. Then $\mathbf{o}_t(\mathbf{x}, \boldsymbol{\alpha}_t)$ is fed into the decoder h_t to generate the prediction and hence the empirical loss for task t is formulated as

$$L_t(\Theta_t, \boldsymbol{\beta}_t) = \frac{1}{n} \sum_{i=1}^{n} \mathscr{L}_t(\mathbf{y}_t^i, h_t(\mathbf{o}_t(\mathbf{x}_t^i, \boldsymbol{\beta}_t)))$$

where Θ_t includes all the parameters in f_S, f_t, and h_t. Then the weighted empirical loss over m tasks is formulated as

$$L(\Theta, \boldsymbol{\beta}, \boldsymbol{w}) = \sum_{t=1}^{m} w_t L_t(\Theta_t, \boldsymbol{\beta}_t), \tag{6.7}$$

where $\Theta = \{\Theta_t\}_{t=1}^m$, $\boldsymbol{w} = (w_1, \ldots, w_m)$ is in $\mathscr{S}_m$ satisfying $w_t \geq 0$ and $\sum_{t=1}^m w_t = 1$, and $\boldsymbol{\beta} = (\boldsymbol{\beta}_1, \ldots, \boldsymbol{\beta}_m)$. $\boldsymbol{w}$ specifies the weighting among all the tasks. Setting them to $\frac{1}{m}$ as in the DSMTL-IL and DSMTL-JL models may lead to suboptimal performance and hence we aim to learn them directly.

Here Θ is viewed as model parameters, while $\boldsymbol{\beta}$ and $\boldsymbol{w}$ are hyperparameters. To learn all of them, we adopt a bi-level formulation as

$$\begin{aligned} &\min_{\boldsymbol{\beta} \in \mathscr{S}_P, \boldsymbol{w} \in \mathscr{S}_m} L_{val}(\hat{\Theta}, \boldsymbol{\beta}, \boldsymbol{w}) \\ &\text{s.t. } \hat{\Theta} = \arg\min_{\Theta} L_{tr}(\Theta, \boldsymbol{\beta}, \boldsymbol{w}), \end{aligned} \tag{6.8}$$

where the entire training dataset is divided into a training set and a validation set, $\mathscr{L}_{tr}(\cdot, \cdot, \cdot)$ denotes the weighted empirical loss defined in Eq. (6.7) on the training set, and

$\mathscr{L}_{val}(\cdot,\cdot,\cdot)$ denotes the weighted empirical loss defined in Eq. (6.7) on the validation set. Here the constraints on $\boldsymbol{\beta}$ and $\mathbf{w}$ can be alleviated via the reparameterization based on the softmax function. We adopt the gradient-based hyperparameter optimization algorithm in [35, 77] to solve problem (6.8) with the first-order approximation. After solving the problem (6.8), we can learn architecture for task t by determining the branch position as

$$\beta_t^* = \arg\max_p(\{\beta_t^p\}_{p=1}^P).$$

With the learned architecture, we can use the entire training dataset to retrain the model parameters Θ. Inspired by the gating mechanism in the DSMTL-IL and DSMTL-JL models, the final encoder for task t consists of the shared encoder and the combined encoder determined by β_t^*. One reason for that is that the shared encoder f_S could be fully used to improve the performance with little increase or even no increase in the model size, which is due to that all the modules in f_S will usually be chosen by at least one task during the architecture learning process. Formally, the final encoder for task t is formulated as

$$\hat{g}_t(\mathbf{x},\beta_t^*) = \alpha_t f_S(\mathbf{x}) + (1-\alpha_t) f_t(f_S(\mathbf{x},\beta_t^*-1),\beta_t^*,P-1), \tag{6.9}$$

where with abuse of notations, $\alpha_t \in \mathscr{M} = [0,1]$ is a learnable parameter to measure the weight of $f_S(\mathbf{x})$ and acts similarly to α_t in the DSMTL-IL and DSMTL-JL models. Mathematically, the objective function of the retraining process is formulated as

$$\min_{\Theta,\boldsymbol{\alpha}\in\mathscr{M}} \sum_{t=1}^m \frac{w_t}{n} \sum_{i=1}^n \mathscr{L}_t(\mathbf{y}_t^i, h_t(\hat{g}_t(\mathbf{x}_t^i,\beta_t^*))), \tag{6.10}$$

where Θ denotes all the model parameters, $\boldsymbol{\alpha} = (\alpha, \ldots, \alpha_m)$, and w_t is the loss weight learned from problem (6.8) for task t. After the retraining process, we can use the learned model parameters to make prediction for each task.

Though the DSMTL-AL model cannot be theoretically proved to achieve some version of *safe multi-task learning*, as shown in the next section, empirically it not only achieves good performance but also learns compact architectures.

6.7 Experiments

In this section, we evaluate the proposed models.

6.7.1 Datasets and Evaluation Metrics

Experiments are conducted on four MTL CV datasets, including CityScapes [21], NYUv2 [116], PASCAL-Context [99], and Taskonomy [149].

The CityScapes dataset consists of high resolution outside street-view images. By following [80], we evaluate the performance on the 7-class semantic segmentation and depth estimation tasks. The NYUv2 dataset consists of RGB-D indoor scene images from three learning tasks: 13-class semantic segmentation, depth estimation, and surface normal prediction. The PASCAL-Context dataset is an annotation extension of the PASCAL VOC 2010 challenge with four learning tasks: 21-class semantic segmentation, 7-class human parts segmentation, saliency estimation, and surface normal estimation, where the last two tasks are generated by [93]. The Taskonomy dataset contains indoor

images. By following [118], we sample five learning tasks, including 17-class semantic segmentation, depth estimation, keypoint detection, edge detection, and surface normal prediction.

The semantic segmentation task on the PASCAL-Context dataset is evaluated by the mean Intersection over Union (mIoU) by following [93]. On the other three CV datasets, this task is additionally evaluated in terms of the Pixel Error (abbreviated as 'Pix Err') by following [124]. For the depth estimation task, the absolute error (abbreviated as 'Abs Err') and relative error (abbreviated as 'Rel Err') are used as the evaluation metrics. For the surface normal prediction task, the mean and median angle distances between the prediction and ground truth of all pixels are used as measures. For this task, the percentage of pixels, whose prediction is within the angles of 11.25°, 22.5°, and 30° to the ground truth, is used as another measure. For the keypoint detection and edge detection tasks, the absolute error (abbreviated as 'Abs Err') is used as the evaluation metric. For the human parts segmentation task, the mIoU is used as the measure. For the saliency estimation task, the mIoU and max F-measure (maxF) are adopted as the evaluation metrics.

As introduced above, for each task, we use one or more evaluation metrics to thoroughly evaluate the performance. To better show the comparison between each method and STL, we compute the relative performance of each method over STL in terms of the jth evaluation metric on task t as $\Delta_{t,j} = (-1)^{p_{t,j}}(\mathrm{M}_{t,j} - \mathrm{STL}_{t,j})$, where for a method M, $\mathrm{M}_{t,j}$ denotes its performance in terms of the jth evaluation metric for task t, $\mathrm{STL}_{t,j}$ is defined similarly, $p_{t,j}$ equals 1 if a lower value represents better performance in terms

of the jth metric in task t and 0 otherwise. So positive relative performance indicates better performance than STL. The overall relative improvement of a method M over STL is defined as $\Delta_I = \frac{1}{m}\sum_{t=1}^{m}\frac{1}{m_t}\sum_{j=1}^{m_t}\frac{\Delta_{t,j}}{\mathrm{STL}_{t,j}}$, where m_t denotes the number of evaluation metrics in task t.

To empirically measure the safeness of each model, we define the safeness coefficient η for a model as the proportion of tasks on which this model empirically performs no worse than the STL model. Formally, η is formulated as $\eta = \frac{1}{m}\sum_{t=1}^{m}\frac{1}{m_t}\sum_{j=1}^{m_t}\delta(\Delta_{t,j}) \times 100$, where $\delta(x)$ is the delta function that outputs 0 when $x < 0$ and otherwise 1. Obviously, η, whose maximum is 100, is expected to be as large as possible.

Table 6.1: Performance of various models on the CityScapes validation dataset. ↑ (↓) indicates the higher (lower) the result, the better the performance. The green color indicates that the corresponding method performs better than the STL method and the red color indicates oppositely. The number of parameters (abbreviated as Parms.) is calculated in MB.

Method	Segmentation		Depth		Δ_I ↑	η ↑	Parms. (M)↓
	mIoU ↑	Pix Err ↓	Abs Err ↓	Rel Err↓			
STL	67.48	9.00	0.0139	46.2507	0	-	79.27
HPS	−0.08	−0.08	−0.0003	+0.8245	−0.0035	25	55.76
Cross-stitch	+0.53	+0.29	+0.0004	+1.8261	+0.0271	100	79.26
MTAN	+1.49	+0.59	+0.0003	+2.4999	+0.0408	100	72.04
NDDR-CNN	+0.54	+0.25	+0.0002	+1.3845	+0.0200	100	101.58
AFA	+1.44	+0.52	−0.0019	−0.9136	−0.0193	50	87.09
RotoGrad	+0.69	+0.37	+0.0004	+1.2729	+0.0274	100	57.86
MaxRoam	−0.76	+0.33	−0.0003	+3.9607	+0.0224	50	55.76
MTL-NAS	−2.13	−0.80	−0.0005	−1.6786	−0.0482	0	87.26
BMTAS	+2.09	+0.81	+0.0017	+1.9947	+0.0723	100	79.04
TSN	+2.27	+0.85	+0.0011	+0.4047	+0.0544	100	50.58
LTB	+2.24	+0.82	+0.0014	−0.4489	+0.0539	75	70.73
DSMTL-IL	+2.94	+1.13	+0.0015	+0.4879	+0.0715	100	102.78
DSMTL-JL	+1.50	+0.62	+0.0005	+3.3886	+0.0501	100	102.78
DSMTL-AL	+3.07	+1.18	+0.0017	+4.3300	+0.0988	100	79.04

Table 6.2: Performance of various models on the NYUv2 validation dataset. ↑ (↓) indicates the higher (lower) the result, the better the performance. The green color indicates that the corresponding method performs better than the STL method and the red color indicates oppositely. The number of parameters (abbreviated as Parms.) is calculated in MB.

Method	Segmentation		Depth		Surface Normal					Δ_I ↑	η	Parms. (M)↓
	mIoU ↑	Pix Err ↓	Abs Err ↓	Rel Err↓	Angle Distance ↓		Within t° ↑					
					Mean	Median	11.25	22.5	30			
STL	53.11	4.80	0.3957	0.1632	22.26	15.49	38.61	64.43	74.69	0	-	118.91
HPS	+1.37	+0.62	+0.0118	+0.0084	−1.24	−1.57	−3.30	−3.33	−2.55	+0.0001	67	71.89
Cross-stitch	+0.35	+0.29	+0.0153	+0.0077	−0.75	−0.84	−1.60	−2.01	−1.68	+0.0051	67	118.89
MTAN	+1.63	+0.58	+0.0161	+0.0083	−0.71	−0.81	−1.70	−1.80	−1.38	+0.0127	67	92.35
NDDR-CNN	+0.73	+0.03	+0.0086	+0.0072	−0.34	−0.58	−0.94	−1.00	−0.77	+0.0066	67	169.10
AFA	−1.57	−1.29	−0.0073	−0.0060	−1.97	−1.91	−3.54	−4.21	−3.73	−0.0507	0	136.88
RotoGrad	+0.98	+0.05	+0.0157	+0.0062	−0.79	−1.01	−2.63	−2.56	−1.77	+0.0009	67	75.03
MaxRoam	+1.42	+0.41	+0.0078	−0.0068	+0.36	+0.51	−2.03	−1.55	−2.46	−0.0005	63	71.89
MTL-NAS	+0.81	+0.01	+0.0110	+0.0102	−0.15	−0.52	−0.46	−0.22	−2.11	+0.0121	67	183.40
BMTAS	+0.73	+0.22	+0.0067	+0.0017	+0.01	−0.03	−0.00	−0.08	−0.00	+0.0082	73	116.04
TSN	−0.87	−0.75	−0.0221	−0.0067	+0.50	+0.30	+1.35	+1.22	+0.90	−0.0167	33	50.58
LTB	+0.31	+0.23	+0.0052	+0.0096	+0.02	−0.22	−0.86	−0.33	+0.04	+0.0119	80	86.85
DSMTL-IL	+0.60	+0.30	+0.0007	+0.0005	+0.26	+0.06	+0.14	+0.33	+0.41	+0.0067	100	142.41
DSMTL-JL	+0.75	+0.36	+0.0114	+0.0051	+0.48	+0.47	+1.27	+1.00	+0.73	+0.0221	100	142.41
DSMTL-AL	+1.25	+0.71	+0.0144	+0.0070	+0.45	+0.40	+0.83	+0.99	+0.74	+0.0281	100	93.96

Table 6.3: Performance of various models on the PASCAL-Context validation dataset. ↑ (↓) indicates the higher (lower) the result, the better the performance. The green color indicates that the corresponding method performs better than the STL method and the red color indicates oppositely. The number of parameters (abbreviated as Parms.) is calculated in MB.

Method	Segmentation	Human Parts	Saliency		Surface Normal					Δ_I ↑	η ↑	Parms. (M)↓
	mIoU↑	mIoU↑	mIoU↑	maxF↑	Angle Distance ↓		Within t° ↑					
					Mean	Median	11.25	22.5	30			
STL	65.14	58.58	65.02	77.47	15.94	24.87	48.42	80.79	90.03	0	-	63.60
HPS	−0.37	−0.67	−0.92	−0.51	−1.73	−1.29	−6.43	−4.86	−3.02	−0.0262	0	30.07
Cross-stitch	−0.17	+0.05	−0.56	−0.40	−0.62	−0.45	−2.36	−2.68	−1.04	−0.0096	25	79.46
MTAN	−0.58	+0.50	−0.45	−0.23	−1.20	−0.89	−4.58	−3.31	−2.04	−0.0147	25	36.61
NDDR-CNN	+0.14	+0.60	+0.07	+0.00	−0.37	−0.24	−1.50	−0.94	−0.59	−0.0008	75	69.25
AFA	+2.12	+2.11	−1.95	−3.96	−1.63	−1.28	−5.68	−4.42	−2.88	−0.0108	50	199.1
RotoGrad	−1.57	−0.23	+0.34	+0.35	+0.11	+0.05	+0.14	+0.13	+0.07	−0.0051	50	33.22
MaxRoam	−1.33	+0.71	−0.99	−3.42	+0.16	+0.10	+0.35	+0.23	+0.15	−0.0082	50	30.07
MTL-NAS	−0.68	+0.18	−0.67	−0.20	+0.68	+0.47	+2.56	+1.58	+0.87	+0.0037	50	41.24
BMTAS	−0.14	+0.39	−0.36	−0.15	−0.06	−0.06	−0.34	−0.29	−0.10	-0.0006	12	45.28
TSN	+1.63	−0.67	+0.08	−0.51	−0.62	−0.29	−5.43	−3.86	−2.02	-0.0089	25	26.85
LTB	−0.65	+1.55	−0.93	−3.59	+0.58	+0.39	+2.03	+1.36	+0.74	+0.0025	75	62.60
DSMTL-IL	+0.01	+1.05	+0.20	+0.13	+0.26	+0.22	+1.08	+0.74	+0.39	+0.0082	100	74.78
DSMTL-JL	+0.18	+1.87	+0.44	+0.47	+0.41	+0.27	+1.21	+0.86	+0.50	+0.0142	100	74.78
DSMTL-AL	+0.82	+1.28	+0.37	+0.31	+0.04	+0.02	+0.01	+0.01	+0.09	+0.0106	100	63.15

Table 6.4: Performance of various models on the Taskonomy validation dataset. ↑ (↓) indicates the higher (lower) the result, the better the performance. The green color indicates that the corresponding method performs better than the STL method and the red color indicates oppositely. The number of parameters (abbreviated as Parms.) is calculated in MB.

Method	Segmentation		Depth		Keypoints	Edges	Surface Normal					$\Delta_I \uparrow$	$\eta \uparrow$	Parms. (M)↓
	mIoU ↑	Pix Err ↓	Abs Err ↓	Rel Err↓	Abs Err ↓	Abs Err ↓	Angle Distance ↓		Within t° ↑					
							Mean	Median	11.25	22.5	30			
STL	65.42	2.37	0.0072	0.0117	0.1103	0.1349	10.39	4.19	73.67	86.21	90.52	0	-	79.50
HPS	+0.33	+0.05	−0.0011	−0.0018	+0.0015	−0.0004	−0.69	−0.45	−1.94	−1.09	−0.79	−0.0348	40	34.79
Cross-stitch	+0.87	+0.67	+0.0005	+0.0009	+0.0100	+0.0019	+1.50	+0.17	+0.42	+0.38	+0.30	+0.0603	100	79.46
MTAN	+0.34	+0.69	−0.0011	−0.0019	−0.0248	−0.0111	+0.99	−0.18	+2.30	+2.89	+2.41	+0.0240	36	36.61
NDDR-CNN	+0.64	+0.69	+0.0002	−0.0028	+0.0133	+0.0043	+1.86	+0.47	+0.48	+0.42	+0.33	+0.0593	90	88.32
AFA	+0.84	+0.57	−0.0016	−0.0027	+0.0465	+0.0524	+1.08	−0.17	+0.30	+0.33	+0.27	+0.0476	76	242.1
RotoGrad	+0.01	−0.03	+0.0012	+0.0019	+0.0028	+0.0003	+0.13	+0.27	+0.20	+0.05	+0.02	+0.0213	90	37.94
MaxRoam	−0.61	−0.10	+0.0013	+0.0021	+0.0028	+0.0015	+0.06	+0.22	+0.06	−0.04	−0.05	+0.0164	72	34.79
MTL-NAS	+0.01	−0.04	+0.0014	+0.0022	+0.0027	+0.0014	+0.16	+0.30	+0.35	+0.13	+0.04	+0.0253	90	45.97
BMTAS	−0.53	−0.10	+0.0019	+0.0031	+0.0048	+0.0005	+0.03	+0.17	+0.01	−0.08	−0.10	+0.0217	72	66.27
TSN	−0.07	−0.02	+0.0012	+0.0019	+0.0020	+0.0001	+0.03	+0.24	−0.12	−0.20	−0.19	+0.0184	68	26.85
LTB	+0.30	+0.01	+0.0015	+0.0024	+0.0038	+0.0001	+0.46	+0.69	+0.80	+0.23	+0.13	+0.0426	100	43.19
DSMTL-IL	+0.67	+0.01	+0.0034	+0.0055	+0.0056	+0.0014	+0.97	+1.06	+2.00	+0.95	+0.65	+0.0844	100	90.68
DSMTL-JL	+0.89	+0.03	+0.0035	+0.0057	+0.0056	+0.0014	+0.98	+1.08	+2.00	+0.95	+0.66	+0.0872	100	90.68
DSMTL-AL	+0.45	+0.02	+0.0012	+0.0019	+0.0038	+0.0013	+0.59	+0.83	+1.11	+0.39	+0.22	+0.0455	100	54.36

6.7.2 Experimental Setup

The baseline methods in comparison include the Single-Task Learning (STL) that trains each task separately, popular MTL architectures including the HPS model that adopts the multi-head hard sharing architecture, Cross-stitch [97], MTAN [80], NDDR-CNN [39], RotoGrad [56], and AFA [23], and popular architecture learning methods for MTL such as MaxRoam [103], TSN [123], MTL-NAS [38], BMTAS [8], and LTB [45]. For fair comparison, we use the same backbone for all the models. Similar to [80], we use the Deeplab-ResNet [15] with atrous convolutions as encoders and the ASPP architecture [15] as decoders. We adopt the ResNet-50 pretrained on ImageNet for the CityScapes and NYUv2 datasets to implement the the Deeplab-ResNet, and use the pretrained ResNet-18 on the larger PASCAL-Context and Taskonomy datasets for training efficiency. We use the cross-entropy loss for the semantic segmentation, human parts segmentation and saliency estimation tasks, the cosine similarity loss for the surface normal prediction

task, and the L_1 loss for other tasks. For the DSMTL-AL method, a module is defined as a layer in the Deeplab-ResNet model and the branch position is set before and after each layer. Therefore, we have five layers (including conv1) and have six branch positions. For optimization, we use the Adam method [59] with the learning rate as 10^{-4}. All the experiments are conducted on Tesla V100 GPUs.

6.7.3 Experimental Results

Tables 6.1-6.4 show the performance of all the models in comparison on different datasets. On the CityScapes dataset, the proposed DSMTL-IL, DSMTL-JL, DSMTL-AL and some baseline methods (i.e., Cross-stitch, MTAN, RotoGrad, BMTAS, TSN, and NDDR-CNN) perform no worse than the STL model on each metric and hence under such setting, they achieve *safe multi-task learning* (i.e., $\eta = 100$). In addition, the proposed DSMTL-AL model achieves the best Δ_I, which demonstrates its effectiveness. On the NYUv2 and PASCAL-Context datasets, none of the baselines can achieve *safe multi-task learning*, while the proposed methods (i.e., DSMTL-IL, DSMTL-JL, and DSMTL-AL) can achieve that, which again shows the effectiveness of the proposed methods. On the Taskonomy dataset, only the cross-stitch network, LTB and the proposed methods can achieve *safe multi-task learning* and among them, the proposed DSMTL-JL method performs the best in terms of Δ_I. According to results shown in Table 6.4, we can see that the AFA method achieves the best performance on the keypoint detection and edge detection tasks for the Taskonomy dataset, but it does not achieve *safe multi-task learning*, which is the focus of the proposed methods. Moreover, the number of parameters in the AFA model is

2.66 times over that of the DSMTL-IL and DSMTL-JL models, which may explain the improvement of the AFA model on some tasks.

For the proposed three methods, all of them achieve safeness on the four datasets, which demonstrates their effectiveness. The proposed DSMTL-JL method performs better than the DSMTL-IL method on the NYUv2, PASCAL-Context, and Taskonomy datasets in terms of Δ_I. One reason is that the DSMTL-JL model can learn a better model by optimizing all the model parameters together, while the DSMTL-IL model adopts a two-stage optimization strategy. Compared with those two models, the DSMTL-AL method can achieve a better trade-off between the performance and the model size as it performs comparable or even better than the better one in DSMTL-IL and DSMTL-JL and its model size is much smaller than those in DSMTL-IL and DSMTL-JL, which matches the design goal of the DSMTL-AL method.

Table 6.5: $\{\alpha_t\}$ learned in the DSMTL models as well as branch points and $\{w_t\}$ learned by the DSMTL-AL method on four CV datasets. 'SS' stands for the semantic segmentation task, 'DE' denotes the depth estimation task, 'SNP' is for the surface normal prediction task, 'HPS' corresponds to the human parts segmentation task, 'SE' stands for the saliency estimation task, 'KD' stands for the keypoint detection task, and 'ED' denotes the edge detection task.

Method	CityScapes		NYUv2			PASCAL-Context				Taskonomy				
	SS	DE	SS	DE	SNP	SS	HPS	SE	SNP	SE	DE	KD	ED	SNP
DSMTL-IL	0.5539	0.4470	0.5624	0.5083	0.4745	0.3823	0.3899	0.4426	0.3100	0.4256	0.3162	0.3143	0.3768	0.3056
DSMTL-JL	0.5002	0.4960	0.4383	0.5188	0.1997	0.4739	0.5529	0.3701	0.2304	0.4886	0.4565	0.4504	0.4578	0.2584
DSMTL-AL	–	0.3675	–	–	0.3931	–	0.5651	0.5545	0.3952	–	–	0.3565	–	0.3438
- branch point	$p=6$	$p=4$	$p=6$	$p=6$	$p=4$	$p=6$	$p=2$	$p=1$	$p=4$	$p=6$	$p=6$	$p=4$	$p=6$	$p=1$
- learned w_t	0.3274	0.6726	0.0283	0.0568	0.9149	0.0574	0.0649	0.8032	0.0745	0.0012	0.0033	0.4932	0.4932	0.0091

6.7.4 Analysis on the Position of Gate

In this section, we study how the position of the gate affects the performance of the proposed models and we use the DSMTL-JL model as an example. As there are $P-1$

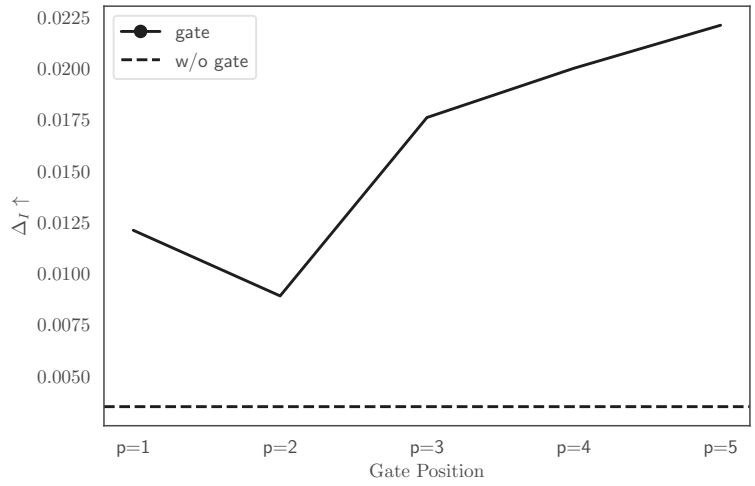


Figure 6.3: The performance of the DSMTL-JL model on the NYUv2 dataset when varying the position of the gates, where p represents the position of the gate.

possible positions for each gate by excluding the position before the first layer, we try each of them to see which one is the best in terms of the performance. To avoid the exponential complexity, we assume that gate positions of different tasks are the same.

Specifically, we put a gate $g_t(\cdot, p)$ after the p-th module of the private encoder in task t and the entire encoder for task t is formulated as

$$g_t(\mathbf{x}, p) = f_t(\alpha_t^p f_S(\mathbf{x}, p) + (1 - \alpha_t^p) f_t(\mathbf{x}, 1, p), p + 1, P - 1),$$

where as defined in Section 6.6, $f_S(\cdot, p)$ denotes the output of the p-th module in f_S and $f_t(\cdot, p, q)$ denotes the output of the q-th module in f_t starting from its p-th module.

According to Figure 6.3, when changing the gate position from $p = 2$ to $p = 5$, the performance of the DMTL-JL model becomes better, and $p = 5$ gives the best performance, which justifies the choice of the gate position in the DSMTL-IL and DSMTL-JL models and also inspires the design of the final encoder in the DSMTL-AL method as defined in Eq. (6.9).

Table 6.6: Ablation study of the DSMTL models on the NYUv2 dataset.

Method	$\Delta_I \uparrow$	$\eta \uparrow$
DSMTL-IL	+0.0067	100
-w/o learnable gate	+0.0034	77
DSMTL-JL	+0.0221	100
-w/o learnable gate	+0.0035	83
DSMTL-AL	+0.0281	100
-w/o learnable gate	+0.0220	100
-w/o learnable task weighting	+0.0166	80

6.7.5 Analysis on Learned Task Relevance

We show the learned $\{\alpha_t\}$ of the proposed methods in Table 6.5. According to the results, we can see that some α_t's are closed to 0.5, which implies that in those cases, the public encoder and the private encoder are both important to the corresponding tasks. Thus, only using the public encoder (i.e., HPS) and only using the private encoder (i.e., STL) cannot achieve good performance, while the proposed models can take the advantages of these two methods to achieve better performance in most cases. Moreover, some of the learned α_t's have relatively small values (i.e., values smaller than 0.3), which are shown in box. These small values indicate that for the surface normal prediction task on the NYUv2, PASCAL-Context, and Taskonomy datasets, the public encoder is relatively unimportant, and this is consistent with the architecture learned by the DSMTL-AL method, where the surface normal prediction task branches out at the first several public modules and switches to the private modules. This may imply that the surface normal prediction task is not strongly related to other tasks on these datasets, which aligns with the task relationship founded in [123]. On the other hand, this observation may explain why HPS performs much worse than STL and why the proposed methods have

good performance on those datasets (refer to Tables 6.2-6.4). For the DSMTL-AL method, we can see that in each dataset, at least one task choose to use the entire shared encoder, which corresponds to the choice of the branch point $p = 6$ and hence indicates that there is no need to learn the corresponding α_t defined in Eq. (6.9), and hence including the public encoder in the design of the final encoder defined in Eq. (6.9) for the DSMTL-AL method will not increase the model size.

As the loss scales and converge speed of different tasks vary a lot, using identical loss weighting (i.e., $w_t = \frac{1}{m}$ for $t = 1, \ldots, m$) may lead to suboptimal performance. The learned w_t's in DSMTL-AL at the bottom of Table 6.5 show that loss weights are uneven. For example, the surface normal prediction task has a larger loss weight than the other two tasks in the NYUv2 dataset, while this task has a smaller loss weight than some other tasks in the PASCAL-Context and Taskonomy datasets, which indicates that the proposed DSMTL-AL method could learn adaptive loss weighting strategies in different datasets.

6.7.6 Ablation Study

In Table 6.6, we provide the ablation study for the proposed DSMTL models. For "w/o learnable gate", we replace the learnable gates in Eq. (6.1) or (6.9) with simply compute the average of outputs of both public and private encoders. Compared with the original DSMTL methods, the performance of those variants degrades in terms of Δ_I, which verifies the usefulness of the learnable gates. The safeness coefficient decreases in both DSMTL-IL and DSMTL-JL cases, which indicates that learnable gates are a key

ingredient for the DSMTL-IL and DSMTL-JL models to achieve the safeness. The safeness coefficient still keeps as 100 in the variant of the DSMTL-AL method, which implies that the DSMTL-AL method can learn a reliable architecture to achieve the safeness.

For "w/o learning task weighting", we replace the learned task weights in the DSMTL-AL method with identical loss weights during the retraining process. This variant without the learnable task weighting has inferior performance to the DSMTL-AL method, which indicates that learning task weighting in the DSMTL-AL method can not only reduce tedious costs to tune loss weights but also improve the performance of the DSMTL-AL method.

6.7.7 Combination and Comparison with Loss Weighting Strategies

The loss weighting scheme adopted in the DSMTL-IL and DSMTL-JL methods is the commonly used Equally Weighting (EW) strategy (i.e., all the loss weights are equal to $\frac{1}{m}$ in problems (6.3) and (6.4)). The loss weighting scheme adopted in the DSMTL-AL method is a Learnable Weighting (LW) strategy as in Eq. (6.7). In this section, we show that the proposed DSMTL models could be combined with some loss weighting methods in MTL, including Uncertainty Weights (UW) [58], Dynamic Weight Average (DWA) [80], Geometric Loss Strategy (GLS) [20], PCGrad [147], and CAGrad [75].

According to experimental results shown in Table 6.7, the combination of the DSMTL-IL method and other methods than the EW strategy has inferior performance and a

Table 6.7: Combining DSMTL models with various loss weighting strategies on the NYUv2 validation dataset.

Architecture	Methods	Train Speedup ↑	Δ_I ↑	η ↑
STL	-	1.0x	0	100
HPS	EW	1.69x	+0.0001	67
	UW	1.66x	-0.0040	67
	DWA	1.68x	+0.0029	67
	GLS	1.68x	**+0.0158**	67
	PCGrad	0.78x	+0.0037	67
	CAGrad	0.61x	+0.0023	67
DSMTL-IL	EW	1.18x	**+0.0067**	100
	UW	1.15x	+0.0035	53
	DWA	1.17x	+0.0038	60
	GLS	1.16x	+0.0041	67
	PCGrad	0.43x	-0.0397	0
	CAGrad	0.36x	-0.0399	0
DSMTL-JL	EW	0.89x	+0.0221	100
	UW	0.83x	+0.0134	100
	DWA	0.84x	+0.0191	100
	GLS	0.88x	**+0.0269**	100
	PCGrad	0.34x	+0.0164	100
	CAGrad	0.30x	+0.0237	100
DSMTL-AL	LW	1.18x	+0.0281	100
	EW	1.18x	+0.0166	100
	UW	1.14x	+0.0163	100
	DWA	1.16x	+0.0204	93
	GLS	1.17x	+0.0203	80
	PCGrad	0.42x	**+0.0299**	100
	CAGrad	0.38x	+0.0243	100

lower safeness coefficient η, which verifies the usefulness of the EW strategy adopted in the DSMTL-IL method. Differently, the performance of the DSMTL-JL method could be improved in terms of Δ_I when combining with some loss weighting strategies (i.e., GLS and CAGrad), and all the combinations have the largest safeness coefficients. One reason for the aforementioned difference between the DSMTL-IL and DSMTL-JL methods is that the DSMTL-IL method fixes all parameters of the private encoders and decoders

in the second stage of the training process, making the corresponding parameters not fully updated and therefore leading to the inferior performance. For the DSMTL-AL method, replacing LW with PCGrad can further improve the performance while other loss weighting strategies degrade the performance.

In terms of computational cost or the training speedup over the STL, the PCGrad and CAGrad methods have large computational overhead since they need to project huge-dimensional gradients of different tasks on each training step. The UW, DWA and GLS methods have relatively small computational overhead but with limited performance improvement. The LW strategy in DSMTL-AL has no computational overhead during the retraining process and have competitive performance compared with various loss weighting strategy, which demonstrate the effectiveness of the LW strategy.

6.8 Summary

In this chapter, we formally define the problem of *safe multi-task learning*, and propose a simple and effective DSMTL method that can learn to combine the shared and task-specific representations. We theoretically analyze the proposed models and prove that the proposed DSMTL-IL and DSMTL-JL methods are guarantee to achieve some versions of *safe multi-task learning*. To solve the scalability issue of the proposed DSMTL-IL and DSMTL-JL methods, we further propose the DMSTL-AL method to learn a compact architecture via techniques in neural architecture search. Extensive experiments demonstrate the effectiveness of the proposed methods. In the future work, we are interested

in generalizing the DSMTL methods to other learning problems.

6.9 Proofs

In this section, we provide proofs for all the theorems.

6.9.1 Generalization Bound for Problem (6.4)

To help analyze the generalization bound of the DSMTL method, we first introduce a useful theorem in terms of Gaussian averages [2, 95].

Theorem 6.9.1. *Let $\mathcal{G}$ be a class of functions $\vartheta : \mathcal{X} \to [0,1]^\top$, and $\mu_1,...,\mu_m$ be the probability measure on $\mathcal{X}$ with $\bar{\mathbf{X}} = (\mathbf{X}_1,...,\mathbf{X}_m) \sim \prod_{t=1}^m (\mu_t)^n$, where $\mathbf{X}_t = (\mathbf{x}_t^1,\ldots,\mathbf{x}_t^n)$. Let and γ be a vector of independent standard normal variables and Z be the random set $\{(\vartheta_t(\mathbf{x}_t^i)) : \vartheta \in \mathcal{G}\}$ where ϑ_t are functions chosen from hypothesis class $\mathcal{G}$. Then for all $\vartheta \in \mathcal{G}$, with probability at least $1-\delta$, we have*

$$\frac{1}{m}\sum_t \left(\mathbb{E}_{\mathbf{x}\sim\mu_t}[\vartheta_t(\mathbf{x})] - \frac{1}{n}\sum_i \vartheta_t(\mathbf{x}_t^i) \right) \leq \frac{\sqrt{2\pi}G(Z)}{mn} + \sqrt{\frac{9\ln\frac{2}{\delta}}{2mn}},$$

where $G(Z) = \mathbb{E}[\sup_{z\in Z}\langle\gamma, z\rangle]$ is the Gaussian average of the random set Z.

Based on Theorem 6.9.1, we first establish the following uniform bound for problem (6.4).

Theorem 6.9.2. *Suppose Assumption 1 is satisfied. Then for $(\bar{\mathbf{X}}, \bar{\mathbf{Y}}) \sim \prod_{t=1}^m (\mu_t)^n$, with*

probability at least $1-\delta$*, we have*

$$\begin{aligned}&\mathscr{E}-\frac{1}{mn}\sum_{t,i}\mathscr{L}_t(\mathbf{y}_t^i,h_t(g_t(f_S(\mathbf{x}_t^i),f_t(\mathbf{x}_t^i))))\\&\leq\frac{C_1G(\mathscr{F}(\bar{\mathbf{X}}))}{mn}+\frac{C_2Q}{\sqrt{n}}+\sqrt{\frac{9\ln\frac{2}{\delta}}{2mn}},\end{aligned}\tag{6.11}$$

where C_1*,* C_2 *are two constants, the quantity* Q *is defined as*

$$Q=\sup_{z\neq\tilde{z}\in\mathbb{R}^{nq}}\frac{1}{\|z-\tilde{z}\|}\mathbb{E}\sup_{h_t\in\mathscr{H}}\sum_{i=1}^{n}\gamma_i(h_t(z_i)-h_t(\tilde{z}_i)),$$

and γ *is a vector of independent standard normal variables.*

Proof. According to Theorem 6.9.1, for $h_t\in\mathscr{H}$, $f_t,f_S\in\mathscr{F}$, and $\alpha_t\in\mathscr{M}$, with probability at least $1-\delta$, we have

$$\begin{aligned}&\mathscr{E}-\frac{1}{mn}\sum_{t,i}\mathscr{L}_t(\mathbf{y}_t^i,h_t(g_t(f_S(\mathbf{x}_t^i),f_t(\mathbf{x}_t^i))))\\&\leq\frac{\sqrt{2\pi}G(\mathscr{S})}{mn}+\sqrt{\frac{9\ln\frac{2}{\delta}}{2mn}},\end{aligned}$$

where $S=\{\mathscr{L}_t(\mathbf{y}_t^i,h_t(g_t(f_S(\mathbf{x}_t^i),f_t(\mathbf{x}_t^i))))\}\subseteq\mathbb{R}^{mn}$ and $G(S)$ represents the Gaussian average of the set S. Then by using the Lipschitz property of $\mathscr{L}_t$ and Slepian's Lemma [65], we have $G(S)\leq G(S')$, where $S'=\{h_t(g_t(f_S(\mathbf{x}_t^i),f_t(\mathbf{x}_t^i))):h_t\in\mathscr{H},\alpha_t\in\mathscr{M},f_t,f_S\in\mathscr{F}\}$.

Note that the input data $\bar{\mathbf{X}}\in\mathscr{X}^{mn}$ and the encoders $f_1,\ldots,f_m,f_S:\mathscr{X}\to\mathbb{R}^q$ are mapping functions chosen from $\mathscr{F}$, the random set $\mathscr{K}(\bar{\mathbf{X}})\subseteq\mathbb{R}^{mnq}$ is defined as $\mathscr{K}(\bar{\mathbf{X}})=\{g_t(f_S(\mathbf{x}_t^i),f_t(\mathbf{x}_t^i)):\alpha_t\in\mathscr{M},f_t,f_S\in\mathscr{F}\}$. We define a class of functions $\mathscr{H}'=\{z\in\mathbb{R}^{mnq}\mapsto h_t(z_t^i):h_t\in\mathscr{H}\}$. Therefore, we have $\mathscr{S}'(\bar{\mathbf{X}})=\mathscr{H}'(\mathscr{K}(\bar{\mathbf{X}}))$.

By using Theorem 2 in [94], we obtain

$$\begin{aligned}G(S')\leq c_1L(\mathscr{H}')G(\mathscr{K}(\bar{\mathbf{X}}))+c_2D(\mathscr{K}(\bar{\mathbf{X}}))Q(\mathscr{H}')\\+\min_{z\in\mathscr{K}(\bar{\mathbf{X}})}G(\mathscr{H}'(z)),\end{aligned}$$

where c_1 and c_2 are two constants, $L(\mathscr{H}')$ denotes the Lipschitz constant of the functions in $\mathscr{H}'$, $D(\mathscr{K}(\bar{\mathbf{X}})) = 2\sup_{\varphi\in\mathscr{K}} \|\varphi(\bar{\mathbf{X}})\|$ denotes the Euclidean diameter of the set $\mathscr{F}(\bar{\mathbf{X}})$, and

$$Q(\mathscr{H}') = \sup_{z\neq\tilde{z}\in\mathbb{R}^{mnq}} \frac{1}{\|z-\tilde{z}\|} \mathbb{E} \sup_{\psi\in\mathscr{H}'} \langle \gamma, \psi(z) - \psi(\tilde{z})\rangle .$$

Let $z, \tilde{z} \in \mathbb{R}^{mnq}$, where $z = (z_t^i)$, $\tilde{z} = (\tilde{z}_t^i)$ and $z_t^i, \tilde{z}_t^i \in \mathbb{R}^q$. Then for any functions $\psi \in \mathscr{H}'$, we have

$$\begin{aligned}
&\mathbb{E} \sup_{\psi\in\mathscr{H}'} \langle \gamma, \psi(z) - \psi(\tilde{z})\rangle \\
=& \sum_{t=1}^{m} \mathbb{E} \sup_{h\in\mathscr{H}} \sum_{i=1}^{n} \gamma_i (h(z_t^i) - h(\tilde{z}_t^i)) \\
\leq& \sqrt{m} \left(\sum_{t=1}^{m} (\mathbb{E} \sup_{h\in\mathscr{H}} \sum_{i=1}^{n} \gamma_i (h(z_t^i) - h(\tilde{z}_t^i)))^2 \right)^{1/2} \\
\leq& \sqrt{m} \left(\sum_{t=1}^{m} Q^2 \sum_{i=1}^{n} \|z_t^i - \tilde{z}_t^i\|^2 \right)^{1/2} \\
=& \sqrt{m} Q \|z - \tilde{z}\|,
\end{aligned}$$

where the first inequality is due to the Cauchy-Schwarz inequality and the second inequality is due to the inequality $\|z-\tilde{z}\| \leq \sup \|z-\tilde{z}\|$ holds for all $z \neq \tilde{z} \in \mathbb{R}^{nq}$. Therefore, $Q(\mathscr{H}') \leq \sqrt{m} Q$. Moreover, suppose the functions in hypothesis classes $\mathscr{H}$ are M-Lipschitz continuous, we have

$$\|\psi(z) - \psi(\tilde{z})\|^2 = \sum_{t,i} (h_t(z_t^i) - h_t(\tilde{z}_t^i))^2 \leq M^2 \|z - \tilde{z}\|^2$$

where the inequality is due to the Lipschitz property. So we obtain $L(\mathscr{H}') \leq M$. Since $0 \in \mathscr{F}$, we have $0 \in \mathscr{K}(\bar{\mathbf{X}})$. Note that $h(0) = 0$, and hence $\min_{z\in\mathscr{F}(\bar{\mathbf{X}})} G(\mathscr{H}'(z)) = 0$ by setting $z = 0$. Therefore, we have

$$G(S) \leq c_1 M G(\mathscr{K}(\bar{\mathbf{X}})) + 2c_2 \sqrt{m} Q \sup_{\varphi\in\mathscr{K}} \|\varphi(\bar{\mathbf{X}})\|. \tag{6.12}$$

Recall that the random set $\mathscr{F}(\bar{\mathbf{X}}) \subseteq \mathbb{R}^{mnq}$ is defined as $\mathscr{F}(\bar{\mathbf{X}}) = \{(f_t(\mathbf{x}_t^i)) : f_t \in \mathscr{F}\}$. Denote the jth entry of the vectors $f_S(\mathbf{x}_t^i)$ and $f_t(\mathbf{x}_t^i)$ by $f_{S,j}(\mathbf{x}_t^i)$ and $f_{t,j}(\mathbf{x}_t^i)$, respectively, and let $\gamma_{j,t,i}$ be the corresponding independent standard normal variable. Then we have

$$\begin{aligned} G(\mathscr{K}(\bar{\mathbf{X}})) &= \mathbb{E}\left[\sup_{f_t, f_S, \alpha_t} \sum_{j=1}^{q} \sum_{t,i} \gamma_{j,t,i} g_t(f_{S,j}(\mathbf{x}_t^i), f_{t,j}(\mathbf{x}_t^i)) \middle| \mathbf{x}_t^i\right] \\ &= \mathbb{E}\left[\sup_{f_t \in \mathscr{F}} \sum_{j=1}^{q} \sum_{t,i} \gamma_{j,t,i} f_{t,j}(\mathbf{x}_t^i)) \middle| \mathbf{x}_t^i\right] = G(\mathscr{F}(\bar{\mathbf{X}})), \end{aligned}$$

where the first and third equality are due to the definition of the Gaussian average, and the second equality holds since f_t and f_S are chosen from the same class $\mathscr{F}$ and $\mathscr{F}$ is uniformly bounded.

Since the hypothesis classes $\mathscr{F}$ is uniformly bounded, suppose that $\|f(\mathbf{x}_t^i)\| \leq R$ for all $f \in \mathscr{F}$ and we have

$$\begin{aligned} \sup_{\varphi \in \mathscr{K}} \|\varphi(\bar{\mathbf{X}})\| &= \sqrt{mn} \sup_{f_t, f_S, \alpha_t} \max_{i,t} \|g_t(f_S(\mathbf{x}_t^i), f_t(\mathbf{x}_t^i))\| \\ &\leq \sqrt{mn} \sup_{f \in \mathscr{F}} \max_{i,t} \|f(\mathbf{x}_t^i)\| \leq \sqrt{mn} R, \end{aligned}$$

where the first inequality holds since f_t and f_S are chosen from the same class $\mathscr{F}$, and the second inequality holds since $\|f(\mathbf{x}_t^i)\| \leq R$. Therefore, we get

$$\begin{aligned} &\mathscr{E} - \frac{1}{mn} \sum_{t,i} \mathscr{L}_t(\mathbf{y}_t^i, h_t(g_t(f_S(\mathbf{x}_t^i), f_t(\mathbf{x}_t^i)))) \\ &\leq \frac{c_1 M \sqrt{2\pi} G(\mathscr{F}(\bar{\mathbf{X}}))}{mn} + \frac{2c_2 R \sqrt{2\pi} Q}{\sqrt{n}} + \sqrt{\frac{9 \ln \frac{2}{\delta}}{2mn}}. \end{aligned}$$

By setting $C_1 = c_1 M \sqrt{2\pi}$ and $C_2 = 2c_2 R \sqrt{2\pi}$, we reach the conclusion. ■

Remark 8. *According to the McDiarmid inequality [98], the upper bound in Theorem 6.9.1 also holds for the sum of $\vartheta_t(\mathbf{x}_t^i)$ minus its expectation. Thus following the same proof*

as above, we can verify that the following inequality holds under the same assumption in Theorem 6.9.1:

$$\begin{aligned}&\frac{1}{mn}\sum_{t,i}\mathscr{L}_t(\mathbf{y}_t^i, h_t(g_t(f_S(\mathbf{x}_t^i), f_t(\mathbf{x}_t^i)))) - \mathscr{E}\\ &\leq \frac{C_1 G(\mathscr{F}(\bar{\mathbf{X}}))}{mn} + \frac{C_2 Q}{\sqrt{n}} + \sqrt{\frac{9\ln\frac{2}{\delta}}{2mn}}.\end{aligned} \tag{6.13}$$

6.9.2 Proof of Theorem 6.5.1

Proof. The STL model aims to solve the following optimization problem as

$$\min_{h_t^{\mathrm{STL}}\in\mathscr{H}, f_t^{\mathrm{STL}}\in\mathscr{F}} \frac{1}{mn}\sum_{i=1}^{n}\sum_{t=1}^{m}\mathscr{L}_t(\mathbf{y}_t^i, h_t^{\mathrm{STL}}(f_t^{\mathrm{STL}}(\mathbf{x}_t^i))),$$

where its solution is denoted by $\{\hat{h}_t^{\mathrm{STL}}\}$ and $\{\hat{f}_t^{\mathrm{STL}}\}$. Therefore, the minimal empirical loss of the STL model on task t is computed as

$$\hat{L}_t^{\mathrm{STL}} = \frac{1}{n}\sum_{i=1}^{n}\mathscr{L}_t(\mathbf{y}_t^i, \hat{h}_t^{\mathrm{STL}}(\hat{f}_t^{\mathrm{STL}}(\mathbf{x}_t^i))).$$

For the DSMTL-IL model, we set $\alpha_t = 0$ for all tasks in the first training stage, thus the corresponding objective function is the same as that of the STL model. Therefore, the solution of the first stage in the DSMTL-IL model, i.e., $\{\hat{h}_t\}$ and $\{\hat{f}_t\}$, satisfies $\hat{h}_t = \hat{h}_t^{\mathrm{STL}}$ and $\hat{f}_t = \hat{f}_t^{\mathrm{STL}}$. In the second training stage of the DSMTL-IL model, for any public encoder f_S in task t, we have

$$\begin{aligned}&\min_{\alpha_t\in\mathscr{M}}\frac{1}{n}\sum_{i=1}^{n}\mathscr{L}_t(\mathbf{y}_t^i, \hat{h}_t(\alpha_t f_S(\mathbf{x}_t^i) + (1-\alpha_t)\hat{f}_t(\mathbf{x}_t^i)))\\ &\leq \frac{1}{n}\sum_{i=1}^{n}\mathscr{L}_t(\mathbf{y}_t^i, \hat{h}_t^{\mathrm{STL}}(\hat{f}_t^{\mathrm{STL}}(\mathbf{x}_t^i))) = \hat{L}_t^{\mathrm{STL}}.\end{aligned}$$

Therefore, $L_t^* \leq \hat{L}_t^{\mathrm{STL}} \leq L_t^{\mathrm{STL}}$ holds for all $1 \leq t \leq m$. This finishes the proof. ■

6.9.3 Proof of Theorem 6.5.2

Proof. According to Theorem 6.9.2, with m tasks and data $\bar{\mathbf{X}}$, we have following inequality

$$\begin{aligned}\frac{1}{m}\sum_{t=1}^{m}\mathscr{E}_t - \frac{1}{mn}\sum_{t,i}\mathscr{L}_t(\mathbf{y}_t^i, h_t(g_t(f_S(\mathbf{x}_t^i), f_t(\mathbf{x}_t^i))))\\ \leq \frac{c_1 G(\mathscr{F}(\bar{\mathbf{X}}))}{mn} + \frac{c_2 Q}{\sqrt{n}} + \sqrt{\frac{9\ln\frac{2}{\delta}}{2mn}},\end{aligned} \tag{6.14}$$

holds with probability at least $1-\delta$. Thus, for one single task such as task t and its corresponding data $\mathbf{X}_t$, setting the number of tasks in the above inequality to be 1 gives

$$\begin{aligned}\mathscr{E}_t - \frac{1}{n}\sum_{i}\mathscr{L}_t(\mathbf{y}_t^i, h_t(g_t(f_S(\mathbf{x}_t^i), f_t(\mathbf{x}_t^i))))\\ \leq \frac{c_1 G(\mathscr{F}'(\mathbf{X}_t))}{n} + \frac{c_2 Q}{\sqrt{n}} + \sqrt{\frac{9\ln\frac{2}{\delta}}{2n}}.\end{aligned} \tag{6.15}$$

By substituting the solution of problem (6.3) into inequality (6.15), we have following inequality as

$$\hat{\mathscr{E}}_t - L_t^* \leq \frac{c_1 G(\mathscr{F}'(\mathbf{X}_t))}{n} + \frac{c_2 Q}{\sqrt{n}} + \sqrt{\frac{9\ln\frac{2}{\delta}}{2n}}. \tag{6.16}$$

Moreover, since the STL model can be considered as a special case of the DSMTL-IL model where g_t adopts g_t^0 defined in Eq. (6.5), substituting it into the inequality (6.13) gives

$$L^{\mathrm{STL}} - \mathscr{E}^{\mathrm{STL}} \leq \frac{c_1' G(\mathscr{F}(\bar{\mathbf{X}}))}{nm} + \frac{c_2' Q}{\sqrt{n}} + \sqrt{\frac{9\ln\frac{2}{\delta}}{2mn}}. \tag{6.17}$$

Therefore, for the STL model in task t, setting m to 1 in the above inequality gives

$$L_t^{\mathrm{STL}} - \mathscr{E}_t^{\mathrm{STL}} \leq \frac{c_1' G(\mathscr{F}'(\mathbf{X}_t))}{n} + \frac{c_2' Q}{\sqrt{n}} + \sqrt{\frac{9\ln\frac{2}{\delta}}{2n}}. \tag{6.18}$$

Add the inequalities (6.16) and (6.18) we get

$$\hat{\mathscr{E}}_t - L_t^* + L_t^{\mathrm{STL}} - \mathscr{E}_t^{\mathrm{STL}} \leq \frac{C_1 G(\mathscr{F}'(\mathbf{X}_t))}{n} + \frac{C_2 Q}{\sqrt{n}} + \sqrt{\frac{18 \ln \frac{2}{\delta}}{n}},$$

where C_1 and C_2 are two constants and $\mathscr{F}'(\bar{\mathbf{X}}_t) = \{f(\mathbf{x}_t^i) : f \in \mathscr{F}\} \subseteq \mathbb{R}^{nq}$. According to Theorem 6.5.1, there exists a constant $\varepsilon_t \geq 0$ such that $\hat{L}_t^* + \varepsilon_t = L_t^{\mathrm{STL}}$. Therefore, we have

$$\hat{\mathscr{E}}_t + \epsilon_t \leq \mathscr{E}_t^{\mathrm{STL}} + \frac{C_1 G(\mathscr{F}'(\mathbf{X}_t))}{n} + \frac{C_2 Q}{\sqrt{n}} + \sqrt{\frac{18 \ln \frac{2}{\delta}}{n}},$$

which completes the proof. ■

6.9.4 Proof of Theorem 6.5.3

Proof. It is easy to see that the STL model can be considered as a special case of the DSMTL-JL model when $\alpha_t = 0$ holds for $1 \leq t \leq m$ and the HPS model is also a special case of the DSMTL-JL model when $\alpha_t = 1$ holds for $1 \leq t \leq m$.

The empirical loss of the DSMTL model is formulated as

$$L = \frac{1}{mn} \sum_{i=1}^{n} \sum_{t=1}^{m} \mathscr{L}_t(\mathbf{y}_t^i, h_t(g_t(f_S(\mathbf{x}_t^i), f_t(\mathbf{x}_t^i)))).$$

We have $L^{\mathrm{STL}} = L(\Theta)\,|_{\alpha_t=0}$ and $L^{\mathrm{HPS}} = L(\Theta)\,|_{\alpha_t=1}$. Since $L^* \leq L$, we can get $L^* \leq \min\{L^{\mathrm{STL}}, L^{\mathrm{HPS}}\}$ and hence we reach the conclusion. ■

6.9.5 Proof of Theorem 6.5.4

Proof. Let L^* be the optimal value of problem (6.4). Substituting the solution of problem (6.4) into inequality (6.11) gives

$$\hat{\mathscr{E}} - L^* \leq \frac{c_1 G(\mathscr{F}(\bar{\mathbf{X}}))}{mn} + \frac{c_2 Q}{\sqrt{n}} + \sqrt{\frac{9 \ln \frac{2}{\delta}}{2mn}}. \tag{6.19}$$

Based on inequalities (6.19) and (6.17), with probability $1-\delta$, we have

$$\hat{\mathscr{E}} - L^* + L^{\mathrm{STL}} - \mathscr{E}^{\mathrm{STL}} \leq \frac{C_1 G(\mathscr{F}(\bar{\mathbf{X}}))}{nm} + \frac{C_2 Q}{\sqrt{n}} + \sqrt{\frac{18 \ln \frac{2}{\delta}}{mn}},$$

where C_1 and C_2 are two constants. According to Theorem 6.5.3, there exists a constant $\varepsilon \geq 0$ such that $L^* + \varepsilon = L^{\mathrm{STL}}$. Therefore, we have

$$\hat{\mathscr{E}} + \epsilon \leq \mathscr{E}^{\mathrm{STL}} + \frac{C_1 G(\mathscr{F}(\bar{\mathbf{X}}))}{nm} + \frac{C_2 Q}{\sqrt{n}} + \sqrt{\frac{18 \ln \frac{2}{\delta}}{mn}},$$

where we reach the conclusion. ■

6.9.6 Proof of Theorem 6.5.5

Proof. Let $\{f_t\}^*$, f_S^*, $\{h_t\}^*$, $\{g_t\}^*$ be the minimizer in $\mathscr{E}^*$. We can decompose $\hat{\mathscr{E}} - \mathscr{E}^*$ as

$$\begin{aligned}
\hat{\mathscr{E}} - \mathscr{E}^* = & \Big(\hat{\mathscr{E}} - \frac{1}{mn}\sum_{t,i} \mathscr{L}_t(\mathbf{y}_t^i, h_t(g_t(f_S(\mathbf{x}_t^i), f_t(\mathbf{x}_t^i))\Big) \\
& + \Big(\frac{1}{mn}\sum_{t,i} \mathscr{L}_t(\mathbf{y}_t^i, h_t(g_t(f_S(\mathbf{x}_t^i), f_t(\mathbf{x}_t^i)) \\
& \quad - \frac{1}{mn}\sum_{t,i} \mathscr{L}_t(\mathbf{y}_t^i, h_t^*(g_t^*(f_S^*(\mathbf{x}_t^i), f_t^*(\mathbf{x}_t^i))\Big) \\
& + \Big(\frac{1}{mn}\sum_{t,i} \mathscr{L}_t(\mathbf{y}_t^i, h_t^*(g_t^*(f_S^*(\mathbf{x}_t^i), f_t^*(\mathbf{x}_t^i)) - \mathscr{E}^*\Big),
\end{aligned}$$

where the first term can be bounded by substituting inequality (6.11) and the last term can be regarded as mn random variables $\mathscr{L}_t(\mathbf{y}_t^i, h_t^*(g_t^*(f_S^*(\mathbf{x}_t^i), f_t^*(\mathbf{x}_t^i))$ with values in $[0,1]$. By using Hoeffding's inequality, with probability at least $1-\delta$, we have

$$\frac{1}{mn}\sum_{ti} \mathscr{L}_t(\mathbf{y}_t^i, h_t^*(\omega_t^{*T} \varphi^*(\mathbf{x}_t^i))) - \mathscr{E}^* \leq \sqrt{\frac{\ln \frac{1}{\delta}}{2mn}}.$$

The second term is non-positive due to the definition of minimizers. Therefore, we have

$$\hat{\mathscr{E}} - \mathscr{E}^* \leq \frac{C_1 G(\mathscr{F}(\bar{\mathbf{X}}))}{mn} + \frac{C_2 Q}{\sqrt{n}} + \sqrt{\frac{8 \ln \frac{4}{\delta}}{mn}},$$

where we reach the conclusion. ■

CHAPTER 7

CONCLUSION AND FUTURE STUDY

7.1 Conclusions

This thesis focuses on how to learn the effective architecture of transfer learning models automatically. Specifically, this thesis addresses four research questions for transfer learning problems in real-world: 1) How to design a deep neural network architecture that satisfies multiple objectives; 2) How to design a feature alignment architecture with varying difficulty levels of domain adaptation tasks; 3) How to alleviate the gradient conflict in multi-task learning; 4) How to guarantee that multi-task learning performance is no worse than its single-task counterpart on each task.

To solve these challenges, this thesis proposes three corresponding research objectives to conduct studies: 1) To automatically design an architecture by explicitly balancing

the trade-off among multiple objectives; 2) To learn feature alignment architecture and domain-invariant feature representations for domain adaptation tasks with varying difficulty levels; 3) To alleviate the gradient conflict in multi-task learning by learning architectures; and 4) To achieve safe multi-task learning where no negative sharing occurs. The findings of these studies are summarized as follows:

To achieve research objective 1, we propose an effective, efficient, and robust neural architecture search method to design architecture by explicitly balancing the trade-off among the performance, resource consumption, and robustness. We formulate the objective function of the proposed method as a multi-objective bi-level optimization problem and propose an efficient gradient-based algorithm to solve it.

To achieve research objective 2, we propose a new similarity measure and corresponding alignment architecture search method to learn domain-alignment architecture and domain-invariant feature representation. We further develop the first architecture learning framework designed for the distance-based domain adaptation method.

To achieve research objective 3, we propose a compact architecture learning method with good scalability to circumvent negative transfer in multi-task learning, which first introduces purely-specific modules into the search space to mitigate the gradient conflict. The proposed method automatically learns when to switch to purely-specific modules in the tree-structured network architectures when the gradient conflict occurs.

To achieve research objective 4, we propose a safe multi-task learning model, which learns how to combine the private encoder and public encoder for the downstream private decoder. The proposed method mitigates negative sharing in multi-task learning and

improves the performance of all tasks compared with single task learning.

7.2 Future Study

This thesis identifies the following directions for future study:

- *Multi-domain multi-task problem* Existing multi-task learning methods mostly focus on homogeneous-feature and heterogeneous-task problems [152], where different tasks share the same input data but are of different types with respective loss functions. The feature space is heterogeneous in multi-domain multi-task problems, and each task has its own input data. Under this scenario, each task requires its own feature extraction module, which complicates the design of multi-task network architecture. We aim to propose an architecture search technique to solve this challenge in future work.

- *Adaptive architecture for continues learning* In real-world applications of deep learning, new tasks can arrive over time from a data stream. Continual learning enables deep learning models to learn and adapt to new tasks over time without forgetting previously learned knowledge. We are interested in extending the CoNAL method to the continual learning setting and learning incremental architecture for newly arrived tasks without mitigating the performance of previously learned tasks.

- *Multi-modal multi-task learning* Real-world scenarios usually provide complementary information about the same group of tasks. Multi-modal multi-task learning

involves training a single model to perform multiple tasks using multiple input data modalities. Multi-task learning methods can potentially train more efficient and effective multi-modal models by sharing parameters and leveraging the knowledge from multiple modalities.

BIBLIOGRAPHY

[1] R. ARDYWIBOWO, S. BOLUKI, X. GONG, Z. WANG, AND X. QIAN, *NADS: neural architecture distribution search for uncertainty awareness*, in Proceedings of the 37th International Conference on Machine Learning, vol. 119, 2020, pp. 356–366.

[2] P. L. BARTLETT AND S. MENDELSON, *Rademacher and gaussian complexities: Risk bounds and structural results*, Journal of Machine Learning Research, 3 (2002), pp. 463–482.

[3] S. BEN-DAVID, J. BLITZER, K. CRAMMER, A. KULESZA, F. PEREIRA, AND J. W. VAUGHAN, *A theory of learning from different domains*, Machine learning, 79 (2010), pp. 151–175.

[4] S. BEN-DAVID, J. BLITZER, K. CRAMMER, F. PEREIRA, ET AL., *Analysis of representations for domain adaptation*, Advances in neural information processing systems, 19 (2007), p. 137.

[5] H. BENMEZIANE, K. E. MAGHRAOUI, H. OUARNOUGHI, S. NIAR, M. WISTUBA, AND N. WANG, *A comprehensive survey on hardware-aware neural architecture search*, arXiv preprint arXiv:2101.09336, (2021).

[6] K. BI, L. XIE, X. CHEN, L. WEI, AND Q. TIAN, *Gold-nas: Gradual, one-level, differentiable*, arXiv preprint arXiv:2007.03331, (2020).

[7] F. J. BRAGMAN, R. TANNO, S. OURSELIN, D. C. ALEXANDER, AND J. CARDOSO, *Stochastic filter groups for multi-task cnns: Learning specialist and generalist convolution kernels*, in Proceedings of the IEEE/CVF International Conference on Computer Vision, 2019, pp. 1385–1394.

[8] D. BRUGGEMANN, M. KANAKIS, S. GEORGOULIS, AND L. V. GOOL, *Automated search for resource-efficient branched multi-task networks*, in 31st British Ma-

chine Vision Conference 2020, BMVC 2020, Virtual Event, UK, September 7-10, 2020, BMVA Press, 2020.

[9] D. BRÜGGEMANN, M. KANAKIS, A. OBUKHOV, S. GEORGOULIS, AND L. VAN GOOL, *Exploring relational context for multi-task dense prediction*, in Proceedings of the IEEE/CVF International Conference on Computer Vision, 2021, pp. 15869–15878.

[10] H. CAI, C. GAN, T. WANG, Z. ZHANG, AND S. HAN, *Once-for-all: Train one network and specialize it for efficient deployment*, in Proceedings of the 8th International Conference on Learning Representations, 2020.

[11] H. CAI, L. ZHU, AND S. HAN, *Proxylessnas: Direct neural architecture search on target task and hardware*, in Proceedings of the 7th International Conference on Learning Representations, 2019.

[12] J. CAO, Y. LI, AND Z. ZHANG, *Partially shared multi-task convolutional neural network with local constraint for face attribute learning*, in Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, 2018, pp. 4290–4299.

[13] R. CARUANA, *Multitask learning*, Machine Learning, 28 (1997), pp. 41–75.

[14] C. CHEN, Z. CHEN, B. JIANG, AND X. JIN, *Joint domain alignment and discriminative feature learning for unsupervised deep domain adaptation*, in Proceedings of the AAAI conference on artificial intelligence, vol. 33, 2019, pp. 3296–3303.

[15] L.-C. CHEN, G. PAPANDREOU, I. KOKKINOS, K. MURPHY, AND A. 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 (2017), pp. 834–848.

[16] L.-C. CHEN, G. PAPANDREOU, F. SCHROFF, AND H. ADAM, *Rethinking atrous convolution for semantic image segmentation*, arXiv preprint arXiv:1706.05587, (2017).

[17] X. CHEN, L. XIE, J. WU, AND Q. TIAN, *Progressive differentiable architecture search: Bridging the depth gap between search and evaluation*, in Proceedings of the IEEE International Conference on Computer Vision, 2019, pp. 1294–1303.

[18] Z. CHEN, V. BADRINARAYANAN, C.-Y. LEE, AND A. RABINOVICH, *Gradnorm: Gradient normalization for adaptive loss balancing in deep multitask networks*, in International Conference on Machine Learning, PMLR, 2018, pp. 794–803.

[19] Z. CHEN, J. NGIAM, Y. HUANG, T. LUONG, H. KRETZSCHMAR, Y. CHAI, AND D. ANGUELOV, *Just pick a sign: Optimizing deep multitask models with gradient sign dropout*, Advances in Neural Information Processing Systems, 33 (2020), pp. 2039–2050.

[20] S. CHENNUPATI, G. SISTU, S. YOGAMANI, AND S. 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, 2019, pp. 0–0.

[21] M. CORDTS, M. OMRAN, S. RAMOS, T. REHFELD, M. ENZWEILER, R. BENENSON, U. FRANKE, S. ROTH, AND B. SCHIELE, *The cityscapes dataset for semantic urban scene understanding*, in Proceedings of the IEEE conference on computer vision and pattern recognition, 2016, pp. 3213–3223.

[22] N. COURTY, R. FLAMARY, D. TUIA, AND A. RAKOTOMAMONJY, *Optimal transport for domain adaptation*, IEEE transactions on pattern analysis and machine intelligence, 39 (2016), pp. 1853–1865.

[23] C. CUI, Z. SHEN, J. HUANG, M. CHEN, M. XU, M. WANG, AND Y. YIN, *Adaptive feature aggregation in deep multi-task convolutional neural networks*, IEEE Transactions on Circuits and Systems for Video Technology, (2021).

[24] J. CUI, P. CHEN, R. LI, S. LIU, X. SHEN, AND J. JIA, *Fast and practical neural architecture search*, in Proceedings of the IEEE International Conference on Computer Vision, 2019, pp. 6509–6518.

[25] J. DENG, W. DONG, R. SOCHER, L.-J. LI, K. LI, AND L. FEI-FEI, *Imagenet: A large-scale hierarchical image database*, in 2009 IEEE conference on computer vision and pattern recognition, Ieee, 2009, pp. 248–255.

[26] L. M. DERY, Y. DAUPHIN, AND D. GRANGIER, *Auxiliary task update decomposition: The good, the bad and the neutral*, arXiv preprint arXiv:2108.11346, (2021).

[27] J.-A. DÉSIDÉRI, *Multiple-gradient descent algorithm (MGDA) for multiobjective optimization*, Comptes Rendus Mathematique, 350 (2012), pp. 313–318.

[28] C. DEVAGUPTAPU, D. AGARWAL, G. MITTAL, AND V. N. BALASUBRAMANIAN, *On adversarial robustness: A neural architecture search perspective*, arXiv preprint arXiv:2007.08428, (2020).

[29] J. DEVLIN, M.-W. CHANG, K. LEE, AND K. TOUTANOVA, *Bert: Pre-training of deep bidirectional transformers for language understanding*, in Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers), 2019, pp. 4171–4186.

[30] J. DONAHUE, Y. JIA, O. VINYALS, J. HOFFMAN, N. ZHANG, E. TZENG, AND T. DARRELL, *Decaf: A deep convolutional activation feature for generic visual recognition*, in International conference on machine learning, PMLR, 2014, pp. 647–655.

[31] Y. DU, W. M. CZARNECKI, S. M. JAYAKUMAR, M. FARAJTABAR, R. PASCANU, AND B. LAKSHMINARAYANAN, *Adapting auxiliary losses using gradient similarity*, arXiv preprint arXiv:1812.02224, (2018).

[32] T. ELSKEN, J. H. METZEN, AND F. HUTTER, *Efficient multi-objective neural architecture search via lamarckian evolution*, in Proceedings of the 7th International Conference on Learning Representations, 2019.

[33] T. EVGENIOU AND M. PONTIL, *Regularized multi–task learning*, in Proceedings of the tenth ACM SIGKDD international conference on Knowledge discovery and data mining, 2004, pp. 109–117.

[34] C. FIFTY, E. AMID, Z. ZHAO, T. YU, R. ANIL, AND C. FINN, *Efficiently identifying task groupings for multi-task learning*, Advances in Neural Information Processing Systems, 34 (2021).

[35] L. FRANCESCHI, P. FRASCONI, S. SALZO, R. GRAZZI, AND M. PONTIL, *Bilevel programming for hyperparameter optimization and meta-learning*, in International Conference on Machine Learning, PMLR, 2018, pp. 1568–1577.

[36] Y. GANIN AND V. LEMPITSKY, *Unsupervised domain adaptation by backpropagation*, in International conference on machine learning, PMLR, 2015, pp. 1180–1189.

[37] Y. GANIN, E. USTINOVA, H. AJAKAN, P. GERMAIN, H. LAROCHELLE, F. LAVIOLETTE, M. MARCHAND, AND V. LEMPITSKY, *Domain-adversarial training of neural networks*, The journal of machine learning research, 17 (2016), pp. 2096–2030.

[38] Y. GAO, H. BAI, Z. JIE, J. MA, K. JIA, AND W. LIU, *MTL-NAS: task-agnostic neural architecture search towards general-purpose multi-task learning*, in 2020 IEEE/CVF Conference on Computer Vision and Pattern Recognition, CVPR 2020, Seattle, WA, USA, June 13-19, 2020, Computer Vision Foundation / IEEE, 2020, pp. 11540–11549.

[39] Y. GAO, J. MA, M. ZHAO, W. LIU, AND A. 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, 2019, pp. 3205–3214.

[40] G. GHIASI, T.-Y. LIN, AND Q. V. LE, *Nas-fpn: Learning scalable feature pyramid architecture for object detection*, in Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition, 2019, pp. 7036–7045.

[41] S. GREEN, C. M. VINEYARD, R. HELINSKI, AND Ç. K. KOÇ, *RAPDARTS: resource-aware progressive differentiable architecture search*, in Proceedings of the International Joint Conference on Neural Networks, 2020, pp. 1–7.

[42] L.-Z. GUO, Z.-Y. ZHANG, Y. JIANG, Y.-F. LI, AND Z.-H. ZHOU, *Safe deep semi-supervised learning for unseen-class unlabeled data*, in ICML, 2020.

[43] M. GUO, Y. YANG, R. XU, Z. LIU, AND D. LIN, *When NAS meets robustness: In search of robust architectures against adversarial attacks*, in Proceedings of the IEEE International Conference on Computer Vision, 2020, pp. 628–637.

[44] P. GUO, C. DENG, L. XU, X. HUANG, AND Y. ZHANG, *Deep multi-task augmented feature learning via hierarchical graph neural network*, in Joint European Conference on Machine Learning and Knowledge Discovery in Databases, Springer, 2021, pp. 538–553.

[45] P. GUO, C. LEE, AND D. ULBRICHT, *Learning to branch for multi-task learning*, in Proceedings of the 37th International Conference on Machine Learning, ICML 2020, 13-18 July 2020, Virtual Event, vol. 119 of Proceedings of Machine Learning Research, PMLR, 2020, pp. 3854–3863.

[46] P. GUO, C.-Y. LEE, AND D. ULBRICHT, *Learning to branch for multi-task learning*, in International Conference on Machine Learning, PMLR, 2020, pp. 3854–3863.

[47] P. GUO, F. YE, AND Y. ZHANG, *Safe multi-task learning*, arXiv preprint arXiv:2111.10601, (2021).

[48] T. HAARNOJA, A. ZHOU, P. ABBEEL, AND S. LEVINE, *Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor*, in International conference on machine learning, PMLR, 2018, pp. 1861–1870.

[49] H. HAN, A. K. JAIN, F. WANG, S. SHAN, AND X. CHEN, *Heterogeneous face attribute estimation: A deep multi-task learning approach*, IEEE TPAMI, (2017).

[50] K. HE, G. GKIOXARI, P. DOLLÁR, AND R. GIRSHICK, *Mask r-cnn*, in Proceedings of the IEEE international conference on computer vision, 2017, pp. 2961–2969.

[51] K. HE, X. ZHANG, S. REN, AND J. SUN, *Deep residual learning for image recognition*, in Proceedings of the IEEE conference on computer vision and pattern recognition, 2016, pp. 770–778.

[52] D. HENDRYCKS AND K. GIMPEL, *A baseline for detecting misclassified and out-of-distribution examples in neural networks*, in Proceedings of the 5th International Conference on Learning Representations, 2017.

[53] J. HU, S. RUDER, A. SIDDHANT, G. NEUBIG, O. FIRAT, AND M. JOHNSON, *Xtreme: A massively multilingual multi-task benchmark for evaluating cross-lingual generalisation*, in International Conference on Machine Learning, PMLR, 2020, pp. 4411–4421.

[54] S. HU, S. XIE, H. ZHENG, C. LIU, J. SHI, X. LIU, AND D. LIN, *Dsnas: Direct neural architecture search without parameter retraining*, in Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition, 2020, pp. 12084–12092.

[55] G. HUANG, Z. LIU, L. VAN DER MAATEN, AND K. Q. WEINBERGER, *Densely connected convolutional networks*, in Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, 2017, pp. 4700–4708.

[56] A. JAVALOY AND I. VALERA, *Rotograd: Gradient homogenization in multitask learning*, in International Conference on Learning Representations, 2022.

[57] X. JIN, J. WANG, J. SLOCUM, M.-H. YANG, S. DAI, S. YAN, AND J. FENG, *Rc-darts: Resource constrained differentiable architecture search*, arXiv preprint arXiv:1912.12814, (2019).

[58] A. KENDALL, Y. GAL, AND R. CIPOLLA, *Multi-task learning using uncertainty to weigh losses for scene geometry and semantics*, in Proceedings of the IEEE conference on computer vision and pattern recognition, 2018, pp. 7482–7491.

[59] D. P. KINGMA AND J. BA, *Adam: A method for stochastic optimization*, in 3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings, Y. Bengio and Y. LeCun, eds., 2015.

[60] A. KRIZHEVSKY, G. HINTON, ET AL., *Learning multiple layers of features from tiny images*, Technical Report TR-2009, (2009).

[61] A. KRIZHEVSKY, I. SUTSKEVER, AND G. E. HINTON, *Imagenet classification with deep convolutional neural networks*, Advances in neural information processing systems, 25 (2012), pp. 1097–1105.

[62] H. W. KUHN AND A. W. TUCKER, *Nonlinear programming*, in Traces and Emergence of Nonlinear Programming, Springer, 2014, pp. 247–258.

[63] A. KUMAR AND H. DAUME III, *Learning task grouping and overlap in multi-task learning*, arXiv preprint arXiv:1206.6417, (2012).

[64] A. KURAKIN, I. J. GOODFELLOW, AND S. BENGIO, *Adversarial examples in the physical world*, in Proceedings of the 5th International Conference on Learning Representations, 2017.

[65] M. LEDOUX AND M. TALAGRAND, *Probability in Banach Spaces: isoperimetry and processes*, Springer Science & Business Media, 2013.

[66] G. LEE, E. YANG, AND S. HWANG, *Asymmetric multi-task learning based on task relatedness and loss*, in International conference on machine learning, PMLR, 2016, pp. 230–238.

[67] K. LEE, H. LEE, K. LEE, AND J. SHIN, *Training confidence-calibrated classifiers for detecting out-of-distribution samples*, in Proceedings of the 6th International Conference on Learning Representations, 2018.

[68] S. LI, C. LIU, Q. LIN, B. XIE, Z. DING, G. HUANG, AND J. TANG, *Domain conditioned adaptation network*, in Proceedings of the AAAI Conference on Artificial Intelligence, vol. 34, 2020, pp. 11386–11393.

[69] S. LI, M. XIE, K. GONG, C. H. LIU, Y. WANG, AND W. LI, *Transferable semantic augmentation for domain adaptation*, in Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition, 2021, pp. 11516–11525.

[70] Y.-F. LI, L.-Z. GUO, AND Z.-H. ZHOU, *Towards safe weakly supervised learning*, IEEE TPAMI, (2019).

[71] Y.-F. LI AND Z.-H. ZHOU, *Towards making unlabeled data never hurt*, IEEE TPAMI, (2014).

[72] J. Z. LIANG, E. MEYERSON, AND R. MIIKKULAINEN, *Evolutionary architecture search for deep multitask networks*, in Proceedings of the Genetic and Evolutionary Computation Conference, GECCO 2018, Kyoto, Japan, July 15-19, 2018, H. E. Aguirre and K. Takadama, eds., ACM, 2018, pp. 466–473.

[73] B. LIN, F. YE, AND Y. ZHANG, *A closer look at loss weighting in multi-task learning*, arXiv preprint arXiv:2111.10603, (2021).

[74] A.-A. LIU, Y.-T. SU, W.-Z. NIE, AND M. KANKANHALLI, *Hierarchical clustering multi-task learning for joint human action grouping and recognition*, IEEE TPAMI, (2016).

[75] B. LIU, X. LIU, X. JIN, P. STONE, AND Q. LIU, *Conflict-averse gradient descent for multi-task learning*, Advances in Neural Information Processing Systems, 34 (2021).

[76] C. LIU, B. ZOPH, M. NEUMANN, J. SHLENS, W. HUA, L.-J. LI, L. FEI-FEI, A. YUILLE, J. HUANG, AND K. MURPHY, *Progressive neural architecture*

search, in Proceedings of the European Conference on Computer Vision, 2018, pp. 19–34.

[77] H. LIU, K. SIMONYAN, AND Y. YANG, *DARTS: differentiable architecture search*, in 7th International Conference on Learning Representations, ICLR 2019, New Orleans, LA, USA, May 6-9, 2019, OpenReview.net, 2019.

[78] J. LIU AND Y. JIN, *Multi-objective search of robust neural architectures against multiple types of adversarial attacks*, arXiv preprint arXiv:2101.06507, (2021).

[79] L. LIU, Y. LI, Z. KUANG, J. XUE, Y. CHEN, W. YANG, Q. LIAO, AND W. ZHANG, *Towards impartial multi-task learning*, in International Conference on Learning Representations, 2021.

[80] S. LIU, E. JOHNS, AND A. J. DAVISON, *End-to-end multi-task learning with attention*, in Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition, 2019, pp. 1871–1880.

[81] W. LIU, T. MEI, Y. ZHANG, C. CHE, AND J. LUO, *Multi-task deep visual-semantic embedding for video thumbnail selection*, in Proceedings of IEEE Conference on Computer Vision and Pattern Recognition, 2015, pp. 3707–3715.

[82] Z. LIU, P. LUO, X. WANG, AND X. TANG, *Deep learning face attributes in the wild*, in Proceedings of the IEEE international conference on computer vision, 2015, pp. 3730–3738.

[83] M. LONG, Z. CAO, J. WANG, AND M. I. JORDAN, *Conditional adversarial domain adaptation*, Advances in neural information processing systems, 31 (2018).

[84] M. LONG AND J. WANG, *Learning multiple tasks with deep relationship networks*, arXiv preprint arXiv:1506.02117, 2 (2015).

[85] M. LONG, J. WANG, G. DING, J. SUN, AND P. S. YU, *Transfer feature learning with joint distribution adaptation*, in Proceedings of the IEEE international conference on computer vision, 2013, pp. 2200–2207.

[86] M. LONG, H. ZHU, J. WANG, AND M. I. JORDAN, *Unsupervised domain adaptation with residual transfer networks*, arXiv preprint arXiv:1602.04433, (2016).

[87] M. LONG, H. ZHU, J. WANG, AND M. I. JORDAN, *Deep transfer learning with joint adaptation networks*, in Proceedings of the 34th International Conference on

Machine Learning, ICML 2017, Sydney, NSW, Australia, 6-11 August 2017, D. Precup and Y. W. Teh, eds., vol. 70 of Proceedings of Machine Learning Research, PMLR, 2017, pp. 2208–2217.

[88] I. LOSHCHILOV AND F. HUTTER, *Decoupled weight decay regularization*, in International Conference on Learning Representations, 2018.

[89] Y. LU, A. KUMAR, S. ZHAI, Y. CHENG, T. JAVIDI, AND R. S. FERIS, *Fully-adaptive feature sharing in multi-task networks with applications in person attribute classification*, in 2017 IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2017, Honolulu, HI, USA, July 21-26, 2017, IEEE Computer Society, 2017, pp. 1131–1140.

[90] Z. LU, K. DEB, E. D. GOODMAN, W. BANZHAF, AND V. N. BODDETI, *Nsganetv2: Evolutionary multi-objective surrogate-assisted neural architecture search*, in Proceedings of the European Conference on Computer Vision, vol. 12346, 2020, pp. 35–51.

[91] J. MA, Z. ZHAO, X. YI, J. CHEN, L. HONG, AND E. H. CHI, *Modeling task relationships in multi-task learning with multi-gate mixture-of-experts*, in Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining, 2018, pp. 1930–1939.

[92] A. MADRY, A. MAKELOV, L. SCHMIDT, D. TSIPRAS, AND A. VLADU, *Towards deep learning models resistant to adversarial attacks*, in Proceedings of the 6th International Conference on Learning Representations, 2018.

[93] K. MANINIS, I. RADOSAVOVIC, AND I. KOKKINOS, *Attentive single-tasking of multiple tasks*, in Proceedings of IEEE Conference on Computer Vision and Pattern Recognition, 2019, pp. 1851–1860.

[94] A. MAURER, *A chain rule for the expected suprema of gaussian processes*, Theoretical Computer Science, 650 (2016), pp. 109–122.

[95] A. MAURER, M. PONTIL, AND B. ROMERA-PAREDES, *The benefit of multitask representation learning*, Journal of Machine Learning Research, 17 (2016), pp. 1–32.

[96] J. MEI, Y. LI, X. LIAN, X. JIN, L. YANG, A. L. YUILLE, AND J. YANG, *Atomnas: Fine-grained end-to-end neural architecture search*, in Proceedings of the 8th International Conference on Learning Representations, 2020.

[97] I. MISRA, A. SHRIVASTAVA, A. GUPTA, AND M. HEBERT, *Cross-stitch networks for multi-task learning*, in Proceedings of the IEEE conference on computer vision and pattern recognition, 2016, pp. 3994–4003.

[98] M. MOHRI, A. ROSTAMIZADEH, AND A. TALWALKAR, *Foundations of machine learning*, MIT press, 2018.

[99] R. MOTTAGHI, X. CHEN, X. LIU, N. CHO, S. LEE, S. FIDLER, R. URTASUN, AND A. L. YUILLE, *The role of context for object detection and semantic segmentation in the wild*, in Proceedings of IEEE Conference on Computer Vision and Pattern Recognition, 2014, pp. 891–898.

[100] Y. NETZER, T. WANG, A. COATES, A. BISSACCO, B. WU, AND A. Y. NG, *Reading digits in natural images with unsupervised feature learning*, Proceedings of the 24rd Advances in Neural Information Processing Systems, (2011).

[101] M. OQUAB, L. BOTTOU, I. LAPTEV, AND J. SIVIC, *Learning and transferring mid-level image representations using convolutional neural networks*, in Proceedings of the IEEE conference on computer vision and pattern recognition, 2014, pp. 1717–1724.

[102] S. J. PAN AND Q. YANG, *A survey on transfer learning*, IEEE Transactions on knowledge and data engineering, 22 (2009), pp. 1345–1359.

[103] L. PASCAL, P. MICHIARDI, X. BOST, B. HUET, AND M. A. ZULUAGA, *Maximum roaming multi-task learning*, arXiv preprint arXiv:2006.09762, (2020).

[104] Z. PEI, Z. CAO, M. LONG, AND J. WANG, *Multi-adversarial domain adaptation*, in Proceedings of the AAAI Conference on Artificial Intelligence, vol. 32, 2018.

[105] X. PENG, B. USMAN, N. KAUSHIK, J. HOFFMAN, D. WANG, AND K. SAENKO, *Visda: The visual domain adaptation challenge*, arXiv preprint arXiv:1710.06924, (2017).

[106] H. PHAM, M. GUAN, B. ZOPH, Q. LE, AND J. DEAN, *Efficient neural architecture search via parameters sharing*, in International Conference on Machine Learning, PMLR, 2018, pp. 4095–4104.

[107] N. QIAN, *On the momentum term in gradient descent learning algorithms*, Neural networks, 12 (1999), pp. 145–151.

[108] E. REAL, A. AGGARWAL, Y. HUANG, AND Q. V. LE, *Regularized evolution for image classifier architecture search*, in Proceedings of the AAAI Conference on Artificial Intelligence, vol. 33, 2019, pp. 4780–4789.

[109] S. REN, K. HE, R. GIRSHICK, AND J. SUN, *Faster r-cnn: Towards real-time object detection with region proposal networks*, arXiv preprint arXiv:1506.01497, (2015).

[110] L. ROBBIANO, M. R. U. RAHMAN, F. GALASSO, B. CAPUTO, AND F. M. CARLUCCI, *Adversarial branch architecture search for unsupervised domain adaptation*, arXiv preprint arXiv:2102.06679, (2021).

[111] C. ROSENBAUM, T. KLINGER, AND M. RIEMER, *Routing networks: Adaptive selection of non-linear functions for multi-task learning*, in 6th International Conference on Learning Representations, ICLR 2018, Vancouver, BC, Canada, April 30 - May 3, 2018, Conference Track Proceedings, OpenReview.net, 2018.

[112] S. RUDER, *An overview of multi-task learning in deep neural networks*, arXiv preprint arXiv:1706.05098, (2017).

[113] S. RUDER, J. BINGEL, I. AUGENSTEIN, AND A. SØGAARD, *Latent multi-task architecture learning*, in Proceedings of the AAAI Conference on Artificial Intelligence, vol. 33, 2019, pp. 4822–4829.

[114] K. SAENKO, B. KULIS, M. FRITZ, AND T. DARRELL, *Adapting visual category models to new domains*, in European conference on computer vision, Springer, 2010, pp. 213–226.

[115] K. SAITO, K. WATANABE, Y. USHIKU, AND T. HARADA, *Maximum classifier discrepancy for unsupervised domain adaptation*, in Proceedings of the IEEE conference on computer vision and pattern recognition, 2018, pp. 3723–3732.

[116] N. SILBERMAN, D. HOIEM, P. KOHLI, AND R. FERGUS, *Indoor segmentation and support inference from rgbd images*, in European conference on computer vision, Springer, 2012, pp. 746–760.

[117] S. SODHANI, A. ZHANG, AND J. PINEAU, *Multi-task reinforcement learning with context-based representations*, arXiv preprint arXiv:2102.06177, (2021).

[118] T. STANDLEY, A. ZAMIR, D. CHEN, L. GUIBAS, J. MALIK, AND S. SAVARESE, *Which tasks should be learned together in multi-task learning?*, in International Conference on Machine Learning, PMLR, 2020, pp. 9120–9132.

[119] G. STREZOSKI, N. V. NOORD, AND M. WORRING, *Many task learning with task routing*, in Proceedings of the IEEE/CVF International Conference on Computer Vision, 2019, pp. 1375–1384.

[120] B. SUN, J. FENG, AND K. SAENKO, *Return of frustratingly easy domain adaptation*, in Proceedings of the AAAI Conference on Artificial Intelligence, vol. 30, 2016.

[121] B. SUN, J. FENG, AND K. SAENKO, *Correlation alignment for unsupervised domain adaptation*, in Domain Adaptation in Computer Vision Applications, G. Csurka, ed., Advances in Computer Vision and Pattern Recognition, Springer, 2017, pp. 153–171.

[122] B. SUN AND K. SAENKO, *Deep coral: Correlation alignment for deep domain adaptation*, in European conference on computer vision, Springer, 2016, pp. 443–450.

[123] G. SUN, T. PROBST, D. P. PAUDEL, N. POPOVIĆ, M. KANAKIS, J. PATEL, D. DAI, AND L. VAN GOOL, *Task switching network for multi-task learning*, in Proceedings of the IEEE/CVF International Conference on Computer Vision, 2021, pp. 8291–8300.

[124] X. SUN, R. PANDA, R. FERIS, AND K. SAENKO, *Adashare: Learning what to share for efficient deep multi-task learning*, in Proceedings of the 33rd Advances in Neural Information Processing Systems, 2020.

[125] M. TAN, B. CHEN, R. PANG, V. VASUDEVAN, M. SANDLER, A. HOWARD, AND Q. V. LE, *MnasNet: Platform-aware neural architecture search for mobile*, in Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, 2019, pp. 2820–2828.

[126] H. TANG, J. LIU, M. ZHAO, AND X. GONG, *Progressive layered extraction (ple): A novel multi-task learning (mtl) model for personalized recommendations*, in Fourteenth ACM Conference on Recommender Systems, 2020, pp. 269–278.

[127] H. TANG AND Y. LIU, *Deep safe incomplete multi-view clustering: Theorem and algorithm*, in ICML, 2022.

[128] ——, *Deep safe multi-view clustering: Reducing the risk of clustering performance degradation caused by view increase*, in CVPR, 2022.

[129] H. TAO, C. HOU, X. LIU, T. LIU, D. YI, AND J. ZHU, *Reliable multi-view clustering*, in AAAI, 2018.

[130] A. TORRALBA AND A. A. EFROS, *Unbiased look at dataset bias*, in CVPR 2011, IEEE, 2011, pp. 1521–1528.

[131] E. TZENG, J. HOFFMAN, K. SAENKO, AND T. DARRELL, *Adversarial discriminative domain adaptation*, in Proceedings of the IEEE conference on computer vision and pattern recognition, 2017, pp. 7167–7176.

[132] E. TZENG, J. HOFFMAN, N. ZHANG, K. SAENKO, AND T. DARRELL, *Deep domain confusion: Maximizing for domain invariance*, arXiv preprint arXiv:1412.3474, (2014).

[133] S. VANDENHENDE, S. GEORGOULIS, L. V. GOOL, AND B. D. BRABANDERE, *Branched multi-task networks: Deciding what layers to share*, in 31st British Machine Vision Conference 2020, BMVC 2020, Virtual Event, UK, September 7-10, 2020, BMVA Press, 2020.

[134] S. VANDENHENDE, S. GEORGOULIS, W. VAN GANSBEKE, M. PROESMANS, D. DAI, AND L. VAN GOOL, *Multi-task learning for dense prediction tasks: A survey*, IEEE Transactions on Pattern Analysis and Machine Intelligence, (2021).

[135] ——, *Multi-task learning for dense prediction tasks: A survey*, IEEE TPAMI, (2021).

[136] H. VENKATESWARA, J. EUSEBIO, S. CHAKRABORTY, AND S. PANCHANATHAN, *Deep hashing network for unsupervised domain adaptation*, in Proceedings of the IEEE conference on computer vision and pattern recognition, 2017, pp. 5018–5027.

[137] J. WANG, Y. CHEN, W. FENG, H. YU, M. HUANG, AND Q. YANG, *Transfer learning with dynamic distribution adaptation*, ACM Transactions on Intelligent Systems and Technology (TIST), 11 (2020), pp. 1–25.

[138] Z. WANG, Z. DAI, B. PÓCZOS, AND J. CARBONELL, *Characterizing and avoiding negative transfer*, in Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition, 2019, pp. 11293–11302.

[139] Z. WANG, Y. TSVETKOV, O. FIRAT, AND Y. CAO, *Gradient vaccine: Investigating and improving multi-task optimization in massively multilingual models*, in International Conference on Learning Representations, 2020.

[140] R. J. WILLIAMS, *Simple statistical gradient-following algorithms for connectionist reinforcement learning*, Machine learning, 8 (1992), pp. 229–256.

[141] T. WOLF, J. CHAUMOND, L. DEBUT, V. SANH, C. DELANGUE, A. MOI, P. CISTAC, M. FUNTOWICZ, J. DAVISON, S. SHLEIFER, ET AL., *Transformers: State-of-the-art natural language processing*, in Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing: System Demonstrations, 2020, pp. 38–45.

[142] E. WONG, L. RICE, AND J. Z. KOLTER, *Fast is better than free: Revisiting adversarial training*, in Proceedings of the 8th International Conference on Learning Representations, 2020.

[143] Y. XU, L. XIE, X. ZHANG, X. CHEN, G.-J. QI, Q. TIAN, AND H. XIONG, *Pc-darts: Partial channel connections for memory-efficient architecture search*, in Proceedings of the 7th International Conference on Learning Representations, 2019.

[144] Q. YANG, Y. ZHANG, W. DAI, AND S. J. PAN, *Transfer learning*, Cambridge University Press, 2020.

[145] R. YANG, H. XU, Y. WU, AND X. WANG, *Multi-task reinforcement learning with soft modularization*, arXiv preprint arXiv:2003.13661, (2020).

[146] J. YOSINSKI, J. CLUNE, Y. BENGIO, AND H. LIPSON, *How transferable are features in deep neural networks?*, arXiv preprint arXiv:1411.1792, (2014).

[147] T. YU, S. KUMAR, A. GUPTA, S. LEVINE, K. HAUSMAN, AND C. FINN, *Gradient surgery for multi-task learning*, Advances in Neural Information Processing Systems, 33 (2020).

[148] T. YU, D. QUILLEN, Z. HE, R. JULIAN, K. HAUSMAN, C. FINN, AND S. LEVINE, *Meta-world: A benchmark and evaluation for multi-task and meta reinforcement learning*, in Conference on Robot Learning, PMLR, 2020, pp. 1094–1100.

[149] A. R. ZAMIR, A. SAX, W. SHEN, L. J. GUIBAS, J. MALIK, AND S. SAVARESE, *Taskonomy: Disentangling task transfer learning*, in Proceedings of the IEEE conference on computer vision and pattern recognition, 2018, pp. 3712–3722.

[150] W. ZELLINGER, T. GRUBINGER, E. LUGHOFER, T. NATSCHLÄGER, AND S. SAMINGER-PLATZ, *Central moment discrepancy (cmd) for domain-invariant representation learning*, arXiv preprint arXiv:1702.08811, (2017).

[151] W. ZHANG, W. OUYANG, W. LI, AND D. XU, *Collaborative and adversarial network for unsupervised domain adaptation*, in Proceedings of the IEEE conference on computer vision and pattern recognition, 2018, pp. 3801–3809.

[152] Y. ZHANG AND Q. YANG, *A survey on multi-task learning*, IEEE Transactions on Knowledge and Data Engineering, 34 (2022), pp. 5586–5609.

[153] Y. ZHANG AND D.-Y. YEUNG, *A convex formulation for learning task relationships in multi-task learning*, in Proceedings of the Twenty-Sixth Conference on Uncertainty in Artificial Intelligence, 2010, pp. 733–742.

[154] ——, *Multi-task warped gaussian process for personalized age estimation*, in 2010 IEEE Computer Society Conference on Computer Vision and Pattern Recognition, 2010, pp. 2622–2629.

[155] Y. ZHANG, Y. ZHANG, AND W. WANG, *Multi-task learning via generalized tensor trace norm*, in Proceedings of the 27th ACM SIGKDD Conference on Knowledge Discovery & Data Mining, 2021, pp. 2254–2262.

[156] J. ZHAO, W. LV, B. DU, J. YE, L. SUN, AND G. XIONG, *Deep multi-task learning with flexible and compact architecture search*, International Journal of Data Science and Analytics, (2021), pp. 1–13.

[157] S. ZHAO, X. YUE, S. ZHANG, B. LI, H. ZHAO, B. WU, R. KRISHNA, J. E. GONZALEZ, A. L. SANGIOVANNI-VINCENTELLI, S. A. SESHIA, ET AL., *A review of single-source deep unsupervised visual domain adaptation*, IEEE Transactions on Neural Networks and Learning Systems, (2020).

[158] F. ZHUANG, X. CHENG, P. LUO, S. J. PAN, AND Q. HE, *Supervised representation learning: Transfer learning with deep autoencoders*, in Twenty-Fourth International Joint Conference on Artificial Intelligence, 2015.

[159] B. ZOPH AND Q. V. LE, *Neural architecture search with reinforcement learning*, arXiv preprint arXiv:1611.01578, (2016).