# Model and Data Efficiency in Deep Learning

Zejiang Hou

A Dissertation
Presented to the Faculty
of Princeton University
in Candidacy for the Degree
of Doctor of Philosophy

Recommended for Acceptance
by the Department of
Electrical and Computer Engineering
Adviser: Sun-Yuan Kung

September 2022

# Abstract

Deep learning has achieved great and broad breakthroughs in numerous real-world applications with the advancement of larger size models and the explosive growth and availability of data. However, the deep learning models usually have excessive computational and memory cost that are not friendly to practical deployment on mobile or edge devices. Moreover, the deep learning models face challenges in learning and adapting rapidly from only a few examples to solve new tasks. Hence, this thesis proposes techniques to learn computationally efficient model architectures and methods to improve the few-shot learning ability.

We start with subspace analysis methods with application to the feature selection problem. We then extend these methods to deep neural network structural learning (SL), with the objective of reducing the redundant parameters to obtain the optimal down-sized model that can retain or even improve the accuracy. More efficient SL method based on the hybrid pruning-regrowing technique and more generalized SL method that can reduce the model across many more dimensions are also introduced. Going beyond static model designs, we also present dynamic neural network approaches that can adapt the model weights and architectures to different inputs on-the-fly during inference to control the computation efficiency and improve the representation ability. Apart from model efficiency, we also present techniques to train models that can rapidly generalize from a few examples. We propose a few-shot architecture adaption method to customize task-specific model structure for diverse few-shot tasks by meta-learning a task-aware architecture controller. Different from the traditional NAS methods that require a separate search cost on each new task, our method directly generates the task-specific model structure from the dataset in GPU minutes after a one-time meta-training cost. Finally, we propose a cross-modality self-supervised learning framework by masked image pretraining on language assisted representations. The resulting models produce high quality transferable representations that advance the accuracy on numerous computer vision tasks and demonstrate strong robustness to adversarial/out-of-distribution samples. Moreover, the resulting models are amenable to structural learning for greater computation efficiency gains and to low-resource task adaptation for better data efficiency.

# Acknowledgements

It is a privilege to attend the PhD program at Princeton University. I am eternally grateful to my parents, who have given the lifetime of supports to let me enjoy the study at Princeton. It is a genuine pleasure to work with professor Sun-Yuan Kung, to whom I owe a deep sense of gratitude. The generous guidance on every stage of my research and the sharing of considerable knowledge and wisdom from professor Kung has been invaluable throughout my PhD. Prof. Kung deserves much credit for the fundamental ideas and research directions I have been able to explore in this thesis.

It was a fortune to have professor S. C. Chan as my undergraduate advisor, who gave me the opportunity to start my first research project and made great efforts to help me make it to the prestigious PhD program at Princeton.

The road towards PhD is tough filled with bushes and thorns. I am especially thankful to have Wei Dai as my girlfriend, who has always been with me to make my life brighter.

Finally, I want to thank our collaborators Yen-Kuang Chen, Fei Sun, Minghai Qin, Sicheng Li from Alibaba Group DAMO Academy, Julian Salazar, George Polovets from Amazon AWS AI for making parts of my research possible. They were great mentors and research partners giving me a lot of inspirations. I was lucky to work with someone as insightful as them in my pursuit of research.

# Contents

# List of Figures

# List of Tables

# List of Algorithms

# Chapter 1

# Introduction

Deep learning (DL) has achieved great and broad breakthroughs in many artificial intelligence (AI) applications such as vision data processing, natural language processing, speech and audio processing among many others. With the availability of large-scale datasets, the emergence of powerful computing devices, and the advancement of larger size models, AI has achieved super human performance in many fields. To name a few, the residual network (ResNet) [85] obtained higher classification accuracy than humans on ImageNet [41] challenge. The DeBERTa [86] model exceeded human baseline on the natural understanding benchmark SuperGLUE [242]. The prosperity of DL also catalyzes the development of intelligent tools that have transformed many aspects of daily life, such as voice assistants, search engines, autonomous driving cars, and industrial robots.

The explosive growth of smart phones and IoT devices give rise to new opportunities of combining AI with these edge devices. By running deep learning models on these edge devices, we can directly perform data analysis near the sensor without the need for a connection to the clouding computing devices. This could bring better privacy and expand the scope of AI to applications such as smart manufacturing, personalized healthcare, precision agriculture, automated retailing. However, the edge devices are highly resource constrained in terms of the memory, storage, and computing capacity, making it hard to deploy many of the deep learning models or run these models for real-time applications. This prompts some urgency on designing computation and memory efficient deep learning models. The pursuit on optimal model architectures remains largely an art of trial-and-error. Although manually designed efficient models [210, 114, 293] introduce many efficient building operators such as depthwise separable convolution or channel shuffling, the overall architectures are far from optimal. To automate the design of efficient models, neural architecture search (NAS) methods

automatically search for the pareto optimal architectures in a pre-defined search space. However, NAS methods suffer from prohibitive computational demand due to certain search algorithms [305, 306] or the excessive memory cost when training large stochastic supernets [161, 74]. Closely related to NAS, model compression approaches reduce the redundant parameters and structures to achieve memory and computation savings while maximally maintaining or even improving the accuracy. These methods can be regarded as architecture search in a restrictive space. For example, channel pruning can be regarded as searching the optimal width in each layer of the model while fixing the depth, kernel size, and operations fixed. Various pruning criteria have been proposed to evaluate the importance of different parameters or structures and remove the redundancy to obtain the down-sized models. Majority of methods use either weight information [141, 88, 270, 40, 87, 90] or activation statistics [91, 173, 303], and rarely consider the correlation with the model output or teacher values. Thus, the reduced models may suffer from sub-optimal quality. Moreover, most previous works [141, 91, 173, 270, 303, 40, 90, 276, 182, 70, 273, 103, 101, 175, 292] adopt a progressive pruning-training pipeline: pre-training a large model to convergence, pruning a few unimportant parameters or structures by the pre-defined criterion, and finetuning the pruned model to restore accuracy. Substantial training cost is consumed on parameters that will be pruned later, making the pruning process very inefficient. Such approaches are restricted to pruning process only, which lacks the flexibility of growing back weights or structures that are considered unimportant early in the training but become significant later.

Another challenge of deep learning models is the poor generalization ability on limited number of examples with supervised information. This is very different from human intelligence that is capable to learn new tasks rapidly given a few examples. Bridging this gap between AI and humans is an important direction. There are also many real scenarios where large-scale training examples with supervised information are hard or impossible to acquire due to privacy, safety or ethic issues. For example, in drug discovery, scientists need to investigate the properties of different molecules to develop useful new drugs. Considering the possible toxicity, they do not have many records of clinic trials of these molecules. To this end, few-shot learning (FSL) [58] has become an important machine learning paradigm to learn models on low-resource tasks with limited labeled examples. Many prior works in FSL focus on designing more suitable distance metrics that better fit the data distributions [238, 216, 272, 225, 97, 12, 288, 214]. However, these methods are only applicable to few-shot classification tasks. Optimization-based meta-learning approaches [59, 7, 209, 224] are model agnostic, and can be extended to different tasks in different domains and modalities. However, current meta-learning methods assumes the few-shot tasks are similar to each other from a fixed

task distribution and forcibly apply the same hypothesis space to diverse tasks. Moreover, these methods are prone to overfitting on the few-shot training examples [110]. Although using very shallow neural networks or constraining the gradient steps can alleviate the overfitting problem, these strategies also limit the task performance. Transfer learning is also widely applied in FSL to use model trained on source domain/task with abundant data as prior knowledge when learning on low-resource tasks. Although learning high quality transferable representations is promising, most previous works [45, 33, 232] apply supervised objectives and only acquire in-task knowledge. The learned representations have good clustering property on seen classes in training, but may be sub-optimal for unseen and novel classes in testing [192].

In a similar vein to improve the training convergence and generalization on small datasets with scare training labels, large-scale pretraining has been widely applied to produce general deep learning models before adapting to different domains and tasks. The superb model qualities heavily depend on the large-scale labeled datasets for supervised learning. Annotation of large-scale datasets are time-consuming and expensive. For example, the most widely used image datasets, ImageNet, contains 15M labeled images from 22K classes. It took 49K Amazon Turk workers years to annotate a dataset at such a large scale. In order to reduce the reliance on large-scale labeled datasets, self-supervised learning methods [21, 235, 57, 142, 66, 286, 23, 36, 31, 81, 35, 68, 22, 11, 80, 262, 255] are gaining popularity, which learn high quality transferable representations from unlabeled data. The model learns useful representations by solving a pretext task and the the supervisory signal is generated from the data itself by leveraging its structure. However, most of existing self-supervised learning methods learn features on data from one modality only. Witnessing the substantial growth of the web data that usually have images along with captions or tags, current self-supervised learning methods cannot leverage the cross-modality information to improve the representation quality.

## 1.1 Summary of contributions

As a whole this thesis provides various approaches dedicated to efficient deep learning. In early chapters, we unify the discriminant-based and correlation-based subspace analysis methods for evaluating the parameter or structure importance when learning the optimal down-size model architecture [99, 102, 100]. We propose a novel differential discriminant metric to remove the deleterious features with minimum impact to the feature-maps discriminant power in local layers, which can be used as the gradient information for updating the model structure [133]. To improve the efficiency of structural learning, we also propose a channel exploration (CHEX) methodology [109].

CHEX starts from a randomly initialized sub-structure and automatically adapt to more optimal structure during training. As opposed to pruning-only strategy, we propose to repeatedly prune and regrow the channels throughout the training process, which reduces the risk of pruning important channels prematurely. From intra-layer's aspect, we tackle the channel pruning problem via a well-known column subset selection (CSS) formulation. From inter-layer's aspect, our regrowing stages open a path for dynamically re-allocating the number of channels across all the layers under a global channel sparsity constraint. CHEX does not need a well trained full model as the starting point. All the exploration process is done in a single training pass from scratch, effectively reducing the training cost. Considering the advent of more and more complicated models that contain heterogeneous building structures, we also propose more generalized omni-dimensional structural learning [107] with joint optimization to search a jointly optimal policy across many different dimensions.

Although various proposed structural learning methods can produce more efficient and accurate models, they all perform the inference in a static manner: the architecture and the model parameters are fixed once trained. Complementing these methods are our dynamic neural network designs [104, 105] that adapt both the model structures and parameters in both sample-wise and spatial-wise dependent manner in inference. The dynamic architectures can control the computation efficiency at test time by selectively activating part of the network conditioned on the input. The dynamic parameters enlarge the parameter space and improve the model representation power.

Apart from model efficiency, we also aim to improve the generalization of deep learning models on few-shot or low-resource tasks. Based on the error decomposition analysis, two approaches are proposed to learn data-efficient models. We first propose a few-shot architecture adaption method [111] to customize task-specific model space for diverse few-shot tasks by by meta-learning a task-aware architecture controller. Different from the traditional NAS methods that require a separate search cost on each new task, our method can directly generate task-specific model architectures from the dataset in GPU minutes after a one-time meta-training cost. In addition, we combine semi-supervised to FSL to obtain more reliable empirical risk minimizer from limited training data, where we propose an unsupervised dependency maximization loss to exploit unlabeled data [108].

Finally in the penultimate chapter, we propose a cross-modality self-supervised learning framework. To harness the best from both generation-based visual representation learning and the language-image pretraining, we propose a two-stage masked image pretraining on language assisted representations, where the first stage learns visual features with language supervision and the second stage use these semantic-rich contextualized features as targets for a more challenging masked image training process. The resulting models largely improve on both the language-image pretraining alone and generation-

based self-supervised learning alone. Moreover, our models demonstrate strong transferability, label efficiency, and robustness to adversaries and out-of-distribution samples. For example, using only 10% labeled ImageNet data, our model achieves the current best 81.9% top-1 accuracy in the semi-supervised classification on ImageNet, based on the public leaderboard[1].

## 1.2 Thesis outline

The remainder of this thesis is organized as follows. Chapter 2 presents the fundamental subspace analysis techniques that can reduce the unwanted redundancy while maximizing the feature-output correlation. A discriminant component analysis method and a discriminant information metric, together with their nonlinear extensions using kernel methods, are introduced, and we evaluate their effectiveness of reducing the redundancy and preserving the relevancy on a input node pruning problem. Chapter 3 extends the subspace analysis to the deep neural network structural learning, and presents a framework to jointly learn the optimal model structure and parameters with the objective of reducing the model redundancy and computation while retaining or even improving the performance. We unify the views of discriminant-based and correlation-based subspace analysis and propose a differential discriminant metric, which can be used as the gradient information for updating the model structure. Chapter 4 continues the topic of structural learning and presents more efficient and generalized approaches. To achieve more efficient structural learning process, a channel exploration method is proposed, which start from a randomly initialized sub-structure and progressively adapt to more optimal structure, by reducing the redundant parameters while growing more orthogonal weights based on our subspace analysis metrics. To deal with more complicated architectures such as the transformers, an omni-dimensional structural learning with joint optimization is proposed to search more general policy covering many different dimensions. Chapter 5 presents our dynamic neural network approaches that greatly improve the representation power and computation efficiency of the deep learning models. We propose both dynamic model parameters and dynamic architectures design strategies, and cover both sample-dependent and spatial-dependent control functions in our dynamic neural networks. Chapter 6 presents our approaches make the deep learning models generalize from very few-labeled examples. Based on the error decomposition analysis, we propose few-shot architecture adaptation and semi-supervised few-shot learning. Chapter 7 presents our cross-modality self-supervised learning framework to learn models with higher label efficiency, transferability, and robustness. Chapter 8 concludes the thesis and discuss the potential future works.

[1] `https://paperswithcode.com/sota/semi-supervised-image-classification-on-2`

## 1.3 Notation

We define some basic notations. We denote matrices by bold capitalized letters, vectors by bold small letters and scalars by regular letters. For matrix subscript, the first one is to index a row and the second is to index a column in the matrix. We use $\mathbf{X} = [\mathbf{x}_1, ..., \mathbf{x}_N]$ to refer to the data matrix of $N$ examples $\{\mathbf{x}_i\}_{i=1}^N$, where each column represents on data point. To achieve zero mean, we use $\mathbf{C} = \mathbf{I} - (1/N\mathbf{1}\mathbf{1}^T)$ to represent a centering matrix and the center the data matrix by $\bar{\mathbf{X}} = \mathbf{X}\mathbf{C}$. The data scatter matrix is represented by $\mathbf{S} = \mathbf{X}\mathbf{C}\mathbf{X}^T$. We use $k(\cdot, \cdot)$ and $\mathbf{K}$ to represent the kernel function and the kernel matrix, where $\mathbf{K}_{i,j} = k(\mathbf{x}_i, \mathbf{x}_j)$. We use $\|\cdot\|_2$ to denote the $\ell_2$ norm of a vector or the spectral norm of a matrix, and $\|\cdot\|_F$ to denote matrix Frobenius norm. $\|\cdot\|_p$ refers to the $\ell_p$ norm of a vector. For matrix operations, we use $\mathbf{A}^{-1}, \mathbf{A}^{\dagger}, \mathrm{tr}\,(\mathbf{A}), \mathbf{A} \odot \mathbf{B}$ to represent inverse, pseudo-inverse, trace, and hadamard product, respectively. The deep learning models (or networks) consist of a graph of parameterized layers that implement a nonlinear function $f$. In a general supervised setting with a training set of input examples $\mathbf{x} \in \mathcal{X}$ and labels $\mathbf{y} \in \mathcal{Y}$, we aim to learn the function $f : \mathbf{X} \mapsto \mathbf{Y}$ parameterized by $\mathbf{\Theta}$ such that the prediction $\hat{y} = f(\mathbf{x}; \mathbf{\Theta})$ is close to $\mathbf{y}$. The quality of the prediction is evaluated using a loss function $\ell : \mathcal{Y} \times \mathcal{Y} \mapsto \mathbb{R}$, which gives the error $\ell(\hat{\mathbf{y}}, \mathbf{y})$. A forward pass of the model is to transform the input layer by layer to produce the output by the function $f(\mathbf{x}; (\Theta)$, while the backward pass is to propagate the error from last layer in the reverse direction to compute gradients by the chain rule.

# Chapter 2

# Subspace Learning

## 2.1 Prologue

This chapter serves as a preliminary on some fundamental techniques for dimension reduction and subspace projection. Earlier methods in the direction rely on random or unsupervised projections to preserve pairwise distances between data points or to maximize the variance of the projected data points. Although these projections maintain the performance on certain machine learning models, they are not able to find the subspace that has the best correlation with the labels. Therefore, a supervised subspace analysis method called the discriminant component analysis (DCA) is introduced. The nonlinear extension of the DCA method using kernel method is also introduced. Finally, we apply this subspace analysis method to the input node pruning problem to pave our way towards deep neural network structural learning in later chapters.

## 2.2 Preliminary on linear discriminant analysis

The Linear Discriminant Analysis (LDA) projects $L$ classes of data onto a subspace with $L-1$ directions so that the ratio between the between-class separation and the within-class variation is maximized. The LDA method was originally developed to find the best projection direction that separates two classes of data in binary classification tasks, and it was later extended to find the best subspace in multi-class settings. The optimal projection directions in LDA can be found by solving the following optimization problem:

$$\max_{\mathbf{F}} \frac{|\mathbf{F}^T \mathbf{S}_B \mathbf{F}|}{|\mathbf{F}^T \mathbf{S}_W \mathbf{F}|}, \tag{2.1}$$

where $\mathbf{S}_W, \mathbf{S}_B$ represents the within-class and between-class scatter matrices, respectively. Suppose we have a training dataset with $L$ classes $\{\mathbf{x}_i, y_i\}_{i=1}^N$ these scatter matrices are computed by:

$$\mathbf{S}_W = \sum_{l=1}^{L} \sum_{y_i=l} (\mathbf{x}_i - \boldsymbol{\mu}_l)(\mathbf{x}_i - \boldsymbol{\mu}_l)^T, \tag{2.2}$$
$$\mathbf{S}_B = \sum_{l=1}^{L} N_l (\boldsymbol{\mu} - \boldsymbol{\mu}_l)(\boldsymbol{\mu} - \boldsymbol{\mu}_l)^T,$$

where $\boldsymbol{\mu} = 1/N \sum_{i=1}^{N} \mathbf{x}_i$ denotes the mean vector of the whole dataset, $\boldsymbol{\mu}_l = 1/N_l \sum_{y_i=1} \mathbf{x}_i$ represents the mean vectors of the data in $l$-th class. Because the between-class scatter $\mathbf{S}_B$ can have a rank at most. $L-1$, LDA can obtain at most $L-1$ directions for subspace projection.

## 2.3 Discriminant component analysis

The previous discriminant analysis have limitations in that the transformation invariant property does not hold. We introduce a new subspace projection method, called discriminant component analysis (DCA). DCA differs from LDA in two important aspects: (1) DCA adds an orthogonality constraint among projection directions to minimize correlation; (2) DCA represents the overall objective as the sum of signal-to-noise ratios across all directions instead of a ratio of determinants. Moreover, the DCA method enjoys the transformation invariant property and monotonic non-decreasing property with respect to the number of feature dimensions.

We start introducing how to find the single best projection direction $\mathbf{f}^T\mathbf{x}$. If there were two classes, a straightforward measure of the class separation is the squared distance between two centroids $(\mathbf{f}^T\boldsymbol{\mu}_1 - \mathbf{f}^T\boldsymbol{\mu}_2)^2$. For multi-class dataset, the between-class separation becomes the average squared distance between class centroid and the mean of whole dataset:

$$\sum_{l=1}^{L} \frac{N_l}{N} (\mathbf{f}^T\boldsymbol{\mu}_l - \mathbf{f}^T\boldsymbol{\mu})^2 = 1/N \cdot \mathbf{f}^T\mathbf{S}_B\mathbf{f}. \tag{2.3}$$

This term is usually regarded as the signal term when analyzing the feature discriminant power or linear separability. When the class centroids are far apart from each other, the signal will be higher. The within-class variation is measured by averaging the sample variance across all classes, which is usually regarded as the noise term:

$$\sum_{l=1}^{L} \sum_{y_i=l} \frac{N_l}{N} (\mathbf{f}^T\mathbf{x}_i - \mathbf{f}^T\boldsymbol{\mu}_l)^2 = 1/N \cdot \mathbf{f}^T\mathbf{S}_W\mathbf{f}. \tag{2.4}$$

The best projection direction maximizes the of between-class separation while minimizing the within-class variation based on the signal-to-noise ratio $\frac{\mathbf{f}^T\mathbf{S}_B\mathbf{f}}{\mathbf{f}^T\mathbf{S}_W\mathbf{f}}$. Moreover, from the equality $\bar{\mathbf{S}} = \sum_{i=1}^{N}(\mathbf{x}_i - \boldsymbol{\mu})(\mathbf{x}_i - \boldsymbol{\mu})^T = \mathbf{S}_B + \mathbf{S}_W$, the final objective of a single direction DCA is given by:

$$\mathbf{f}^* = \underset{\mathbf{f}}{\operatorname{argmax}} \frac{\mathbf{f}^T\mathbf{S}_B\mathbf{f}}{\mathbf{f}^T\mathbf{S}_W\mathbf{f}} = \underset{\mathbf{w}}{\operatorname{argmax}} \frac{\mathbf{f}^T\mathbf{S}_B\mathbf{f}}{\mathbf{f}^T\bar{\mathbf{S}}\mathbf{f}}. \tag{2.5}$$

For multi-class problems, DCA maximizes the sum of the signal-to-noise ratios across all possible directions under an orthogonality constraint to avoid overlapping among different directions. To ensure different directions are uncorrelated, we can impose $\mathbf{f}_i^T\bar{\mathbf{S}}\mathbf{f}_j = 0$ for $i \neq j$. To also normalizes the scales, we impose the constraint $\mathbf{F}^T\bar{\mathbf{S}}\mathbf{F} = \mathbf{I}$, where $\mathbf{F} = [\mathbf{f}_1, ..., \mathbf{f}_r]$ for a total of $r$ directions. The final DCA objective is given by:

$$\max_{\mathbf{F}:\mathbf{F}^T\bar{\mathbf{S}}\mathbf{F}=\mathbf{I}} \sum_{i=1}^{r} \frac{\mathbf{f}_i^T\mathbf{S}_B\mathbf{f}_i}{\mathbf{f}_i^T\bar{\mathbf{S}}\mathbf{f}_i} = \max_{\mathbf{F}:\mathbf{F}^T\bar{\mathbf{S}}\mathbf{F}=\mathbf{I}} \operatorname{tr}(\mathbf{F}^T\mathbf{S}_B\mathbf{F}). \tag{2.6}$$

In practice, we usually add a ridge parameter to the scatter matrix $\bar{\mathbf{S}}$ to reduce the variance of the obtained solution on small number of training data. The DCA optimization objective becomes:

$$\max_{\mathbf{F}:\mathbf{F}^T(\bar{\mathbf{S}}+\rho\mathbf{I})\mathbf{F}=\mathbf{I}} \operatorname{tr}(\mathbf{F}^T\mathbf{S}_B\mathbf{F}), \tag{2.7}$$

whose optimal solution is given by the principal eigenvectors of the matrix $(\bar{\mathbf{S}} + \rho\mathbf{I})^{-1}\mathbf{S}_B$. When $\rho = 0$, the optimal subspace spanned by the columns of the DCA solution is also optimal for LDA, but the reverse is not always true due to the orthogonality constraint in DCA is not included in LDA. Moreover, in DCA solutions, the eigenvectors corresponding to the largest eigenvalues of $(\bar{\mathbf{S}} + \rho\mathbf{I})^{-1}\mathbf{S}_B$ achieve the highest signal-to-noise ratios.

### 2.3.1 Discriminant information

The maximal value of the DCA objective is the quality we refer to the discriminant information (DI), which provides a multi-class extension of the Fisher discriminant ratio (FDR), which is usually

applied to measure the linear separability of features.

$$\begin{aligned}
\text{DI} &= \max_{\mathbf{F}:\mathbf{F}^T(\bar{\mathbf{S}}+\rho\mathbf{I})\mathbf{F}=\mathbf{I}} \operatorname{tr}(\mathbf{F}^T\mathbf{S}_B\mathbf{F}) \\
&= \max_{\hat{\mathbf{F}}:\hat{\mathbf{F}}^T\hat{\mathbf{F}}=\mathbf{I}} \operatorname{tr}\left(\hat{\mathbf{F}}^T(\bar{\mathbf{S}}+\rho\mathbf{I})^{-1/2}\mathbf{S}_B(\bar{\mathbf{S}}+\rho\mathbf{I})^{-1/2}\hat{\mathbf{F}}\right) \\
&= \operatorname{tr}\left((\bar{\mathbf{S}}+\rho\mathbf{I})^{-1/2}\mathbf{S}_B(\bar{\mathbf{S}}+\rho\mathbf{I})^{-1/2}\right) \\
&= \operatorname{tr}\left((\bar{\mathbf{S}}+\rho\mathbf{I})^{-1}\mathbf{S}_B\right), \qquad (2.8)
\end{aligned}$$

where the second equality is from changing the variables $\mathbf{F} = (\bar{\mathbf{S}} + \rho\mathbf{I})^{-1/2}\hat{\mathbf{F}}$, and the third equality is from the solution of the generalized Rayleigh quotient problem.

### 2.3.2 Equivalency between DI and ridge LSE

Although the DCA method is originally developed for classification problems, it is also closely related to the ridge LSE, which means the DCA method can encompasses the regression applications as well. We first present an equivalence between DI and ridge LSE in classification setting. Let $\mathbf{X}$ be the input data matrix and $\mathbf{Y}$ be the label matrix. For both class balanced and unbalanced scenarios, when $\mathbf{Y}$ contains the weighted one-hot encoding (the label is replaced by $\sqrt{N_l}^{-1}$ instead of "1" as in the one-hot encoding) and each row of $\mathbf{Y}$ has zero-mean, then the between-class scatter matrix becomes $\mathbf{S}_B = \mathbf{X}\mathbf{Y}^T\mathbf{Y}\mathbf{X}^T$. On the other hand, the ridge LSE objective is given by:

$$\min_{\mathbf{W},\mathbf{b}} \ \|\mathbf{W}^T\mathbf{X} + \mathbf{b}\mathbf{1}^T - \mathbf{Y}\|_F^2 + \rho\|\mathbf{W}\|_F^2. \qquad (2.9)$$

The minimal objective value of the ridge LSE problem is given by:

$$\text{RidgeLSE} = \operatorname{tr}(\mathbf{Y}^T\mathbf{Y}) - \operatorname{tr}\left((\bar{\mathbf{X}}\bar{\mathbf{X}}^T + \rho\mathbf{I})^{-1}\mathbf{X}\mathbf{Y}^T\mathbf{Y}\mathbf{X}^T\right). \qquad (2.10)$$

Combining Eq.(2.8) and Eq.(2.10), we have

$$\text{RidgeLSE} = C - \text{DI} \qquad (2.11)$$

where the constant $C$ is equal to $\operatorname{tr}(\mathbf{Y}^T\mathbf{Y}) = L - 1$ and $L$ denotes the number of different classes. This equivalence and the re-defined between-class scatter matrix ($\mathbf{S}_B = \mathbf{X}\mathbf{Y}^T\mathbf{Y}\mathbf{X}^T$) extend the application the DCA method and the DI metric to arbitrary type of label in classification and regression problems.

## 2.4 Kernelized DCA and DI

In this section, we introduce the non-linear extensions of the DCA method using kernels methods. Let $\mathcal{X}$ be a set from which data is drawn. Denote a set of examples drawn from $\mathcal{X}$ by $\mathcal{S} = \{\mathbf{x}_i\}_{i=1}^N$. We consider a pairwise similarity function $k(\mathbf{x}_i, \mathbf{x}_j)$ for any pairs of data in the set $\mathcal{S}$. If the $N \times N$ kernel matrix $\mathbf{K}$ ($K_{ij} = k(\mathbf{x}_i, \mathbf{x}_j)$) is symmetric positive semi-definite for all $\mathcal{S} \in \mathcal{X}$, then there exists a mapping $\phi : \mathcal{X} \to \mathcal{H}$ and a Hibert space $\mathcal{H}$ such that $k(\mathbf{x}, \mathbf{x}') = \langle \phi(\mathbf{x}), \phi(\mathbf{x}') \rangle_{\mathcal{H}}$. Any pairwise similarity function that guarantees a symmetric positive semi-definite kernel matrix can define an non-linear mapping to a Hilbert space.

Before we give the formulations for the kernel discriminant component analysis, we shall introduce a learning space property (LSP) [132] which is important to the derivation of the kernel trick so that we do not need to compute the kernel feature mappings explicitly. The LSP says that the optimal projection matrix for the DCA method lies in the span of the data matrix $\mathbf{X}$, i.e., $\mathbf{F}^* = \mathbf{X}\mathbf{U}$ for some matrix $\mathbf{U}$. We define $\mathbf{\Phi} = [\phi(\mathbf{x}_1), ..., \phi(\mathbf{x}_N)]$ as the matrix after mapping the data into the kernel feature space and $\bar{\mathbf{\Phi}} = \mathbf{\Phi}\mathbf{\Gamma}$ be the centered kernel data matrix, where $\mathbf{\Gamma} = \mathbf{I}_N - \frac{1}{N}\mathbf{1}_N\mathbf{1}_N^T$ is the centering matrix. The centered kernel matrix is denoted by $\bar{\mathbf{K}} = \bar{\mathbf{\Phi}}^T\bar{\mathbf{\Phi}}$. Based on LSP, we can reformulate the following terms:

$$\mathbf{F}^T\bar{\mathbf{S}}\mathbf{F} = \mathbf{U}^T\bar{\mathbf{\Phi}}^T\bar{\mathbf{\Phi}}\bar{\mathbf{\Phi}}^T\bar{\mathbf{\Phi}}\mathbf{U} = \mathbf{U}^T\bar{\mathbf{K}}^2\mathbf{U}, \tag{2.12}$$

$$\mathbf{F}^T\mathbf{S}_B\mathbf{F} = \mathbf{U}^T\bar{\mathbf{\Phi}}^T\bar{\mathbf{\Phi}}\mathbf{Y}^T\mathbf{Y}\bar{\mathbf{\Phi}}^T\bar{\mathbf{\Phi}}\mathbf{U} = \mathbf{U}^T\bar{\mathbf{K}}\mathbf{Y}^T\mathbf{Y}\bar{\mathbf{K}}\mathbf{U} = \mathbf{U}^T\mathbf{K}_B\mathbf{U}. \tag{2.13}$$

Bu plugging (2.12) and (2.13), the kernel discriminant component analysis (KDCA) problem is finally defined as:

$$\max_{\mathbf{U}:\mathbf{U}^T(\bar{\mathbf{K}}^2+\rho\bar{\mathbf{K}})\mathbf{U}=\mathbf{I}} \ \mathrm{tr}\left(\mathbf{U}^T\mathbf{K}_B\mathbf{U}\right). \tag{2.14}$$

The maximal objective value of problem (2.14) is the kernel discriminant information (KDI) metric:

$$\mathrm{KDI} = \mathrm{tr}\left((\bar{\mathbf{K}}^2 + \rho\bar{\mathbf{K}})^{-1}\mathbf{K}_B\right). \tag{2.15}$$

KDI is a closed-form metric to evaluate the linear separability of the features after kernel mapping. In the next section, we will show that KDI can serve as a general feature-label dependency measure, and we use it as an optimization objective for selecting input features that best predict the labels.

## 2.5 Application to input node pruning

### 2.5.1 Problem setting

Let $\mathcal{X} \subset \mathbb{R}^d$ be the domain of the input vector $\mathbf{x}$ and $\mathcal{Y} \subset \mathbb{R}^q$ be the domain of the target vector $\mathbf{y}$. The input node pruning problem is to select $m$ features ($m \ll d$) of the input vector that best predict $\mathbf{y}$. Denote the data matrix and label matrix of $N$ samples by $\mathbf{X} = [\mathbf{x}_1, \mathbf{x}_2, ..., \mathbf{x}_N] \in \mathbb{R}^{d \times N}$ and $\mathbf{Y} = [\mathbf{y}_1, \mathbf{y}_2, ..., \mathbf{y}_N] \in \mathbb{R}^{q \times N}$, respectively.

### 2.5.2 Relation to subspace learning

Subspace learning aims to find the optimal projections $\mathbf{F} \in \mathbb{R}^{d \times m}$ ($m \ll d$) to transform the original data into a lower dimensional subspace $\mathbf{F}^T\mathbf{X} \in \mathbb{R}^{m \times N}$. Input node pruning, on the other hand, finds an optimal feature subset such that a metric $\mathcal{Q}$ is optimized. Suppose $m$ out of $d$ features of the original data $\mathbf{X}$ are selected, a general supervised input node pruning problem is given by:

$$\underset{\mathbf{w} \in \{0,1\}^d, \|\mathbf{w}\|_0 = m}{\text{Optimize}} \quad \mathcal{Q}\big(\texttt{diag}(\mathbf{w})^T\mathbf{X}, \mathbf{y}\big), \tag{2.16}$$

where $\texttt{diag}()$ means diagonalizing a vector, the selection vector has elements $w_i$ being either 0 ($i$-th feature is pruned) or 1 ($i$-th feature is selected), and at most $m$ features are selected. Projecting the data $\mathbf{X}$ by $\texttt{diag}(\mathbf{w})$ places an indicator on each input node, and the magnitude reflects the feature's importance.

### 2.5.3 KDI-based input node pruning

The input node pruning problem optimizes a selection matrix such that a properly chosen criterion is optimized. The KDI metric can measure the inter-class separability of the data in the kernel induced feature space. Therefore, our optimization objective is to search a feature subset with the highest KDI metric so that the features can best predict the labels. Integrating input node pruning to the KDCA problem gives our objective:

$$\begin{aligned} \max_{\mathbf{w},\mathbf{F}} \quad & \text{tr}(\mathbf{F}^T\mathbf{K}_B(\mathbf{w})\mathbf{F}), \\ \text{s.t.} \quad & \mathbf{F}^T[\bar{\mathbf{K}}(\mathbf{w})^2 + \rho\bar{\mathbf{K}}(\mathbf{w})]\mathbf{F} = \mathbf{I}, \\ & \mathbf{w} \in \{0,1\}^d, \\ & \|\mathbf{w}\|_0 = m, \end{aligned} \tag{2.17}$$

where $\bar{\mathbf{K}}(\mathbf{w})$ denotes the centered kernel matrix evaluated on the features weighted by the selection vector $\mathbf{K}(\mathbf{w}) = [k_{\mathbf{w}}(\mathbf{x}, \mathbf{y})] = [\langle \phi(\mathbf{w} \odot \mathbf{x}), \phi(\mathbf{w} \odot \mathbf{y}) \rangle]$. Similarly, $\mathbf{K}_B(\mathbf{w})$ denotes the kernel between-class matrix on features weighted by the selection vector.

A block-wise algorithm can be adopted to solve the maximization problem, which iterates between solving the KDCA problem w.r.t. the projection matrix $\mathbf{F}$ and estimating the selection vector $\mathbf{w}$ to maximize the KDI metric. The sub-problem w.r.t. $\mathbf{F}$ has analytical solution $\mathbf{F}^* = [\bar{\mathbf{K}}^2 + \rho\bar{\mathbf{K}}]^{-1}\mathbf{\Delta}$ Substituting $\mathbf{F}^*$ into Eq.(2.17) leads to a simpler problem w.r.t. $\mathbf{w}$ only:

$$\begin{aligned} \max_{\mathbf{w}} \quad & \mathrm{tr}\left([\bar{\mathbf{K}}(\mathbf{w})^2 + \rho\bar{\mathbf{K}}(\mathbf{w})]^{-1}\mathbf{K}_B(\mathbf{w})\right), \\ \text{s.t.} \quad & \mathbf{w} \in \{0,1\}^d, \\ & \|\mathbf{w}\|_0 = m, \end{aligned} \tag{2.18}$$

where the optimization metric is the kernelized discriminant information (KDI).

This binary optimization problem is hard to optimize directly, and it is computationally intractable to perform exhaustive search on very high dimensional features. We perform continuous relaxation to the problem and add an extra bi-model regularization term [184] The domain of $\mathbf{w}$ is relaxed into a continuous range $[0,1]^d$. To learn binary values, we apply a regularization term $\mathbf{w}^T(1-\mathbf{w})$. This extra term is minimized when $\mathbf{w} \in \{0,1\}^d$. The relaxed problem is formulated as:

$$\begin{aligned} \max_{\mathbf{w}} \quad & \mathrm{tr}\left([\bar{\mathbf{K}}(\mathbf{w})^2 + \rho\bar{\mathbf{K}}(\mathbf{w})]^{-1}\mathbf{K}_B(\mathbf{w})\right) - \lambda\mathbf{w}^T(1-\mathbf{w}), \\ \text{s.t.} \quad & \mathbf{w} \in [0,1]^d, \\ & \mathbf{1}^T\mathbf{w} = m. \end{aligned} \tag{2.19}$$

We solve the problem by projected gradient ascent. Due to the non-concave nature of the objective arising from the kernel function and bi-model regularization, the quality of the obtained solution requires a good initialization.. We found that assigning uniform weights to the features at initialization works the best. Therefore, the vector $\mathbf{w}$ is initialized by $(m/d) \cdot \mathbf{1}$. After a solution to the relaxed problem is obtained, the optimal input features corresponds to the top $m$ weights in $\mathbf{w}$. The pseudo-code of the proposed KDI-based input node pruning algorithm is provided in Algorithm 1. A brief complexity analysis of the proposed approach shows that it requires $\mathcal{O}(N^3 + N^2 d)$ to evaluate KDI metric, where the first term takes into account the inversion of an $N \times N$ matrix and the second term is the complexity of computing the kernel matrix. The second term dominates on the

**Algorithm 1:** KDI Input Node Pruning.

**Input**: training data $\{\mathbf{x}_i, \mathbf{y}_i\}_{i=1}^N$, desired number of input nodes $m$, hyper-parameters $\rho$ and $\lambda$, learning rate $\eta$, iterations $T$ ;

**Output**: optimal selection vector $\mathbf{w}^*$ ;

Initialize $\mathbf{w}$ by $(m/d) \cdot \mathbf{1}$ ;

**while** *not converge and* $T$ *is not reached* **do**

  Compute the gradient of the KDI metric w.r.t $\mathbf{w}$ by $\partial\mathcal{L}/\partial\mathbf{w}$ ;

  Project gradient to the positive simplex $\mathcal{C} = \{\mathbf{w} \in \mathbb{R}^d : \mathbf{1}^T\mathbf{w} = m, \mathbf{w} \geq \mathbf{0}\}$ and update the selection vector: $\mathbf{w} \leftarrow \text{Clip}\big(\text{Proj}_{\mathcal{C}}(\mathbf{w} + \eta\partial\mathcal{L}/\partial\mathbf{w}), [0,1]^d\big)$ ;

majority of the considered datasets, which are highly over-parameterized ($d \gg N$). An interesting property is that if a feature weight becomes zero at certain iteration, it remains zero at all subsequent iterations. It is easy to check, say when a Gaussian kernel is used, the derivative of the kernel $\nabla_{\mathbf{w}_i^{(t-1)}} = -k_{\mathbf{w}}(\mathbf{x}, \mathbf{y})/\sigma^2 \mathbf{w}_i^{(t-1)}(\mathbf{x}_i^2 + \mathbf{y}_i^2 - 2\mathbf{x}_i\mathbf{y}_i)$ is zero if $\mathbf{w}_i^{(t-1)} = 0$. This has advantages of maintaining feature sparsity and reducing search directions during optimization.

### 2.5.4 Experiments.

We compare KDI-based input node pruning with state-of-the-art kernel-based approaches, including backward eliminate with HSIC (BAHSIC) [220], HSIC Lasso solved by least angle regression (LAND) [266], recursive feature elimination (RFE) [76], CCM [26], MRMR [196], JMI [268], reliefF [160] and trace ratio with Fisher score [186]. We use Gaussian kernel $k(\mathbf{x}, \mathbf{y}) = \exp(-\|\mathbf{x} - \mathbf{y}\|_2^2/(2\sigma^2))$ to the input features and linear kernel $k(\mathbf{x}, \mathbf{y}) = \mathbf{x}^T\mathbf{y}$ to the labels. For the kernel bandwidth, we follow the heuristic of median pair-wise distance between standardized samples as suggested in [26, 220]. The number of desired features is predefined. The ridge parameter $\rho$ in KDI formula is tuned among $\{10^{-3}, 10^{-2}, 10^{-1}\}$ through cross-validation for each dataset. We set the factor $\lambda$ in the bi-model regularization to $10^{-3}$ throughout these experiments. Since hyper-parameters such as kernel bandwidth $\sigma$ and ridge $\rho$ have great impacts on the actual performance. Extra experiments are designed to evaluate the sensitivity to such parameters.

**Results.** After each algorithm is evaluated on a dataset and obtains the feature rank, a kernel SVM with Gaussian kernel is trained on the top $m = \{5, 10, ..., 100\}$ features and compute the accuracy. We follow the works [220, 26, 196, 265, 186] by setting the number of desired features for each method as 100, and select the top $m = \{5, 10, ..., 100\}$ features having the highest scores to compute the classification accuracy. For all experiments, the accuracy reported are based on 5-fold nested cross-validation. In each trial one fold is used for testing, one fold is used for validation, and the rest are for training. The results are averaged over 100 different training/validation/testing splits.

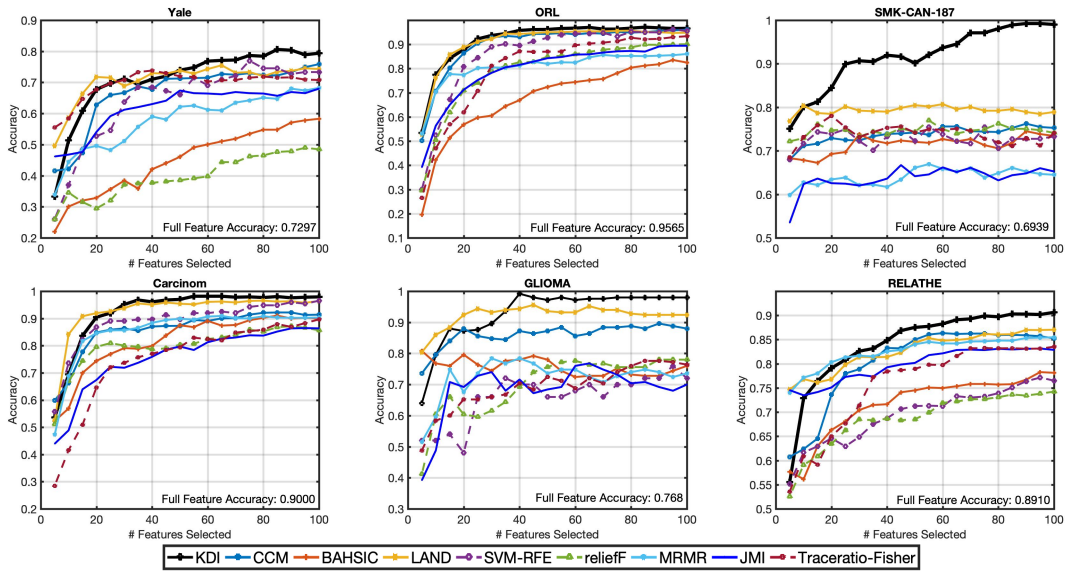



Figure 2.1: Compare KDI with other input node pruning algorithms. Kernel SVM with 5-fold nested cross-validation is used for the final classification. Horizontal axis is the number of selected features. Vertical axis is the classification accuracy. Full feature accuracy is included in each case. KDI is our approach.

For fair comparison, different methods are trained and evaluated on the same set of training and testing samples in each trial. In kernel SVM, such hyper-parameters as kernel bandwidth and $C$ are chosen based on the 5-fold cross-validation. Furthermore, to verify that feature selection methods can choose non-redundant features, we investigate the redundancy rate (RED):

$$\text{RED} = \frac{2}{m(m-1)} \sum_{i>j} |p_{i,j}| \tag{2.20}$$

where $p_{i,j}$ is the Pearson-correlation coefficient between features $i, j$. A large RED value implies that the subset of features selected are highly correlated with each other. To conclude, feature subsets achieving high classification accuracy and low RED are preferable.

Figure 2.1 shows the mean classification accuracy versus the number of selected features. Figure 2.2 shows the RED values for the top 100 features selected by each method. We observe that (1) after removing the unwanted redundancy in the input features, the generalization performance is improved; (2) KDI outperforms other competitors by clear margin. Although our method is occasionally outperformed when the number of selected features is small, it either ties or overtakes as more features are selected. This may be due to the fact that we firstly select the top 100 features, among which smaller subsets are selected based on rank; (3) KDI achieves smaller RED values on most of the datasets, indicating that our method can more effectively reduce the feature redundancy. Interestingly, linear methods like trace ratio achieve very low RED values. The linearity of the

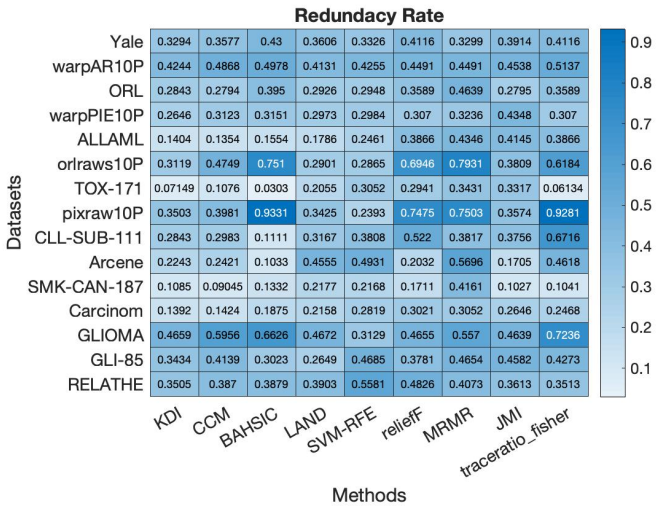
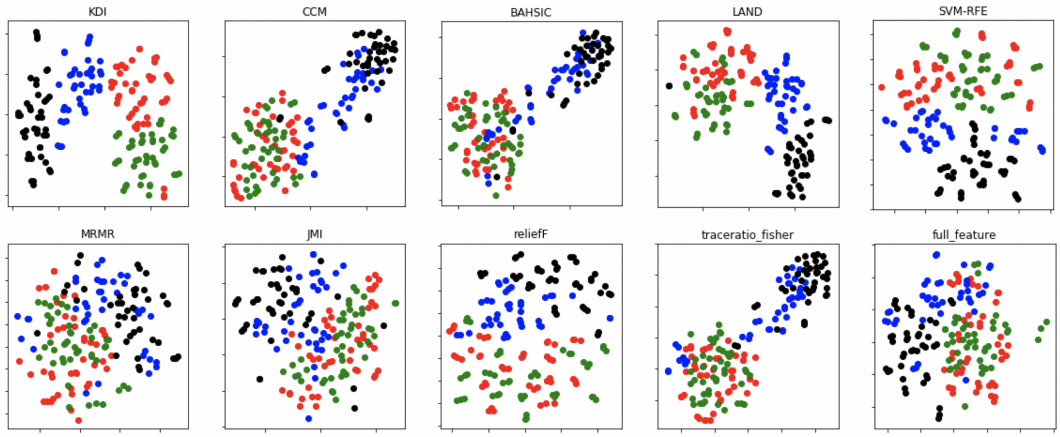



Figure 2.2: Redundancy Rate (RED) comparisons on benchmark datasets evaluated on the top 100 features selected by each method. Full feature RED is also included. KDI is our approach.




(a) Qualitative comparisons in terms of the inter-class separability on TOX-171 dataset by embedding the top 100 selected features into 2D visualization via t-SNE. KDI is our approach.

Pearson-correlation coefficient may explain why linear methods can produce smaller RED than nonlinear methods on certain datasets.

**Visualizations.** In Figure 2.2, we compare the t-SNE plots of the features retained by different methods. Dots with different colors represent samples affiliated to different classes. We can see that features retained by our method demonstrate the best inter-class separability, suggesting that the KDI metric indeed reflects the discriminant power of the features and is more suitable for searching features that best predict the labels.

**Sensitivity analysis.** The Gaussian kernel bandwidth $\sigma$ and the ridge parameter $\rho$ play important roles. The ridge $\rho$ introduces a regularizing effect and ensures matrix invertability so that the solution

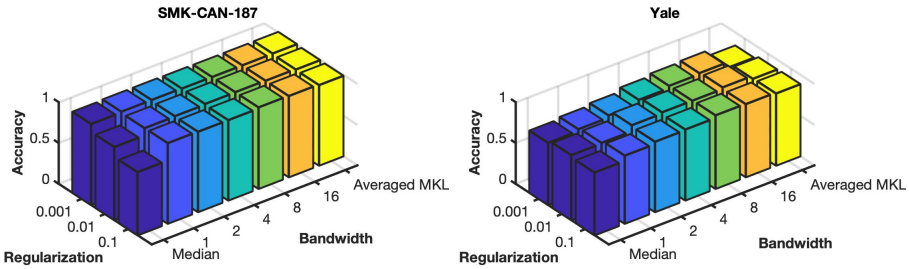



Figure 2.4: Sensitivity of the KDI method to hyper-parameters $\rho$ and $\sigma$ on SMK-CAN-187 (left) and Yale (right) datasets.

is more stable, while the kernel bandwidth can greatly influence the statistical efficacy of the KDI metric. We investigate how these two parameters can affect the performance of the proposed method by varying $\rho$ in the set $\{10^{-3}, 10^{-2}, 10^{-1}\}$ and $\sigma$ in the set $\{\text{Median}, 1, 2, 4, 8, 16, \text{Multi-Kernel}\}$ (Multi-Kernel stands for a uniform mixture of the kernels considered). In Figure 2.4, our method is robust to both ridge and Gaussian kernel bandwidth, except when the median pair-wise distance heuristic is used. In this case, different ridge parameters yield dramatically different accuracy. Therefore, a generic kernel or a uniform mixture of kernels are sufficient to obtain good results.

## 2.6 Conclusion

In this chapter, we went over basic concepts on dimension reduction and subspace projection, and introduced a discriminant component analysis method that finds a subspace representation maximizing the feature separability. Extensions to non-linear settings using kernel methods were also provided. The DCA and KDCA methods provided us powerful selection and optimization metrics called DI and KDI, which we applied to the input node pruning problem. In the next chapter, we will extend our subspace analysis method and the DI metric to deep learning models.

# Chapter 3

# From Subspace Analysis to Structural Learning

## 3.1 Prologue

Deep learning models have provided versatile platforms to facilitate a broad spectrum of AI applications. The task of training the model parameters has been masterly handled by the back-propagation learning. However, the pursuit on optimal model structures remains largely an art of trial-and-error. This prompts some urgency to explore various structural learning approaches for automating the design of more efficient and accurate model structures.

Along this vein, this chapter extends the subspace analyses to deep learning models, and proposes a framework to jointly learn the model structure and parameters, with the objective of reducing the model redundancy and computation demands while retaining or even improving the performance. A long-standing question is how to develop a theoretical footing to rank the importance of parameters or features so that reducing the unimportant features causes little impact to the model quality. Most of the existing methods use heuristics based on the filter weights [141, 90, 40]: weights with larger magnitude or norm values are more important. These heuristics ignore the distribution of the input data and the feature-output correlation, and thus may not accurately capture the feature importance. On the other hand, using statistics computed from the feature-maps can reflect how the inputs are transformed to predict the final labels by the model weights. Inspired by the design principle of GoogLeNet [227, 226] and DSN [134] showing the importance of intermediate feature discriminant power to the final performance, recent methods begin to explore discriminant-based metrics. For

example, [303] introduced auxiliary cross-entropy losses to intermediate layers in order to evaluate the feature-maps discriminant power. However, due to the additional auxiliary losses, the retraining step requires expensive computation time.

In this chapter, we develop a closed-form optimization metric to evaluate the feature-maps discriminant power more efficiently by unifying the discriminant-based and correlation-based subspace analysis methods. We define the deleterious features as the one with minimum impact to the feature-maps discriminant power in local layers, and we propose a novel differential discriminant metric, which can be used as the gradient information for updating the model structure. Although our metric is data-dependent, we find it robust to the input samples distribution, meaning that we only need a small portion of training samples (for example, only one mini-batch of 256 images) to accurately evaluate the parameter importance.

Moreover, we propose an iterative structural learning algorithm to automatically adapt the model structures to the optimal down-sized one satisfying certain resource budget. To ensure convergence of the iterative learning and the effectiveness of the differential discriminant metric, we stick to the minimal updating principle to seamlessly inherit most of the existing weight parameters and avoid training the entire network from scratch. At each iteration, we generate a pool of candidates, each of which have one layer being reduced by an unit of FLOPs, storage, power, or latency, etc. Assessment of the candidates is based on the generalization performance on the validation data, and the best candidate is selected to the next iteration after short-term finetuning. We iteratively update the structure and parameters until reaching the target. More details and simulations are introduced next.

## 3.2 Methodology

### 3.2.1 Feature-maps discriminant power

Evaluating the feature-maps discriminant power can be considered from two perspectives: the discriminant component analysis for analyzing class separability and the correlation analysis on how well the features predict the label. Consider a pre-trained $L$-layer model with weights $\{\mathbf{w}^l, ..., \mathbf{w}^L\}$, where $\mathbf{w}^l$ represents the weights in $l$-th layer, $\mathbf{w}^l_{j,:}$ indicates the $j$-th neuron in $l$-th layer, and $C^l$ is the number of neurons. Given an input sample $\mathcal{I}$, let $\mathbf{x}^l(\mathcal{I})$ denote the feature-maps of $l$-th layer, $\mathbf{x}^l_{j,:}(\mathcal{I})$ indicates the feature from the $j$-th neuron. We apply a binary indicator $\mathbf{m}^l \in \{0,1\}^{C^l}$ to the feature-maps (i.e. $\mathbf{x}^l(\mathcal{I}) \odot \mathbf{m}^l$), where setting $m^l_j = 0$ is equivalent to reducing $j$-th neuron.

**Discriminant-based subspace analysis.** Recall from previous chapter that the ratio of the inter-class separability and intra-class variation can evaluate the feature discriminant power. Let $\mathbf{X}^l = [\mathbf{x}^l(\mathcal{I}_1), ..., \mathbf{x}^l(\mathcal{I}_N)] \in \mathbb{R}^{N \times C^l}$ be the matrix of feature-maps for $N$ input examples in the $l$-th layer of the model, $\mathbf{Y} = [\mathbf{y}_1, ..., \mathbf{y}_N]$ be the label matrix (e.g. one-hot encoding vectors in classification tasks). For all the derivations as follows, we assume both the features and labels have mean subtracted. We use the DCA formula to measure the discriminant power for local layers:

$$\underset{\mathbf{F}:\mathbf{F}^T(\bar{\mathbf{K}}^l+\rho\mathbf{I})\mathbf{F}=\mathbf{I}}{\text{maximize}} \quad \text{tr}(\mathbf{F}^T\mathbf{K}_B^l\mathbf{F}), \tag{3.1}$$

where matrix $\mathbf{K}_B^l = {\mathbf{X}^l}^T\mathbf{Y}\mathbf{Y}^T\mathbf{X}^l$ is the between-class scatter matrix and $\bar{\mathbf{K}}^l = {\mathbf{X}^l}^T\mathbf{X}^l$ is the standard scatter matrix with mean-subtracted. $\text{tr}(\cdot)$ stands for the matrix trace operation. The term $\mathbf{F}^T\mathbf{K}_B^l\mathbf{F}$ measures the inter-class data variability in the subspace spanned by columns of matrix $\mathbf{F}$, while the term $\mathbf{F}^T(\bar{\mathbf{K}}^l + \rho\mathbf{I})\mathbf{F}$ measures the within-class data variability in the subspace. Parameter $\rho$ is introduced to ensure numerical stability. Equation 3.1 can be easily solved by generalized eigen-decomposition, which yields $\mathbf{F}^* = (\bar{\mathbf{K}}^l + \rho\mathbf{I})^{-1}{\mathbf{X}^l}^T\mathbf{Y}$. After plugging in $\mathbf{F}^*$, we obtain the maximum discriminant power, which we refer to as the discriminant information (DI):

$$\text{DI} = \text{tr}\left((\bar{\mathbf{K}}^l + \rho\mathbf{I})^{-1}\mathbf{K}_B^l\right). \tag{3.2}$$

**Correlation-based subspace analysis.** Since the learning process is to gradually transform the features to make them more aligned to the target output. Mathematically, it means that correlation between the subspace spanned by the hidden layers' features and the subspace at the output layer is gradually improved [254]. The Least-Squared-Error (LSE) has been a classic metric for correlation analysis in statistics and estimation theory. The LSE is also related to the linear probing strategy to inspect the hidden features [5], which appends intermediate linear models and examine whether the hidden features can predict the labels. The known LSE formula (with ridge penalty) is given by:

$$\text{LSE} = \|\mathbf{Y}\|_F^2 - \text{tr}\left(({\mathbf{X}^l}^T\mathbf{X}^l + \rho\mathbf{I})^{-1}{\mathbf{X}^l}^T\mathbf{Y}\mathbf{Y}^T\mathbf{X}^l\right) \tag{3.3}$$

In correlation-based subspace analysis, we find the best subspace representation of the features, denoted by $\mathbf{X}^l\mathbf{F}$, that can achieve the minimal LSE (MLSE):

$$\underset{\mathbf{F}}{\text{minimize}} \ \|\mathbf{Y}\|_F^2 - \text{tr}\left((\mathbf{F}^T{\mathbf{X}^l}^T\mathbf{X}^l\mathbf{F} + \rho\mathbf{I})^{-1}\mathbf{F}^T{\mathbf{X}^l}^T\mathbf{Y}\mathbf{Y}^T\mathbf{X}^l\mathbf{F}\right), \tag{3.4}$$

whose minimize is attained by setting $\mathbf{F}$ to the principal eigenvectors of the matrix $({\mathbf{X}^l}^T\mathbf{X}^l + \rho\mathbf{I})^{-1}{\mathbf{X}^l}^T\mathbf{Y}\mathbf{Y}^T\mathbf{X}^l$.

An equivalence is established between the discriminant-based and correlation-based subspace analysis:

$$\text{MLSE} = \|\mathbf{Y}\|_F^2 - \text{DI} \tag{3.5}$$

When the feature-maps achieve higher inter-class separability and lower with-in class variation, then the feature-output will have high correlation in terms of the least-squared error. We use the symbol $\psi$ to denote the discriminant information from now on. $\psi$ is permutation invariant: let $\pi(\mathbf{X}^l)$ denote an arbitrary permutation of the feature matrix along its feature dimension, then we have $\psi(\pi(\mathbf{X}^l)) = \psi(\mathbf{X}^l)$. This property ensures all neurons are equally treated without bias. Another property is is that $\psi$ non-decreasing with respect to the number of neurons: denote $\mathcal{T}, \mathcal{S}$ as different subset of neurons, we have $\psi(\mathcal{T}) \leq \psi(\mathcal{S})$ when $\mathcal{T} \subseteq \mathcal{S}$. This property shows that reducing neurons will degrade (at most maintain) the discriminant power. We shall only reduce the neurons with minimum impact to the discriminant power.

### 3.2.2 Differential discriminant

Given a desired sparsity $\kappa^l$, we aim to find $\kappa^l C^l$ neurons that maximally preserve the discriminant power, i.e. maximizing $\psi$ evaluated on the neuron subset. In previous chapter, we have formulated an optimization problem by finding the best subgroup of input nodes that maximizes the DI objective, solve it using a relaxation. However, for a deep learning model with tens of thousands of neurons, solving the optimization problem becomes problematic in terms of convergence. Therefore, we evaluate the importance of single neuron individually by measuring the impact of removing a single neuron on the discriminant power $\psi$. Recall that we apply a binary indicator $\mathbf{m}^l \in \{0,1\}^{C^l}$ to the feature-maps (i.e. $\mathbf{x}^l \odot \mathbf{m}^l$). We compute the absolute difference of $\psi$ when $m_j^l = 1$ (channel $j$ is retained) and $m_j^l = 0$ (channel $j$ is pruned):

$$|\operatorname{tr}\left((\bar{\mathbf{K}}^l + \rho\mathbf{I})^{-1}\mathbf{K}_B^l\right) - \operatorname{tr}\left((\operatorname{diag}(\mathbf{1}-\mathbf{e}_j)\bar{\mathbf{K}}^l\operatorname{diag}(\mathbf{1}-\mathbf{e}_j) + \rho\mathbf{I})^{-1}\operatorname{diag}(\mathbf{1}-\mathbf{e}_j)\mathbf{K}_B^l\operatorname{diag}(\mathbf{1}-\mathbf{e}_j)\right)|, \tag{3.6}$$

where $\mathbf{e}_j$ has zeros everywhere except for $j$-th element where it is one, and $\operatorname{diag}(\cdot)$ converts a vector to diagonal matrix. Based on the first-order Tylor series, the difference can be approximated by the

derivative of $\psi$, which we call as the **differential discriminant** $d\psi_j^l$:

$$d\psi_j^l = \Big|\mathbf{e}_j{}^T\Big(\frac{\partial \operatorname{tr}\left((\operatorname{diag}(\mathbf{m}^l)\bar{\mathbf{K}}^l \operatorname{diag}(\mathbf{m}^l) + \rho\mathbf{I})^{-1} \operatorname{diag}(\mathbf{m}^l)\mathbf{K}_B^l \operatorname{diag}(\mathbf{m}^l)\right)}{\partial \mathbf{m}^l}\Big|_{\mathbf{m}^l=\mathbf{1}}\Big)\Big|. \tag{3.7}$$

When the number of neurons is very large, computation of the matrix inverse operation becomes prohibitive. We derive an equivalent but more efficient formula as follows:

$$\begin{aligned} &\|\mathbf{Y}\|_F^2 - \operatorname{tr}\left((\mathbf{X}^{l^T}\mathbf{X}^l + \rho\mathbf{I})^{-1}\mathbf{X}^{l^T}\mathbf{Y}\mathbf{Y}^T\mathbf{X}^l\right) \\ &= \rho \cdot \operatorname{tr}\left(\mathbf{Y}\mathbf{Y}^T(\mathbf{X}^l\mathbf{X}^{l^T} + \rho\mathbf{I})^{-1}\right). \end{aligned} \tag{3.8}$$

The differential discriminant formula becomes $d\psi_j^l = \Big|\mathbf{e}_j{}^T\Big(\frac{\partial \operatorname{tr}\left(\mathbf{Y}\mathbf{Y}^T(\tilde{\mathbf{X}}^l\tilde{\mathbf{X}}^{l^T} + \rho\mathbf{I})^{-1}\right)}{\partial \mathbf{m}^l}\Big|_{\mathbf{m}^l=\mathbf{1}}\Big)\Big|$, where $\tilde{\mathbf{X}}^l = \mathbf{X}^l \odot \mathbf{m}^l$. Note that the inverse operation in Eq.(3.7) is performed on a matrix with size $\mathbb{R}^{C^l \times C^l}$, where $C^l$ is the number of neurons. While the inverse in the equivalent formula is on a matrix with size $\mathbb{R}^{B\times B}$, where $B$ is the batch size. Since we only use a mini-batch to compute the metric in practice, the second formula has much lower complexity on wide layers.

When the value of the derivative is high, $j$-th neuron has a considerable impact to feature-maps discriminant power. When using the differential discriminant metric to rank the neuron importance, we compute $d\psi_j^l, j \in [C^l]$ for each neuron in the layer and sort them in a descending order $\text{Rank}^l = \text{Sort}(d\psi_1^l, ..., d\psi_{C^l}^l)$. We evaluate $\text{Rank}^l$ for each layer independently. In each layer, we only retain the top-ranked neurons while reducing the other unimportant neurons.

### 3.2.3 Iterative structural learning

Our final goal is to learn a model (both parameters and structures) that meet certain resource budget while maximizing the accuracy. Considering that different layers have different importance and computations, we propose an iterative algorithm to reduce the model until satisfying the budget by comparing the layer importance at each step. The algorithm is illustrated in Figure 3.2. At each step, two operations are applied to each layer individually: (i) determine how many neurons to reduce in the layer so that the model can reduce the computation by a predefined factor $\delta$ (denoted as reduction schedule, e.g. $\delta$=0.5% of total FLOPs); (ii) reduce the unimportant neurons based on the differential discriminant metric. As a result, each step will generate $L$ different models. We evaluate them on the validation set, and finetune the model with the highest validation accuracy for very short epochs and get into the next step. The process ends when the obtained model at certain step has a computation less than the target constraint, and we finetune the final model to recover the accuracy.

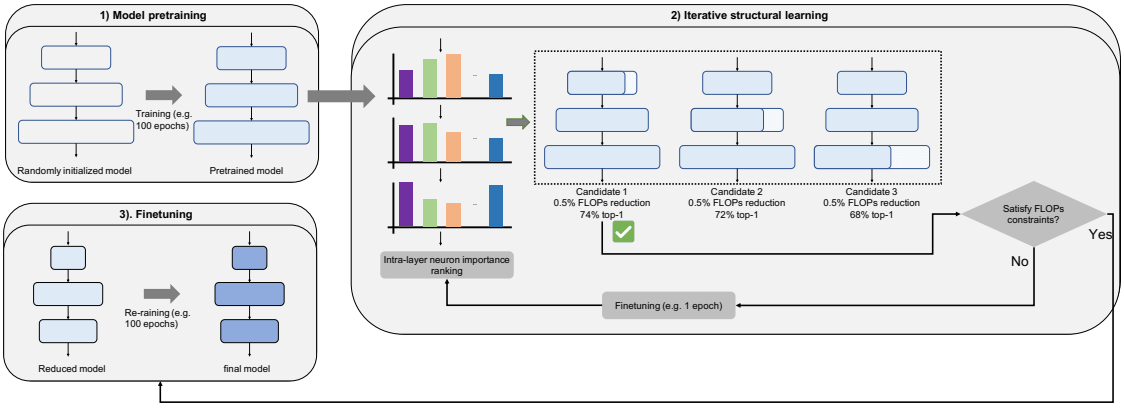



Figure 3.1: Illustration of the proposed iterative structural learning, where the intra-layer neuron importance ranking is obtained by the differential discriminant formula, and we compare inter-layer importance based on validation accuracy drop after an unit of reduction on each layer individually.

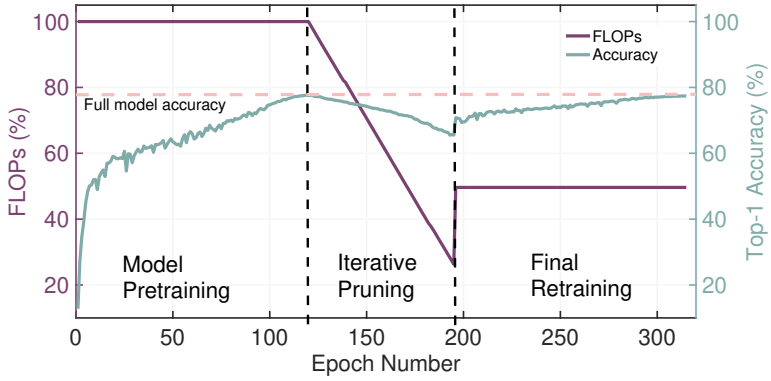



Figure 3.2: Illustration of overall training curve when applying the iterative structural learning algorithm to ResNet-50 model on ImageNet dataset.

## 3.3 Experiments

We evaluate on CIFAR10/100 [131] and ImageNet [42] datasets. We use ResNet [85] and MobileNetV2 [210] as the base architectures for structural learning. Since we propose a feature-maps differential discriminant metric, we call our method as **FMD**. We investigate two method variants. In **FMD-unif**, we evaluate the neuron importance and simply prune all layers uniformally. Many existing works adopt the uniform one-shot pruning strategy and focus on the comparison among different criteria for importance estimation, we use FMD-unif to validate the proposed differential discriminant metric. In **FMD-iterative**, we adopt iterative structural learning algorithm in Figure 3.2.

In Table 3.1, we compare the results of ResNet20/56/110 on CIFAR10/100. Firstly, our most naive variant FMD-unif yields better accuracy than the state-of-the-art methods under same or more FLOPs reduction, indicating that the proposed differential discriminant metric can preserve more important neurons/channels and achieve better accuracy. Moreover, FMD-iterative improves the

| Model | Method | CIFAR10 | | | CIFAR100 | | |
|---|---|---|---|---|---|---|---|
| | | Acc.% | FLOPs (PR%) | Params. (PR%) | Acc.% | FLOPs (PR%) | Params. (PR%) |
| ResNet20 | TAS [53] | 92.9→92.9 | 22.4M (45.0) | - | 68.7→68.9 | 22.4M (45.0) | - |
| | FPGM [90] | 92.2→91.1 | 24.3M (42.2) | - | 67.6→66.9 | 24.3M (42.2) | - |
| | **FMD-unif** | 92.4→93.0 | 22.1M (45.7) | 0.17M (37.0) | 68.5→69.3 | 22.1M (45.7) | 0.17M (37.0) |
| | **FMD-iterative** | 92.4→93.2 | 22.1M (45.7) | 0.17M (37.0) | 68.5→69.6 | 22.1M (45.7%) | 0.17M (37.0) |
| ResNet56 | DCP [303] | 93.8→93.5 | 62.5M (50.0) | 0.43M (49.2) | - | - | - |
| | FPGM [90] | 93.6→93.5 | 59.4M (52.6) | - | 71.4→69.7 | 59.4M (52.6) | - |
| | TAS [53] | 94.5→93.7 | 59.5M (52.7) | - | 73.2→72.3 | 61.2M (51.3) | - |
| | CPLI [70] | 93.7→93.8 | 62.5M (50.0) | - | - | - | - |
| | HRank [154] | 93.3→93.5 | 88.7M (29.3) | 0.71M (16.8) | - | - | - |
| | **FMD-unif** | 93.6→93.9 | 57.9M (53.7) | 0.41M (51.8) | 72.7→72.3 | 58.0M (53.6) | 0.42M (50.6) |
| | **FMD-iterative** | 93.6→94.2 | 50.0M (60.0) | 0.44M (48.2) | 72.7→72.4 | 49.7M (60.2) | 0.52M (38.8) |
| ResNet110 | SFP [88] | 93.7→93.9 | 150M (40.8) | - | 74.1→71.3 | 121M (52.3) | - |
| | FPGM [90] | 93.7→93.9 | 121M (52.3) | - | 74.1→72.6 | 121M (52.3) | - |
| | TAS [53] | 95.0→94.3 | 119M (53.0) | - | 75.1→73.2 | 120M (52.6) | - |
| | HRank [154] | 93.5→94.2 | 148M (41.2) | 1.04M (39.4) | - | - | - |
| | **FMD-unif** | 93.7→94.4 | 117M (53.7) | 0.80M (53.4) | 74.1→73.7 | 117M (53.7) | 0.80M (53.4) |
| | **FMD-iterative** | 93.7→95.3 | 100M (60.5) | 0.83M (51.7) | 74.1→74.3 | 101M (60.1) | 1.09M (36.6) |

Table 3.1: Pruning results on CIFAR10/100. "PR": pruning ratios for FLOPs or parameters. "-": results not reported by the corresponding method.

accuracy upon FMD-unif and other methods while achieving higher FLOPs reduction. FMD-iterative automatically searches the optimal structure instead of using manually designed uniform pruning strategy. Being aware to the different impact of different layers, FMD-iterative can achieve better accuracy by preserving more important layers while reducing more on redundant layers, leading to more computation reduction compared to other methods.

In Table 3.2, we compare the pruning results of ResNet50 and MobileNetV2 on ImageNet. Again, FMD-unif outperforms previous state-of-the-art methods. For example, our pruned ResNet50 achieves 75.12% top-1 accuracy (compared to 74.4% by DMCP [72]), while reducing FLOPs more than 70%. FMD-unif does not require any additional constraints, auxiliary loss terms, or layer sensitivity analysis. Since we finetune the model with the same procedure as comparative methods, the performance improvement validates the effectiveness of the proposed different discriminant metric on large-scale datasets. Moreover, FMD-iterative further improves the accuracy upon FMD-unif. For examples, ResNet50 with 70% FLOPs reduction achieves 75.6% accuracy by FMD-iterative, which is 0.5% higher than FMD-unif.

## 3.4 Analyses

### 3.4.1 Ablation study

**Impact of input samples to FMD.** The proposed FMD is a data-dependent metric. We need to feed input data, extract intermediate feature-maps from middle layers and compute the metric. We find that the FMD metric is robust to the input samples distribution. In Figure 3.3, we use

| Model | Method | Top-1% | ΔTop-1% | FLOPs (PR%) | Params. (PR%) |
|---|---|---|---|---|---|
| ResNet50 | Uniform 0.5x | 76.15 → 72.10 | -4.05 | 1.10G (73.3) | 6.40M (75.0) |
| | DCP [303] | 76.10 → 72.75 | -3.35 | 1.18G (71.4) | 8.71M (66.0) |
| | Taylor-FO-BN [182] | 76.18 → 71.69 | -4.49 | 1.34G (67.5) | 7.90M (69.1) |
| | Meta-Pruning [169] | 76.60 → 73.40 | -3.20 | 1.10G (73.3) | - |
| | DMCP [72] | 76.60 → 74.40 | -2.20 | 1.10G (73.3) | - |
| | Hrank [154] | 76.15 → 71.98 | -4.17 | 1.55G (62.4) | 13.77M (46.2) |
| | PFS [253] | 76.10 → 72.80 | -3.30 | 1.00G (75.7) | - |
| | Polarization [302] | 76.15 → 74.15 | -2.00 | 1.23G (70.0) | - |
| | LFPC [87] | 76.15 → 74.46 | -1.69 | 1.60G (60.8) | - |
| | **FMD-unif** | 76.15 → 75.12 | -1.03 | 1.10G (73.3) | 8.70M (66.0) |
| | **FMD-iterative** | 76.15 → 75.60 | -0.55 | 1.00G (75.7) | 8.23M (67.9) |
| MobileNetV2 | DCP [303] | 70.11 → 64.22 | -5.89 | 165M (45) | 2.57M (25.9) |
| | CPLI [70] | 72.19 → 67.35 | -4.84 | 165M (45) | 2.57M (25.9) |
| | DMC [63] | 71.88 → 68.37 | -3.51 | 162M (46) | - |
| | **FMD-unif** | 71.80 → 69.33 | -2.47 | 150M (50) | 1.92M (44.7) |
| | **FMD-iterative** | 71.80 → 69.70 | -2.10 | 150M (50) | 2.20M (37.1) |

Table 3.2: Pruning results on ImageNet. "PR%": pruning ratios for FLOPs or parameters. "-": results not reported by corresponding method.

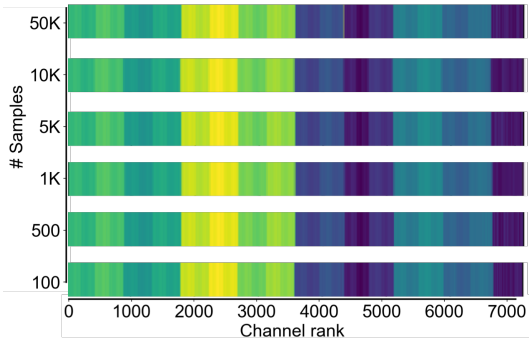



Figure 3.3: Influence of number of input samples on FMD-based neuron ranking.

different colors to represent the global ranking of all neurons/channels in MobileNetV2 when using different number of input samples to compute the FMD metric. The ranking is almost unchanged regardless of the number of input samples. Moreover, we compare the accuracy when using different number of examples to compute the FMD metric in Table 3.3. We can efficiently estimate the neuron importance using only a small portion of training samples.

| Number of samples | 100 | 500 | 1K | 5K | 10K | 50K |
|---|---|---|---|---|---|---|
| Accuracy (%) | 76.12 | 76.08 | 76.06 | 76.13 | 76.13 | 76.16 |

Table 3.3: Influence of number of input samples used to compute the FMD metric to the final accuracy.

**Impact of the reduction factor $\delta$.** In Figure 3.4, we investigate the impact of the per-step reduction factor $\delta$ in our iterative structural learning algorithm. A smaller reduction factor yields better accuracy but requires more iterations to satisfy the resource budget.

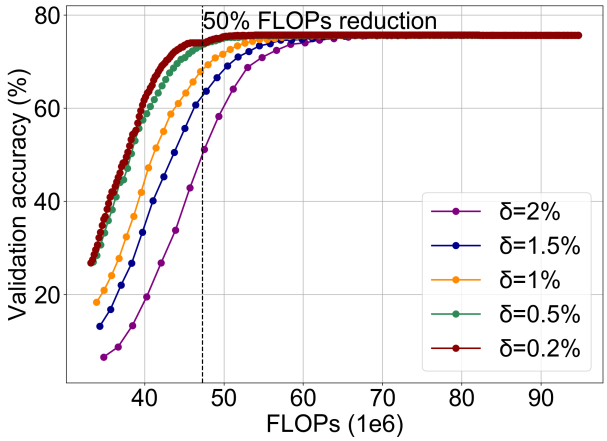



Figure 3.4: Impact of the per-step reduction factor $\delta$ on the validation accuracy (without finetuning).

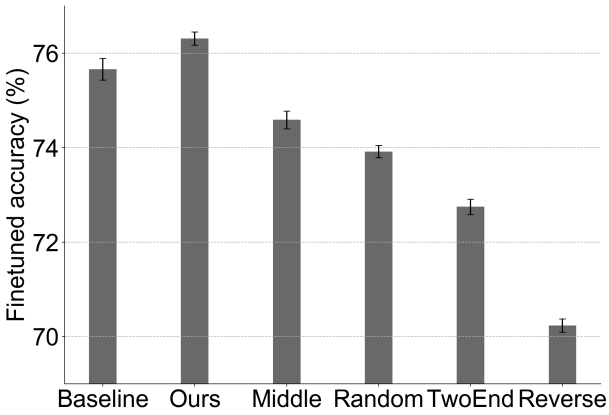



Figure 3.5: Comparison of different FMD-based selection stratgies. Our default is to retain neurons with the high FMD values.

**Comparison of different selection strategies.** To verify the appropriateness of retaining neurons with high FMD value, we compare with four other strategies, including (1) Reverse: channels with high FMD are reduced, (2) TwoEnd: neurons with both the highest and lowest FMD are reduced, (2) Middle: channels with both the highest and lowest FMD are retained, (3) Random: reduce neurons randomly. Results are shown in Figure 3.5. Among other four strategies, "Middle" yields the best accuracy while "Reverse" performs the worst. This suggests that neurons with low FMD contain less information. But as long as neurons with the highest FMD are retained, the critical information is well preserved. We can see that our default strategy of retaining neurons with high FMD values outperforms all alternatives.

| Method | CIFAR10 | | ImageNet | |
|---|---|---|---|---|
| | ResNet-20→ResNet-56 | Search on ResNet-56 | ResNet-18→ResNet-50 | Search on ResNet-50 |
| FLOPs reduction | 60% | 60% | 75% | 75% |
| Top-1 | 94.9% | 95.3% | 75.4% | 75.6% |

Table 3.4: Compare transferring the structure versus directly searching the structure woth FMD-iterative.

### 3.4.2 Transferable structure patterns.

We find that the FMD-iterative method is able to learn general structure design principles for models with different depth on a specific dataset. For example, for ResNet-18 and ResNet50 on ImageNet, the optimal structure learned by the FMD-iterative method preserve most channels in down-sampling convolution layers and layers closer to the final classification layers, while shallow layers only require few channels for low-level feature extraction. This consistency suggests that the structure patterns may be transferable among different models. We can first apply FMD-iterative on smaller model with less layers (e.g. ResNet-18 on ImageNet), then apply the structure pattern to prune deeper and larger models (e.g. ResNet-50 on ImageNet) without running the iterative process again. We compare transferring the structure versus directly searching the structure with FMD-iterative in Table 3.4. For example, transferring the structure from ResNet-18 to ResNet-50 achieves 75.4% accuracy with 75% FLOPs, compared to 75.6% accuracy by directly searching on ResNet-50.

## 3.5 Conclusion

In this chapter, we apply the subspace analysis to learning better model structures with less redundancy and better accuracy. We unify the views of feature discriminant power and feature-output correlation analysis, giving rise to a differential discriminant metric for evaluating the feature importance. We propose to remove neurons or channels with minimum impact to the feature discriminant power or correlation with the model output, measured by the derivative of a closed-form discriminant information objective. Moreover, we propose iterative structural learning algorithm, which adapts the starting model to the optimal downsized one by allocate the optimal number of channels/neurons in each layer and trains the model parameters jointly. We have applied our method to ResNet and MobileNet models for adapting more efficient architectures with less computation demands. Our results on CIFAR10/100 and ImageNet datasets are by and large competitive compared to the state-of-the-arts approaches, and we even achieve both high-performance and low-computation on CIFAR datasets.

# Chapter 4

# Regressive-Progressive and Omni-Dimensional Structural Learning

## 4.1 Prologue

Neural network pruning has gone beyond model compression, and become a general approach for structural learning to achieve better model efficiency and accuracy. Many recent architecture design or search methods have applied pruning to obtain high accuracy models satisfying computation constraints [270, 113, 20]. Moreover, the model scaling law suggests that larger model size usually leads to performance improvement. To control the effective parameters and computations, recent works begin exploring large sparse models [24].

However, most of the pruning methods require the cost of training a full model and finetuning the pruned model. In these methods, the maximum size of model is limited to the full model that can be trained, and we cannot use pruning to explore the limit of larger and more accurate sparse models. Moreover, substantial training cost is consumed on parameters that will be pruned later, and we cannot use pruning to achieve training efficiency. On the other hand, recently developed deep learning models usually contain heterogeneous structures either for more complex tasks [82, 260, 295] or better accuracy [54, 236]. To optimize these complicated architectures, we need more generalized pruning approaches that learn more efficient and accurate architectures across many different dimensions.

In this chapter, we first introduce a novel Channel Exploration methodology dubbed as CHEX in section 4.2. With the CHEX method, a well-trained full model is no longer needed as the starting point for pruning. Moreover, we can start from a random sub-network structure and progressively adapt to more optimal structure during training. As opposed to pruning-only strategy, we propose to repeatedly prune and regrow the channels throughout the training process, which reduces the risk of pruning important channels prematurely. From intra-layer's aspect, we tackle the channel pruning problem via a well-known column subset selection (CSS) formulation. From inter-layer's aspect, our regrowing stages open a path for dynamically re-allocating the number of channels across all the layers under a global channel sparsity constraint . In addition, all the exploration process is done in a single training from scratch without the need of a pre-trained large model.

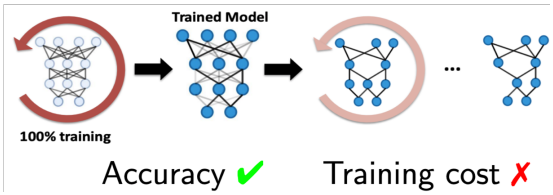
In section 4.3, we also propose an omni-dimensional structural learning approach to compress complicated architectures such as the transformer model across many dimensions jointly (attention heads, neurons and tokens). We propose a statistical dependence based pruning criterion that is generalizable to the different dimensions for identifying the redundant features. Moreover, we propose a joint optimization to search for the optimal policy on how much to reduce for different dimensions so that compressed model's accuracy can be maximized under a computational constraint. The problem is solved by an adapted Gaussian process search with the expected improvement technique.

For both methods, we evaluate on a variety of computer vision tasks, including image classification, object detection, instance segmentation, and 3D vision. Our results advance the Pareto frontier on diverse CNN and transformer architectures.

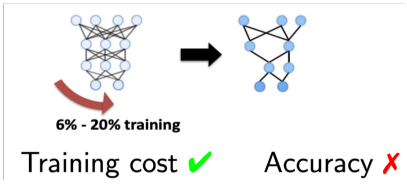
## 4.2 CHEX: CHannel EXploration

### 4.2.1 Background

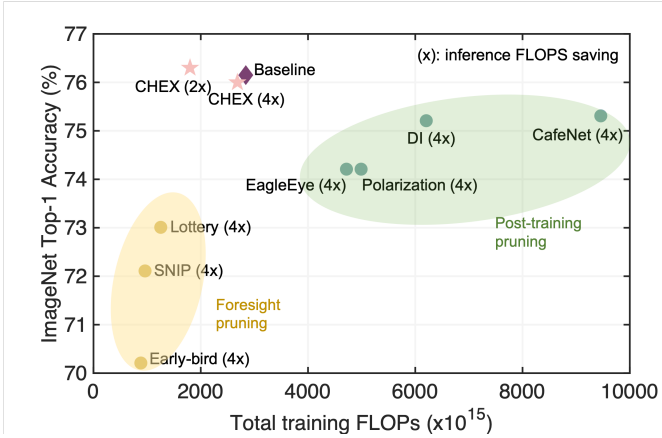
Most existing channel pruning methods [141, 91, 173, 270, 303, 40, 90, 276, 182, 70, 273, 103, 101, 175, 292] adopt a progressive pruning-training pipeline (Figure 4.1(a)): pre-training a large model until convergence, pruning a few unimportant channels by the pre-defined criterion, and finetuning the pruned model to restore accuracy. The last two stages are usually executed in an interleaved manner repeatedly, which suffers from long training time. Various attempts have been made to improve the pruning efficiency. Training-based channel pruning methods [166, 155, 143, 71, 149, 302] impose sparse regularization such as LASSO or group LASSO to the model parameters during training. However, these commonly adopted regularization may not penalize the parameters to exactly zero.




(a) Post-training pruning.




(b) Foresight pruning.




(c) Training cost of different pruning methods.

Figure 4.1: Previous works on DNN pruning. (a): The iterative pruning methods suffer from expensive training cost. (b): The early-bird pruning methods (e.g., pruning at initialization as a winning ticket) suffer from low accuracy. (3): Training flops of different methods to obtain compressed ResNet-50 on ImageNet.

Removing many small but non-zero parameters inevitably damages the model accuracy. Although this problem can be addressed by applying specialized optimizers [119, 46] or model reparameterization tricks [48], these methods require a well-pretrained large model, which counters our goal of improving pruning efficiency. Sampling-based methods [283, 188, 127, 63, 171, 106] directly train sparse models. However, these methods may suffer from training instability and converge to sub-optimal points [73]. [253, 275, 136, 244] shorten or eliminate the pre-training phase, and extract the sub-model at an early stage or even from random initialization (Figure 4.1(b)). These methods are incapable to recover prematurely pruned important channels, which limits the model capacity and leads to unacceptable accuracy degradation.

To rectify the aforementioned limitations, a *channel exploration* methodology called **CHEX** is proposed, which obtains high accuracy sub-models without pre-training a large model or finetuning the pruned model, as shown in Fig. 4.2. In contrast to the traditional pruning approaches that permanently remove channels [141, 173, 278, 303], CHEX dynamically adjusts the importance of the channels via a *periodic pruning and regrowing process*, which allows the prematurely pruned channels to be recovered and prevents the model from losing the representation ability early in the training process. From intra-layer's perspective, the channel pruning problem is reformulated into the classic

*column subset selection* (CSS) problem in linear algebra, leading to a closed-form solution. From inter-layer's perspective, rather than sticking to a fixed or manually designed sub-model structure, CHEX re-evaluates the importance of different layers after each regrowing stage. This leads to a *sub-model structure exploration* technique to dynamically re-allocate the number of channels across all the layers under a given budget. With only one training pass from scratch, sub-models obtained from CHEX yield better accuracy than the previous methods under the same FLOPs reductions.

In summary, the contributions are highlighted as follows:

- Our channel exploration methodology CHEX is proposed with with three novel features: (1) a periodic channel pruning and regrowing process, (2) a data-independent pruning criterion based on column subset selection, and (3) an efficient structure exploration technique.
- CHEX obtains the sub-model in one training pass from scratch, effectively reducing the training cost, as it circumvents the expensive pretrain-prune-finetune cycles.
- Experimentally, CHEX exhibits superior accuracy under different FLOPs constraints, and is applicable to a plethora of computer vision tasks. For image classification on ImageNet, our compressed ResNet-50 model yields 4× FLOPs reduction while achieving 76% top-1 accuracy, improving previous best result [222] by 0.7%. For object detection on COCO dataset, our method achieves 2× FLOPs reduction on the SSD model while improving 0.7% mAP over the unpruned baseline. For instance segmentation on COCO dataset and 3D point cloud segmentation on ShapeNet dataset, our method achieves 2× and 11× FLOPs reductions on the Mask R-CNN and PointNet++ models with negligible quality loss compared to the unpruned models, respectively.
- We present theoretical convergence guarantee of the CHEX method from the view of non-convex optimization setting, which is applicable to deep learning.

### 4.2.2 Overview of CHEX method

CHEX takes an existing CNN model as the channel exploration space, which provides the maximum number of explored channels. As illustrated in Figure 4.2, we describe the training flow of CHEX using a 3-layer CNN model as an example, in which we set the target channel sparsity to 50%. From top to bottom, the three layers contain 6, 8, and 10 channels, respectively, denoted as $\text{Conv}-6-8-10$. CHEX starts with a randomly initialized sub-model. During training, at periodically spaced training iterations (e.g., pre-defined $\Delta T$ iterations), a channel pruning and regrowing process is performed,

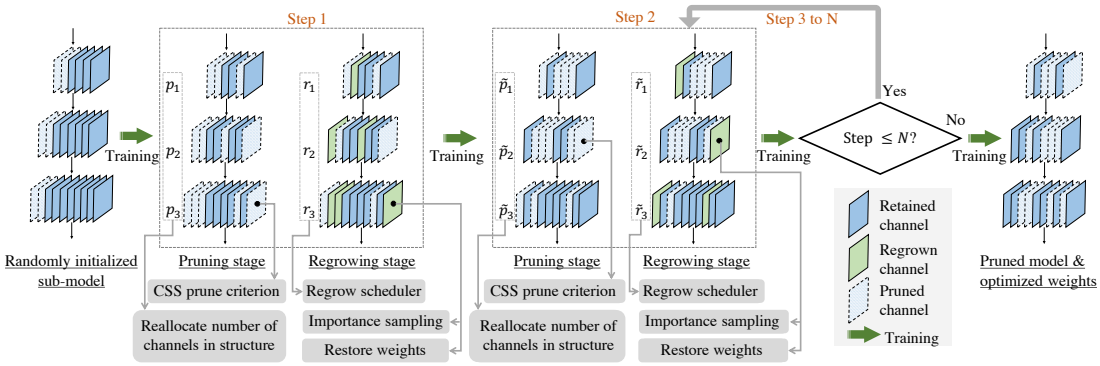



Figure 4.2: An illustration of CHEX, which jointly optimizes the weight values and learns the sub-model structure in one training pass from scratch. In CHEX, both retained and regrown channels in the sub-model are active, participating in the training iterations.

which is referred to as one step in CHEX. A total of $N$ steps are applied, and each step is composed of the following stages:

- A pruning stage removes the unimportant channels to the target sparsity. The number of channels are re-allocated across all different layers via the sub-model structure exploration technique. For example, in Figure 4.2, the pruned sub-model in step 1 has an architecture $\text{Conv}-3-4-5$, retaining 12 out of 24 channels. It may be adjusted later to $\text{Conv}-2-3-7$ in step 2. Such adaptations continue in the loop across the $N$ steps.

- Immediately after pruning, a channel regrowing stage regrows back a fraction of the previously pruned channels, whose weight values are restored to their most recently used values before being pruned. Note that the regrown channels may be pruned multiple steps before. A decay scheduler is adopted to gradually reduce the number of regrown channels across the $N$ steps. For example, in Figure 4.2, the channel regrowing stage in step 1 re-activates six channels, while it only re-activates four channels in step 2.

Our method interleaves the model training and the periodic pruning-regrowing stages. Since the total number of regrown channels is reduced at each step, the sub-model under training enjoys gradually decreased computation cost and converges to the target channel sparsity in the end. As an algorithm guideline, the pseudo-code of CHEX is provided in Algorithm 2.

**Algorithm 2:** CHEX: CHannel EXploration for CNN Model Compression.

**Input**: An $L$-layer CNN model with weights $\mathbf{W} = \{\mathbf{w}^1, ..., \mathbf{w}^L\}$; target channel sparsity $S$; total training iterations $T_{\text{total}}$; initial regrowing factor $\delta_0$; training iterations between two consecutive steps $\Delta T$; total pruning-regrowing steps $T_{\max}$; training set $\mathcal{D}$ ;
**Output**: A sub-model satisfying the target sparsity $S$ and its optimal weight values $\mathbf{W}^*$;
Randomly initialize the model weights $\mathbf{W}$;
**for** *each training iteration* $t \in [T_{total}]$ **do**
    Sample a mini-batch from $\mathcal{D}$ and update the model weights $\mathbf{W}$ ;
    **if** $Mod(t, \Delta T) = 0$ *and* $t \leq T_{max}$ **then**
        Redistribute the number of channels across layers for searching architecture by Eq.(4.4) ;
        Remove $\{\kappa^l C^l, l \in [L]\}$ channels by CSS-based pruning algorithm 3 ;
        Compute the channel regrowing factor by a decay scheduler function ;
        Reactive some channels by importance sampling-based regrowing algorithm 4 ;

### 4.2.3 Pruning stage in CHEX.

Suppose a CNN contains $L$ layers with parameters $\{\mathbf{w}^1, ..., \mathbf{w}^L\}$, with $\mathbf{w}^l \in \mathbb{R}^{H^l W^l C^{l-1} \times C^l}$ denoting the reshaped[1] convolution weight matrix. $C^{l-1}, C^l, H^l \times W^l$ represent the number of input channels, the number of output channels, and the kernel size, respectively. The $j$-th channel in the $l$-th layer is denoted as $\mathbf{w}^l_{:,j}$. That is, a column in $\mathbf{w}^l$ represents one channel in the convolution. For notation simplicity, we use $K^l = H^l W^l C^{l-1}$ in the following text, i.e., $\mathbf{w}^l \in \mathbb{R}^{K^l \times C^l}$.

Suppose $\kappa^l$ denotes the channel sparsity of the $l$-th layer. For each layer, channel pruning identifies a set of important channels with index set $\mathcal{T}^l$ ($|\mathcal{T}^l| = \lceil (1-\kappa^l) C^l \rceil$), which retains the most important information in $\mathbf{w}^l$, such that the remaining ones $\{\mathbf{w}^l_{:,j}, j \notin \mathcal{T}^l\}$ may be discarded with minimal impact to model accuracy. In other words, channel pruning selects the most "representative" columns from $\mathbf{w}^l$ that can reconstruct the original weight matrix with minimal error. From this perspective, channel pruning can be naturally represented as the *Column Subset Selection* (CSS) problem in linear algebra [69]. This provides us a new theoretical guideline for designing channel pruning criterion in a principled way, rather than depending on heuristics. Notably, the proposed CSS-based pruning algorithm jointly evaluate the importance of group of channels in the context of recovering the full matrix space, in contrast of previous heuristics that evaluate the importance of each channel independently. Moreover, CSS presents a data-independent (or data-free) approach for channel pruning, which is more efficient than previous data-dependent heuristics that require feedforwarding the input data to the model for metric computation. To rigorously characterize the most "representative" columns of a matrix, we formally define CSS as follows:

**Definition 4.2.1** (Column Subset Selection)**.** *Given a matrix* $\mathbf{w}^l \in \mathbb{R}^{K \times C^l}$*, let* $c \leq C^l$ *be the number*

[1]For ease of derivation, we convert the convolution weight tensor of size $H^l \times W^l \times C^{l-1} \times C^l$ to the matrix of size $H^l W^l C^{l-1} \times C^l$, where the channels are listed as columns in the reshaped weight matrix.

**Algorithm 3:** CSS-Based Channel Pruning in CHEX.

**Input**: Model weights $\mathbf{w}^l$; pruning ratios $\kappa^l$ ;
**Output**: The pruned layer $l$ ;
Compute the number of retained channels $\tilde{C}^l = \lceil (1-\kappa^l)C^l \rceil$ ;
Perform SVD on $\mathbf{w}^l$ and extract the top $\tilde{C}^l$ right singular vectors $\mathbf{V}^l_{\tilde{C}^l}$ ;
Compute the leverage scores $\psi^l_j = \|[\mathbf{V}^l_{\tilde{C}^l}]_{j,:}\|_2^2$ for each channel $j \in [C^l]$ in the layer ;
Retain the important channels identified as $\mathcal{T}^l = \text{ArgTopK}(\{\psi^l_j\}; \tilde{C}^l)$ ;
Prune channels $\{\mathbf{w}^l_{:,j}, j \notin \mathcal{T}^l\}$ from layer $l$ ;

*of columns to select. Find c columns of* $\mathbf{w}^l$*, denoted by* $\mathbf{w}^l_c$*, that would minimize:*

$$\|\mathbf{w}^l - \mathbf{w}^l_c {\mathbf{w}^l_c}^\dagger \mathbf{w}^l\|_F^2 \quad or \quad \|\mathbf{w}^l - \mathbf{w}^l_c {\mathbf{w}^l_c}^\dagger \mathbf{w}^l\|_2^2, \tag{4.1}$$

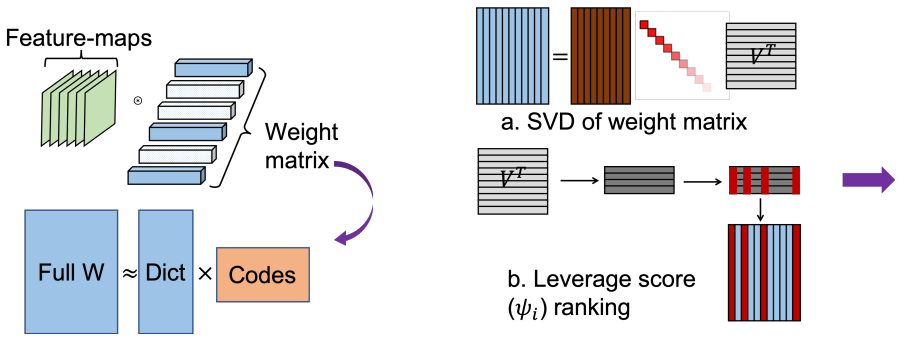
*where* $\dagger$ *stands for the Moore-Penrose pseudo-inverse,* $\|\cdot\|_F$ *and* $\|\cdot\|_2$ *represent matrix Frobenius norm and spectral norm, respectively.*

Since channel pruning and CSS share the same goal of best recovering the full matrix by a subset of its columns, we leverage the rich theoretical foundations of CSS to derive a new pruning criterion. Our channel pruning stage is conducted periodically during training, thus we employ a computationally efficient deterministic CSS algorithm, referred to as the *Leverage Score Sampling* [194]. The core of this algorithm involves the leverage scores of matrix $\mathbf{w}^l$, which are defined as follows:

**Definition 4.2.2** (Leverage Scores)**.** *Let* $\mathbf{V}_c \in \mathbb{R}^{C^l \times c}$ *be the top-c right singular vectors of* $\mathbf{w}^l$ *(c represents the number of selected columns from* $\mathbf{w}^l$*). Then, the leverage score of the j-th column of* $\mathbf{w}^l$ *is given as:* $\psi_j = \|[\mathbf{V}_c]_{j,:}\|_2^2$*, where* $[\mathbf{V}_c]_{j,:}$ *denotes the* $j^{th}$ *row of* $\mathbf{V}_c$*.*

The leverage score sampling algorithm samples $c$ columns of $\mathbf{w}^l$ that corresponds to the largest $c$ leverage scores of $\mathbf{w}^l$. Despite its simplicity, theoretical analysis in [194] has shown that this deterministic solution provably obtains near optimal low-rank approximation error for Eq.(4.1).

Based on the above analysis, we propose a CSS-based channel pruning method with leverage score sampling, as shown in Algorithm 3 and Figure 4.3. When given a pruning ratio $\kappa^l$ for layer $l$, we need to select and retain $\lceil (1-\kappa^l)C^l \rceil$ important channels. We first compute the top $\lceil (1-\kappa^l)C^l \rceil$ right singular vectors of the weight matrix $\mathbf{w}^l$. Then, we calculate the leverage scores of all the channels in this layer as Definition 4.2.2, and rank them in descending order. Finally, we identify the set of important channels to retain as $\mathcal{T}^l = \text{ArgTopK}(\{\psi^l_j\}; \lceil (1-\kappa^l)C^l \rceil)$, which gives the indices of channels with the top $\lceil (1-\kappa^l)C^l \rceil$ leverage scores of $\mathbf{w}^l$. The remaining bottom-ranking channels are pruned.




(a) Column subset selection problem (b) Leverage score sampling method.

Figure 4.3: In the pruning stages of CHEX, we use the column subset selection (CSS) method. We use the leverage score sampling algorithm to determine the most important channels that can best recover the matrix.

### 4.2.4 Regrowing stage in CHEX.

Since the method trains from a randomly initialized model and the pruning stage may be based on the weights that are not sufficiently trained. In the early stage of training, the pruning decisions may not be optimal and some important channels are prematurely pruned. Therefore, after each pruning stage, our method regrows *a fraction* of the previously pruned channels back to the model. The regrown channels are updated in the subsequent training. If they are important to the model, they may survive the future pruning stages after a number of iterations of training. Moreover, the channel regrowing stage enables the model to have better representation ability during training, since the model capacity is not permanently restricted as the one-shot pruning methods.

To complete the regrowing stage, we need to assign proper weight values to the newly activated channels. One straightforward choice is to assign zero values for stable training, since the regrown channels do not affect the output of the model. However, we find that regrowing channels with zeros would receive zero gradients in the subsequent training iterations. This is undesirable because the regrown channels would remain deactivated and the method degenerates to the one-shot early-bird pruning [275]. Based on our ablations, we find the best scheme is that the newly activated channels restore their *most recently used* (MRU) parameters, which are the last values before they are pruned. We constantly maintain a copy of the weight values of the pruned channels, in case that they may get regrown back in the future regrowing stages. Note that the regrowing stage may regrow channels that are pruned multiple steps before, instead of just re-activating what are pruned at the pruning stage immediately before the current regrowing stage.

To determine the channels to regrow, a naive way is to perform uniform sampling from the candidate set $\{j|j \in [C^l] \setminus \mathcal{T}^l\}$. However, uniform sampling does not consider the possible inter-

**Algorithm 4:** Importance Sampling for Channel Regrowing in CHEX.

**Input**: Indices of active channels $\mathcal{T}^l$ in the model; regrowing factor $\delta_t$;
**Output**: Layer $l$ after regrowing ;
Compute the sampling distribution by Eq.(4.2): $p_j^l = \exp(\epsilon_j^l)/\sum_{j'} \exp(\epsilon_{j'}^l)$ for all $j \in [C^l] \backslash \mathcal{T}^l$ ;
Compute the number of regrown channels $k^l = \lceil \delta^t C^l \rceil$ ;
Perform importance sampling $\mathcal{G}^l$=Multinomial($\{p_j^l\}; k^l$) ;
Restore the MRU weights of the reactivated channels $\{\hat{\mathbf{w}}_j^l, j \in \mathcal{G}^l\}$ ;
Regrow channels $\{\hat{\mathbf{w}}_j^l, j \in \mathcal{G}^l\}$ to layer $l$ ;

channel dependency between the active channels survived in the sub-model and the candidate channels to regrow. Instead, we propose an *importance sampling* strategy based on channel orthogonality for regrowing, as shown in Algorithm 4. Channel orthogonality automatically implies linear independency, which helps avoid trivial regrowing where the newly regrown channels lie in the span of the active channels. Channel orthogonality also encourages the channel diversity and improves model accuracy [10]. Formally, we denote the active channels in layer $l$ by matrix $\mathbf{w}_{\mathcal{T}^l}^l$. The orthogonality $\epsilon_j^l$ of a channel $\mathbf{w}_j^l$ in the candidate set with respect to the active channels can be computed by the classic orthogonal projection formula [95]:

$$\epsilon_j^l = \|\mathbf{w}_j^l - \mathbf{w}_{\mathcal{T}^l}^l (\mathbf{w}_{\mathcal{T}^l}^{l^T} \mathbf{w}_{\mathcal{T}^l}^l)^\dagger \mathbf{w}_{\mathcal{T}^l}^{l^T} \mathbf{w}_j^l\|_2^2. \tag{4.2}$$

A higher orthogonality value indicates that the channel is harder to approximate by others, and may have a better chance to be retained in the CSS pruning stage of the future steps. Thus, the corresponding channel may be sampled with a relatively higher probability. We use the orthogonality values to design our importance sampling distribution, and the probability $p_j^l$ to regrow a channel $\mathbf{w}_j^l$ is given as:

$$p_j^l = \exp(\epsilon_j^l)/\sum\nolimits_{j'} \exp(\epsilon_{j'}^l). \tag{4.3}$$

Then, the channels to regrow are sampled according to the distribution Multinomial($\{p_j^l | j \in [C^l] \setminus \mathcal{T}^l\}; \lceil \delta^t C^l \rceil$) without replacement, where $\delta^t$ is the regrowing factor introduced as follows.

In the regrowing stage, we employ a cosine decay scheduler to gradually reduce the number of regrown channels so that the sub-model converges to the target channel sparsity at the end of training. Specifically, the regrowing factor at $t$-th step is computed as: $\delta_t = \frac{1}{2}\left(1 + \cos\left(\frac{t \cdot \pi}{T_{\max}/\Delta T}\right)\right)\delta_0$, where $\delta_0$ is the initial regrowing factor, $T_{\max}$ denotes the total exploration steps, and $\Delta T$ represents the frequency to invoke the pruning-regrowing steps.

### 4.2.5 Adapting model structures.

The starting model architecture may not have optimal layer distributions. Some layers are more important to the model accuracy and more channels need to be preserved, while some other layers may contain excessive number of channels. To better preserve model accuracy, CHEX also presents an efficient way to adapt the model structure to a more optimal one, by dynamically re-distributing the surviving channels across different layers.

Inspired by [166], we use the learnable scaling factors in the batch normalization (BN) layers [2] [120] to reflect the layer importance. Denote the BN scaling factors of all channels across all layers by $\Gamma = \{\boldsymbol{\gamma}^1, ..., \boldsymbol{\gamma}^L\}$, $\boldsymbol{\gamma}^l \in \mathbb{R}^{C^l}$, and the overall target channel sparsity by $S$. We calculate the layer-wise pruning ratios by ranking all scaling factors in descending order and preserving the top $1 - S$ percent of the channels. Then, the sparsity $\kappa^l$ for layer $l$ is given as:

$$\kappa^l = \big(\sum\nolimits_{j \in [C^l]} \mathbb{1}_{\{\gamma_j^l \leq q(\Gamma, S)\}}\big)/C^l, l \in [L], \tag{4.4}$$

where $\mathbb{1}_{\{\gamma_j^l \leq q(\Gamma,S)\}}$ is 0 if $\gamma_j^l > q(\Gamma, S)$ and 1 if $\gamma_j^l \leq q(\Gamma, S)$. $q(\Gamma, S)$ represents the $S$-th percentile of all the scaling factors $\Gamma$. Accordingly, the number of channels in each layer of the sub-model is obtained as $\lceil (1 - \kappa^l)C^l \rceil$.

[166] relies on LASSO regularization to identify insignificant channels, it extracts the pruned model from a fully pre-trained model in one-shot manner, and the finetuning procedure fixes the architecture without adaptation. In contrast, our structural learning applies the CSS-based pruning algorithm without requiring any sparse regularization, and we propose a repeated pruning and regrowing paradigm as opposed to the pruning-only strategy. We only use the scaling factors for re-allocating the number of channels. While such re-allocation is repeatedly applied at each step to take into account the changing layer importance during model training. Due to the novel regrowing stages which can maintain the structural search space, we can progressively learn the optimal structure by gradually re-distributing channels from the less crucial layers to the more important ones.

### 4.2.6 Theoretical justification

We now provide the convergence guarantee for the CHEX method. Let $F(\mathbf{W}) = \mathbb{E}_{x \sim \mathcal{D}}[f(\mathbf{W}; x)]$ be the loss function of the deep learning task where $x$ is the data following a distribution $\mathcal{D}$ and $\mathbb{E}[\cdot]$ is the expectation. In addition, let $\mathbf{W}_t \in \mathbb{R}^d$ be the model parameter at the $t$-th training

[2] BN applies affine transformations to standardized input feature-maps: $X_{out} = \gamma \tilde{X}_{in} + \beta$, where $\gamma$ / $\beta$ are learnable scaling / shifting factors.

| Tasks | Models | Datasets | Eval metrics |
|---|---|---|---|
| Image classification | ResNet-18/34/50/101, MobileNetV2, EfficientNet | ImageNet (1.28M train data, 1K categories) | Top-1 accuracy |
| Object detection detection | SSD, Mask R-CNN | COCO (118K train data, 80 categories) | Mean average precision |
| Instance segmentation | Mask R-CNN | COCO (118K train data, 80 categories) | Mean average precision |
| 3D segmentation | PointNet++ | ShapeNet (16K shapes, 16 categories, 50 parts) | Mean intersection over union |

Table 4.1: Summary of the tasks, models, and datasets on which we have evaluated the CHEX method.

iteration, and $m_t \in \{0,1\}^d$ be a binary channel mask vector for $t = 1, \cdots, T$. Apparently, quantity $\mathbf{W}_t \odot m_t \in \mathbb{R}^d$ is a sub-model pruned from $\mathbf{W}_t$ where $\odot$ denotes the element-wise product. The following proposition shows that the CHEX will converge to a neighborhood around the stationary solution at rate $O(1/\sqrt{T})$ when learning rate is set properly. The proof is provided in Appendix ???.

**Proposition 1** (Convergence Guarantee of CHEX)**.** *Suppose the loss function $F(\mathbf{W})$ is L-smooth, the sampled stochastic gradient is unbiased and has bounded variance, and the relative error introduced by each mask is bounded, i.e., $\|\mathbf{W} - \mathbf{W} \odot m_t\|^2 \leq \delta^2 \|\mathbf{W}\|^2$ and $\|\nabla F(\mathbf{W}) - \nabla F(\mathbf{W}) \odot m_t\|^2 \leq \zeta^2 \|\nabla F(\mathbf{W})\|^2$ for constants $\delta \in [0,1]$ and $\zeta \in [0,1]$. If learning rate $\eta = \frac{\sqrt{2C_0}}{\sigma\sqrt{L(T+1)}}$ in which $C_0 = \mathbb{E}[F(\mathbf{W}_0)]$, the sub-models obtained by CHEX will converge as follows:*

$$\frac{1}{T+1}\sum_{t=0}^{T}\mathbb{E}[\|\nabla F(\mathbf{W}_t \odot m_t)\|^2] \leq \frac{4\sigma\sqrt{LC_0}}{(1-\zeta)^2\sqrt{T+1}} + \frac{2L^2\delta^2}{(T+1)(1-\zeta)^2}\sum_{t=0}^{T}\mathbb{E}[\|\mathbf{W}_t\|^2]. \tag{4.5}$$

**Remark 1.** *If there is no pruning (i.e., $m_t = \mathbf{1} \in \mathbb{R}^d$) in the training process, it holds that $\delta = 0$ and $\zeta = 0$. Substituting it into (4.5), we find that the CHEX method can converge exactly to the stationary solution, i.e., $\mathbb{E}[\|\nabla F(\mathbf{W}_T)\|^2] \to 0$ as $T$ increases to infinity.*

**Remark 2.** *When a sparse channel mask is utilized, it holds that $\delta \neq 0$ and $\zeta \neq 0$. In this scenario, the mask-induced error will inevitably influence the accuracy of the trained sub-model, i.e., constants $\delta$ and $\zeta$ will influence the magnitude of the second term in the upper bound (4.5).*

### 4.2.7 Experiments

**Setup.** To evaluate the efficacy and generality of CHEX, we experiment on a variety of computer vision tasks with diverse CNN architectures. All experiments run on PyTorch framework based on DeepLearningExamples [191] with NVIDIA V100 GPUs. In Table 4.1, we summarize the deep learning tasks, models, and datasets on which we have evaluated the CHEX method. The CHEX

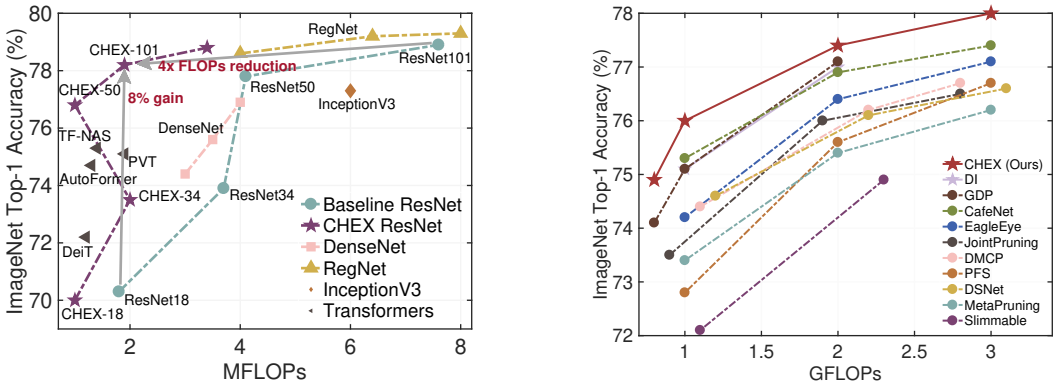
| Method | PT | FLOPs | Top-1 | Epochs | Method | PT | FLOPs | Top-1 | Epochs |
|---|---|---|---|---|---|---|---|---|---|
| ***ResNet-18*** | | | | | ***ResNet-50*** | | | | |
| PFP [152] | Y | 1.27G | 67.4% | 270 | GBN [276] | Y | 2.4G | 76.2% | 350 |
| SCOP [230] | Y | 1.10G | 69.2% | 230 | LeGR [40] | Y | 2.4G | 75.7% | 150 |
| SFP [88] | Y | 1.04G | 67.1% | 200 | SSS [119] | N | 2.3G | 71.8% | 100 |
| FPGM [90] | Y | 1.04G | 68.4% | 200 | TAS [53] | N | 2.3G | 76.2% | 240 |
| DMCP [72] | N | 1.04G | 69.0% | 150 | GAL [155] | Y | 2.3G | 72.0% | 150 |
| **CHEX** | N | 1.03G | **69.6%** | 250 | Hrank [154] | Y | 2.3G | 75.0% | 570 |
| ***ResNet-34*** | | | | | Taylor [182] | Y | 2.2G | 74.5% | - |
| Taylor [182] | Y | 2.8G | 72.8% | - | C-SGD [46] | Y | 2.2G | 74.9% | - |
| SFP [88] | Y | 2.2G | 71.8% | 200 | SCOP [230] | Y | 2.2G | 76.0% | 230 |
| FPGM [90] | Y | 2.2G | 72.5% | 200 | DSA [188] | N | 2.0G | 74.7% | 120 |
| GFS [273] | Y | 2.1G | 72.9% | 240 | CafeNet [222] | N | 2.0G | 76.9% | 300 |
| DMC [63] | Y | 2.1G | 72.6% | 490 | **CHEX-1** | N | 2.0G | **77.4%** | 250 |
| NPPM [62] | Y | 2.1G | 73.0% | 390 | SCP [127] | N | 1.9G | 75.3% | 200 |
| SCOP [230] | Y | 2.0G | 72.6% | 230 | Hinge [149] | Y | 1.9G | 74.7% | - |
| CafeNet [222] | N | 1.8G | 73.1% | 300 | AdaptDCP [303] | Y | 1.9G | 75.2% | 210 |
| **CHEX** | N | 2.0G | **73.5%** | 250 | LFPC [87] | Y | 1.6G | 74.5% | 235 |
| ***ResNet-101*** | | | | | ResRep [48] | Y | 1.5G | 75.3% | 270 |
| SFP [88] | Y | 4.4G | 77.5% | 200 | Polarize [302] | Y | 1.2G | 74.2% | 248 |
| FPGM [90] | Y | 4.4G | 77.3% | 200 | DSNet [140] | Y | 1.2G | 74.6% | 150 |
| PFP [152] | Y | 4.2G | 76.4% | 270 | CURL [172] | Y | 1.1G | 73.4% | 190 |
| AOFP [47] | Y | 3.8G | 76.4% | - | DMCP [72] | N | 1.1G | 74.1% | 150 |
| NPPM [62] | Y | 3.5G | 77.8% | 390 | MetaPrune [169] | N | 1.0G | 73.4% | 160 |
| DMC [63] | Y | 3.3G | 77.4% | 490 | EagleEye [139] | Y | 1.0G | 74.2% | 240 |
| **CHEX-1** | N | 3.4G | **78.8%** | 250 | CafeNet [222] | N | 1.0G | 75.3% | 300 |
| **CHEX-2** | N | 1.9G | **77.6%** | 250 | **CHEX-2** | N | 1.0G | **76.0%** | 250 |

Table 4.2: Results of compressing ResNet with different depth on ImageNet dataset. "PT": require pre-training. "Y": Yes, "N": No.

method introduces few hyper-parameters: we set $\delta_0 = 0.3$ and $\Delta T = 2$ epoch, where $\delta_0$ is the initial regrowing factor, and $\Delta T$ is the number of training iterations between two consecutive pruning-regrowing steps. To keep CHEX simple and generic, the above hyper-parameters are kept constant for our experiments. We set the other training-related hyper-parameters in the default settings and specify them for corresponding tasks. In our results, FLOPs is calculated by counting multiplication and addition as one operation by following [85].

**Image recognition.** For image recognition, we apply CHEX to ResNet [85] with different depths on ImageNet [42]. The baseline ResNet-18/34/50/101 models have 1.8/3.7/4.1/7.6 GFLOPs with 70.3%/73.9%/77.8%/78.9% top-1 accuracy, respectively.

As shown in Table 4.2 and Figure 4.4, CHEX achieves noticeably higher accuracy than the state-of-the-art channel pruning methods under the same FLOPs. For example, compared with MetaPruning [169], DCMP [72] and CafeNet [222], our pruned ResNet-50 model with 4× FLOPs reduction achieves 2.6%, 1.6%, and 0.7% higher top-1 accuracy, respectively. On the other hand, at the same target accuracy, CHEX achieves higher FLOPs reduction. For example, CHEX achieves 4× FLOPs reduction on ResNet-101 model with 77.6% top-1 accuracy, compared to previous best work




(a) Apply CHEX to compress ResNet on ImageNet. (b) Compare CHEX with SOTA pruning methods.

Figure 4.4: Compare CHEX with state-of-the-art channel pruning methods and efficient architectures on ImageNet.

NPPM [62] which only yields 2.2× FLOPs reduction.

The results in Table 4.2 also show an interesting property of our CHEX method. When we search a sub-model with a small FLOPs target, it is better to start our method on a larger model than on a smaller one. For instance, pruning a ResNet-50 model to 1 GFLOPs yields an accuracy 6.6% higher than pruning a ResNet-18 model to 1 GFLOPs. This indicates that CHEX performs training-time structural exploration more effectively when given a larger parametric space. To illustrate this point further, we conduct an additional experiment by applying CHEX to a model with twice the number of channels as the ResNet-50 model, with the goal of reducing its FLOPs to the same level as the original ResNet-50 model. Notably, this sub-model achieves 78.9% top-1 accuracy at 4.1 GFLOPs, which is almost 1% higher than the baseline. This suggests that CHEX has the potential to optimize existing CNNs for higher accuracy.

We also apply CHEX to compress more compact models such as MobileNetV2 and EfficientNet. As shown in Table 4.3, our compressed MobileNetV2 model with 30% FLOPs reduction achieves almost no accuracy loss. With 50% FLOPs reduction, our compressed MobileNetV2 model outperforms previous state-of-the-art channel pruning methods by 0.8∼2.3% accuracy. Similarly, our method achieves noticeably higher accuracy when compressing EfficientNet-B0 compared to previous methods.

**Object detection.** For object detection, we apply CHEX to the SSD model [163] on COCO2017 dataset [158]. Table 4.4 summarizes the performance with different FLOPs reductions. Our pruned model outperforms the baseline model by 0.7% AP (25.9 vs. 25.2) while achieving 2× FLOPs reduction. Compared with previous SOTA channel pruning method [72], our method achieves 1.8% higher AP (25.9 vs. 24.1) with 2× FLOPs reduction. Moreover, our method achieves 4× FLOPs

| Model | Method | FLOPs | Top-1 |
|---|---|---|---|
| MobileNetV2 | Baseline | 300M | 72.2% |
| | LeGR [40] | 220M | 71.4% |
| | GFS [273] | 220M | 71.6% |
| | MetaPruning [169] | 217M | 71.2% |
| | DMCP [72] | 211M | 71.6% |
| | AMC [89] | 210M | 70.8% |
| | PFS [253] | 210M | 70.9% |
| | JointPruning [170] | 206M | 70.7% |
| | **CHEX-1** | 220M | **72.0%** |
| | DMC [63] | 162M | 68.4% |
| | GFS [273] | 152M | 69.7% |
| | LeGR [40] | 150M | 69.4% |
| | JointPruning [170] | 145M | 69.1% |
| | MetaPruning [169] | 140M | 68.2% |
| | **CHEX-2** | 150M | **70.5%** |
| EfficientNet-B0 | Baseline | 390M | 77.1% |
| | PEEL [98] | 346M | 77.0% |
| | **CHEX-1** | 330M | **77.4%** |
| | DSNet [140] | 270M | 75.4% |
| | **CHEX-2** | 270M | **76.2%** |
| | CafeNet-R [222] | 192M | 74.5% |
| | **CHEX-3** | 192M | **74.8%** |

Table 4.3: Results of compressing lightweight CNNs on ImageNet dataset.

reduction with less than 1% AP loss.

**Instance segmentation.** For instance segmentation, we apply CHEX to the Mask R-CNN [83] on COCO2014 dataset. Since Mask R-CNN is a multi-task framework, we evaluate both the bounding box AP for object detection and the mask AP for instance segmentation. As shown in Table 4.4, the models pruned using our method achieve 2× FLOPs reduction without AP loss, even for the challenging instance segmentation task where the model needs to detect all objects correctly in an image while precisely segmenting each instance.

**3D classification/segmentation.** Apart from 2D computer vision problems, we also apply CHEX to compress PointNet++ [200] for 3D shape classification on ModelNet40 [258] dataset and 3D part segmentation on ShapeNet [274] dataset. The results are shown in Table 4.5. On shape classification, the model pruned using our method achieves around 7.5× FLOPs reduction while improving the accuracy slightly compared to the unpruned baseline. On part segmentation, the model pruned using our method achieves 11× FLOPs reduction while maintaining similar mIoU compared with the unpruned baseline.

**Compare with NAS.** We also compare the CHEX method with the state-of-the-art NAS methods [20] on ImageNet, as shown in Figure 4.5. Our results are obtained by applying CHEX to ResNet-50

| Method | FLOPs reduction | AP | $AP_{50}$ | $AP_{75}$ | $AP_S$ | $AP_M$ | $AP_L$ |
|---|---|---|---|---|---|---|---|
| ***SSD object detection*** | | | | | | | |
| Baseline [163] | 0% | 25.2 | 42.7 | 25.8 | 7.3 | 27.1 | 40.8 |
| DMCP [72] | 50% | 24.1 | 41.2 | 24.7 | 6.7 | 25.6 | 39.2 |
| **CHEX-1** | 50% | **25.9** | 43.0 | 26.8 | 7.8 | 27.8 | 41.7 |
| **CHEX-2** | 75% | **24.3** | 41.0 | 24.9 | 7.1 | 25.6 | 40.1 |
| ***Mask R-CNN object detection*** | | | | | | | |
| Baseline [83] | 0% | 37.3 | 59.0 | 40.2 | 21.9 | 40.9 | 48.1 |
| **CHEX** | 50% | **37.3** | 58.5 | 40.4 | 21.7 | 39.0 | 49.5 |
| ***Mask R-CNN instance segmentation*** | | | | | | | |
| Baseline [83] | 0% | 34.2 | 55.9 | 36.2 | 15.8 | 36.9 | 50.1 |
| **CHEX** | 50% | **34.5** | 55.7 | 36.7 | 15.9 | 36.1 | 51.2 |

Table 4.4: Results of compressing SSD and Mask R-CNN on COCO dataset. For objection detection, we evaluate the bounding box AP. For instance segmentation, we evaluate the mask AP.

| Method | #Points | FLOPs reduction | Quality |
|---|---|---|---|
| ***3D shape classification (Quality is accuracy)*** | | | |
| Baseline [200] | 1k | 0% | 92.8% |
| **CHEX** | 1k | 87% | **92.9%** |
| ***3D part segmentation (Quality is class/instance mIoU)*** | | | |
| Baseline [200] | 2k | 0% | 82.5%/85.4% |
| ADMM [205] | 2k | 90% | 77.1%/84.0% |
| **CHEX** | 2k | 91% | **82.3%/85.2%** |

Table 4.5: Results of compressing PointNet++ for 3D point clouds classification on ModelNet40 dataset and segmentation on ShapeNet dataset.

and MobileNetV2. NASNet applies reinforcement learning to search the optimal architecture. OFA firstly trains a supernet, then applies progressive shrinking in four dimensions, including number of layers, number of channels, kernel sizes, and input resolutions, and finally finetunes the obtained sub-models. DARTS introduces learnable architecture parameters to encode the architecture and apply gradient descent to train the architecture parameters. In contrast, our CHEX method only adjusts the number of channels via the repeated pruning and regrowing process, and we do not require training a supernet nor extra finetuning. As shown in Table 4.6, CHEX achieves superior accuracy under similar FLOPs constraints but with significantly less model parameters and training GPU hours than OFA. However, it is important to note that the comparison results are highly dependent on the specific dataset and backbone we chose to apply CHEX. NAS has more comprehensive search space, thus the obtained model may be more optimal, but the training cost may be prohibitive. CHEX has restrictive architecture space (model width), thus is more efficient than search-based methods, but the accuracy is also upper-bounded by the restricted space.

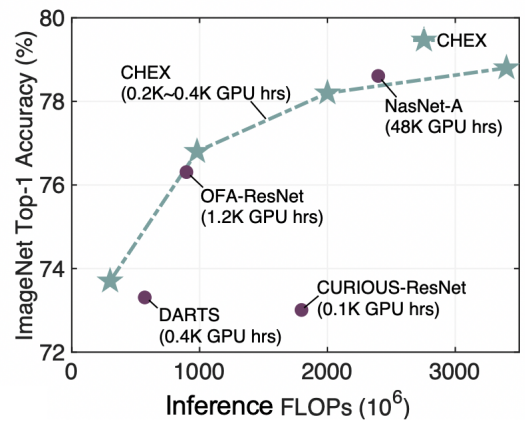



Figure 4.5: Compare CHEX with representative architecture search methods with different turnaround time over a range of inference FLOPs on ImageNet.

| **Method** | **Type** | **FLOPs** | **Params.** | **Top-1** | **Training cost (GPU hours)** |
|---|---|---|---|---|---|
| NASNet-A [306] | reinforcement learning | 2.4G | 10.9M | **78.6%** | 48K |
| **CHEX** | grow-and-prune | 2.0G | 12.8M | 78.2% | 0.2K |
| OFA (ResNet-50) [20, 19] | one-shot NAS | 900M | 14.5M | 76.3% | 1.2K |
| **CHEX** | grow-and-prune | 980M | 7.2M | **76.8%** | 0.3K |
| DARTS [161] | differentiable search | 574M | 4.7M | 73.3% | 0.4K |
| **CHEX** | grow-and-prune | 300M | - | **73.7%** | 0.4K |

Table 4.6: Compare CHEX with representative NAS methods on ImageNet dataset.

### 4.2.8 Analyses

We investigate the effectiveness of different components in CHEX through ablation studies. All the following results are based on pruning ResNet-50 to 1 GFLOPs (4× reduction) on ImageNet dataset.

**Ablation study.** In Table 4.7, we study the effectiveness of different components in CHEX, namely the repeated pruning-regrowing process, the structural learning technique, and the importance sampling. The baseline is one-shot early-stage pruning [275], where the model is pruned by CSS very early after 6% of the total training epochs, and there is no regrowing stage. The sub-model architecture is based on the BN scaling factors and kept fixed in the subsequent training. The periodic pruning and regrowing process repeatedly samples the important channels and prevents the channels from being pruned prematurely. This brings in 0.9% accuracy improvement. When the number of channels in each layer is also dynamically adjusted instead of sticking to the fixed structure determined very early in training, we can obtain a more optimal sub-model structure, which further improves the accuracy by 0.8%. Finally, using the importance sampling strategy described in Eq.(4.3) instead of uniform sampling in the regrowing stages improves the top-1 accuracy to 76%.

| Method | Top-1 |
|---|---|
| Baseline | 73.9% |
| + Periodic pruning and regrowing | 74.8% |
| + Dynamic sub-model structure exploration | 75.6% |
| + Importance sampling-based regrowing | 76.0% |

Table 4.7: Ablation study of different components in CHEX.

| Prune criterion | **BN** [166] | **Magnitude** [141] | **CSS (Ours)** |
|---|---|---|---|
| Pretrain-prune-finetune | 73.1% | 73.8% | 74.6% |
| CHEX | 75.3% | 75.7% | 76.0% |

Table 4.8: Influence of the pruning criterion to the top-1 accuracy in pretrain-prune-finetune framework and our CHEX method.

**Influence of pruning criterion.** We study the influence of pruning criterion to the final accuracy by comparing CSS versus filter magnitude [141] and BN [166] pruning in Table 4.8. As suggested by [141, 173, 182, 103], pruning criterion is an important factor in the traditional pretrain-prune-finetune (PPF) approach. Indeed, when the PPF approach is used, our proposed CSS shows the best accuracy, outperforming magnitude and BN pruning by 0.8% and 1.5%, respectively. On the other hand, we observe that CHEX is more robust to the pruning criterion, as it improves the accuracy for all three criteria and the accuracy gap among them becomes smaller: CSS outperforms magnitude and BN pruning by 0.3% and 0.7%, respectively. We suspect that this is because CHEX can dynamically adjust the channel importance and can recover from sub-optimal pruning decisions.

**Design choices for the regrowing stage.** In Table 4.9, we investigate the impact of different schemes in the channel regrowing stage of CHEX:

(1) Different initialization schemes for the regrown channels affect model quality critically. We have experimented several methods by initializing the regrown channels with random normal distribution [84], zero weights, the exponential moving average (EMA) weights, and most recently used (MRU) weights. The results show that MRU yields the best top-1 accuracy, outperforming the other three initialization schemes by $0.5\% \sim 1.9\%$.

(2) The initial regrowing factor $\delta_0$ also affects the model quality. Intuitively, a large regrowing factor can maintain relatively larger model capacity in the early stage of training, and involve more channels into the exploration steps. As shown, the accuracy is improved by 0.8% as the $\delta_0$ increases from 0.1 to 0.3, at the price of more training cost. However, we did not observe further improvement when the regrowing factor increases from $\delta_0 = 0.3$ to full model size, in which case one step in CHEX consists of pruning to target channel sparsity and regrowing all pruned channels. This suggests that regrowing *a fraction* of the previously pruned channels may be enough for a comprehensive

| | | | | | |
|---|---|---|---|---|---|
| ***Initialization of regrown channels*** | | | | | |
| Design choices | Zero | Random | EMA | MRU | |
| Top-1 | 74.1% | 74.5% | 75.5% | 76.0% | |
| ***Regrowing factor*** | | | | | |
| Design choices | $\delta_0 = 0.1$ | $\delta_0 = 0.2$ | $\delta_0 = 0.3$ | $\delta_0 = 0.4$ | Full |
| Top-1 | 75.2% | 75.6% | 76.0% | 75.9% | 76.0% |
| ***Scheduler for channel regrowing*** | | | | | |
| Design choices | Constant | Linear decay | | Cosine decay | |
| Top-1 | 75.3% | 75.6% | | 76.0 | |
| ***Pruning-regrowing frequency*** | | | | | |
| Design choices | $\Delta T = 1$ | $\Delta T = 2$ | $\Delta T = 5$ | $\Delta T = 10$ | $\Delta T = 20$ |
| Top-1 | 75.2% | 76.0% | 75.8% | 75.3% | 74.9% |

Table 4.9: Compare different design choices in the regrowing stages of the CHEX method.

exploration of the channels if we also aim more training cost savings.

(3) We decay the number of regrown channels in each regrowing stage to gradually restrict the scope of exploration. We compare several decay schedulers, including constant, linear, and cosine. The cosine decay scheduler performs better than the other two schemes by 0.7% and 0.4% accuracy, respectively. With a decay scheduler, the sub-model under training will enjoy gradually decreased computation cost.

(4) The training iterations between two consecutive pruning-regrowing steps, $\Delta T$, also affects the model quality. A smaller $\Delta T$ incurs more frequent pruning-regrowing steps in CHEX, leading to better accuracy in general. We use $\Delta T = 2$ epoch, as it gives the best accuracy.

## 4.3 Omni-dimensional structural learning

### 4.3.1 Background

The transformer models [55, 233, 236] have achieved substantial progress in both language and vision tasks. However, the huge computational cost remain an issue for practical deployment. Moreover, compared to previous MLP or CNN models, transformer models have heterogeneous building structures, making transformer strtuctural learning a challenging task. Each type of structures accounts for a portion of the total computations, thus reducing only one type of them may not effectively accelerate the model. Moreover, each type of structure is only redundant to some extent, excessive pruning may inevitably yield sub-optimal model quality. Although one can sequentially or individually compress each type of structure, the coupling effect among different structures make it hard to decide the optimal order or amount of reduction in sequential or individual compression strategy.

Prior works on compressing transformer models [39, 247, 263] in language tasks place their

focus on replacing the quadratic complexity softmax-attention by linear complexity approximations, because the input has very long sequence in language tasks. However, for vision transformers, the softmax-attention constitutes a small fraction of the total FLOPs, as shown in Table 4.10. We find that the number of attention heads, the number of neurons for feature embedding, and the number of tokens jointly affect the model computation.

ViT models split the input image into a sequence of image tokens. Considering that not all tokens contribute to the final predictions [204] and the high similarity between tokens within a layer [229], recent ViT compression methods [204, 264, 229] adopt unstructured token pruning by removing the redundant and unimportant tokens. Since the self-attention operator can process variable sequence length, these unstructured token pruning methods can achieve practical acceleration. However, to achieve more compelling computational cost reduction, excessive pruning of a single dimension (i.e., sequence length) leads to unacceptable accuracy loss. This motivates our study on how to search an optimal compression policy to reduce the computational cost from multiple dimensions jointly, in order to achieve better computation-accuracy trade-off. In this section, we consider reducing the number of attention heads in MHSA modules, the number of neurons in FFN modules, and the sequence length jointly.

Multi-dimensional ViT compression is challenging. Most of the current pruning algorithms are designed by the unique property of a single dimension, e.g., using the column mean of the attention matrix for sequence reduction [246]. These methods may hardly generalize to other pruning dimensions. Moreover, the large decision space stemming from three integrated dimensions makes it hard to decide how much of each dimension should be compressed.

We firstly propose a omni-potent criterion that is applicable to both structured neuron or head reduction and unstructured sequence reduction. The criterion measures the statistical dependency between the features of a dimension and the output predictions of the model based on the Hilbert-Schimdt norm of the cross-variance operator. Moreover, we formulate the multi-dimensional model compression as an optimization problem to search a jointly optimal policy that maximizes the compressed model's accuracy under a target computational cost. Considering the non-differentiability of the problem and the optimization efficiency, we propose to use Gaussian process (GP) search with expected improvement to estimate the compressed model's accuracy for different pruning policies, and solve the problem by non-linear programming solver. To fit a GP, we need to evaluate the actual accuracy of a small set of sampled pruning policies. We further design a weight sharing mechanism for fast accuracy evaluation without training each compressed model from scratch. When compressing DeiT [233] and T2T-ViT [281] on ImageNet [42], our method reduces $40\% \sim 60\%$ FLOPs and yields

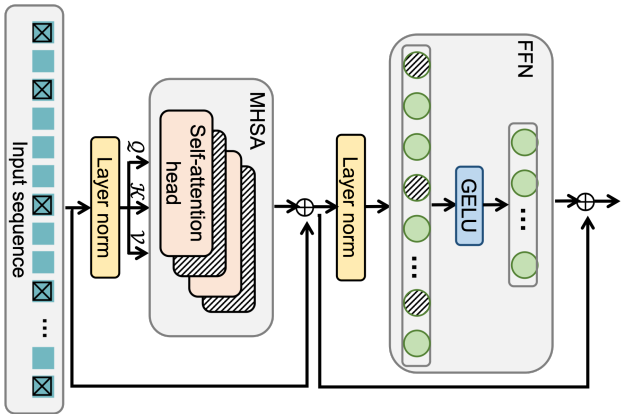



Figure 4.6: Illustration of the MHSA and FFN module in the transformer model. Actual transformer models can have cascaded MHSA+FFN modules for several layers.

$1.3\times \sim 2.2\times$ practical speedup with negligible accuracy drop ($< 0.5\%$).

### 4.3.2 Preliminaries on the transformer model.

The transformer model [55, 236] contains interleaved multi-head self-attention (MHSA) and feed-forward network (FFN) modules. Denote the input features to MHSA and FFN in the $l$-th layer by $X^l, Z^l \in \mathbb{R}^{N\times d}$, where $N$ is the sequence length (i.e., the number of tokens) and $d$ is the embedding dimension. The MHSA module aggregates the input features with input-dependent weighting factors computed from the softmax attention:

$$\text{MHSA}(X^l) = \sum_{h=1}^{H} \text{softmax}(\frac{Q_h K_h^T}{\sqrt{d_h}}) V_h W_h^o, \tag{4.6}$$

where the query, key and value features are computed by $Q_h = X^l W_h^Q, K_h = X^l W_h^K, V_h = X^l W_h^V$ ($W_h^Q, W_h^K, W_h^V \in \mathbb{R}^{d\times d_h}$), $H$ is the number of attention heads, and $W_h^o \in \mathbb{R}^{d_h\times d}$ is the output projection. The FFN module applies fully-connected layers to the embeddings:

$$\text{FFN}(Z^l) = \sigma(Z^l W_1 + b_1) W_2 + b_2, \tag{4.7}$$

where $\sigma$ denotes the non-linear activation function and $W_1, W_2^T \in \mathbb{R}^{d\times d'}$, $b_1 \in \mathbb{R}^{d'}, b_2 \in \mathbb{R}^d$ are the projection matrices and biases. An illustration of the transformer blocks is provided in Figure 4.6.

Notably, transformer has input-adaptive weighting and global receptive field, compared to the static weighting and local receptive field in CNNs. However, the translation equivariance proporty of the CNN models does not hold for transformer, therefore training transformer models usually require large-scale datasets, heavy data augmentations, and regularizations [233].

| Model | Softmax-attention layers $O(N^2 d_h H)$ | Query/Key/Value layers $O(Nd^2)$ | Other fully-connected layers $O(Nd^2)$ | Total |
|---|---|---|---|---|
| DeiT-Small | 0.36G (8%) | 1.39G (30%) | 2.79G (61%) | 4.6G |
| DeiT-Base | 0.72G (4%) | 5.58G (32%) | 11.15G (64%) | 17.5G |

Table 4.10: FLOPs composition of the DeiT transformer models [233]. "MHSA": multi-head self-attention. "FFN": feed-forward network. We use the standard 224×224 input resolution. $N, d, d_h, H$ represents sequence length, embedding dimension, feature dimension per attention head, and number of attention heads, respectively.

Our goal is to accelerate transformer models by omni-dimensional structural learning. Based on our complexity analysis in Table 4.10, we include the embedding dimension (i.e., number of neurons in FNN modules), the number of self-attention heads, and the sequence length into our structural learning space. Note that the transformer depth is also implicitly considered, since pruning all the neurons or self-attention heads equivalently remove the entire MHSA or FFN layer from the model.

### 4.3.3 Joint optimization.

We aim to strike a balance among the different dimensions in our structural learning space so that the compressed model has the best accuracy under a target constraint. We formulate a joint optimization for omni-dimensional structural learning (OSDL):

$$\begin{aligned} \max_{\{\kappa^l,\zeta^l,\nu^l\}_{l=1}^L} \quad & \text{Accuracy}(\{\kappa^l,\zeta^l,\nu^l\}_{l=1}^L), \\ \text{s.t.} \quad & \mathcal{C}(\{\kappa^l,\zeta^l,\nu^l\}_{l=1}^L) \leq \mathcal{T}, \end{aligned} \tag{4.8}$$

where Accuracy$(\cdot)$ gives the accuracy of the compressed model with pruning ratios $\{\kappa^l,\zeta^l,\nu^l\}_{l=1}^L$, $\mathcal{C}(\cdot)$ represents the FLOPs of the model, and $\mathcal{T}$ is the target constraint. $\kappa^l,\zeta^l,\nu^l$ represent the pruning ratio for the embedding dimension, self-attention heads, and the sequence length in $l$-th layer, respectively.

### 4.3.4 Expected improvement (EI) based search.

To solve the OSDL problem (4.8), we resort to Bayesian optimization, as it provides an efficient framework for objectives that may not be differentiable or expressed in a closed-form. We use expected improvement (EI) [125] to estimate the accuracy function in closed form, so that problem (4.8) can be transformed to a simpler and solvable constrained non-linear optimization. The overall flow is provided in Algorithm 5 and Figure 4.7. Our method starts from a pretrained model. Next, we randomly sample an initial population of policies to fit a Gaussian process, where the GP

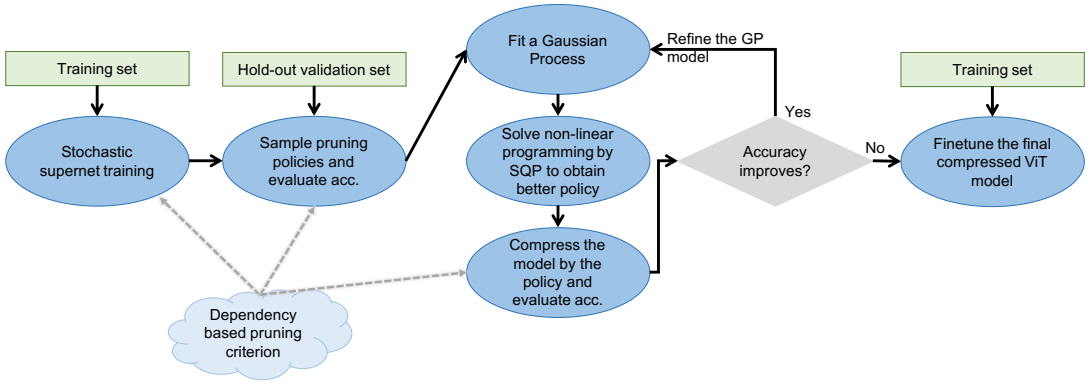



Figure 4.7: Overall flow of the omni-dimensional structural learning (OSDL) with expected improvement based search.

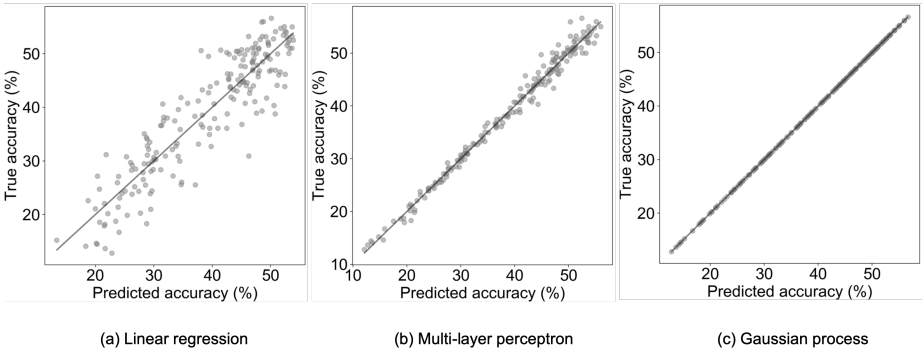



Figure 4.8: Compare GP with other regression methods on fitting the sampled policy-accuracy pairs. GP achieves the minimum MSE error..

model's input is some policy and the output is the corresponding accuracy. We then begin searching more optimal policy by solving an non-linear programming. If the returned policy shows accuracy improvement, we append it to the population and refine the GP model. We iterate the search process until no further accuracy improvement. Finally, we use the most optimal policy to compress the original model and perform finetuning to regain the accuracy. More specific details are introduced next.

We abbreviate the pruning policy as $\omega = \{\kappa^l, \zeta^l, \nu^l\}_{l=1}^{L}$. A Gaussian process is described by:

$$f \sim \mathcal{GP}(\mu(\cdot), k(\cdot, \cdot)), \tag{4.9}$$

where the GP has a mean function $\mu(\omega) = \mathbb{E}[f(\omega)]$ and a covariance kernel $k(\omega, \omega') = \mathbb{E}[(f(\omega) - \mu(\omega))(f(\omega') - \mu(\omega'))]$. We sample $m$ different policies $\Omega = \{\omega_i\}_{i=1}^{m}$ satisfying the constraint $\mathcal{T}$, obtain $m$ compressed models using our pruning algorithm, evaluate their actual accuracy $\mathcal{A}(\omega_i)$ on a hold-out set, and fit the GP model by the set $\{\omega_i, \mathcal{A}(\omega_i)\}_{i=1}^{m}$. In practice, we find that the GP model

can fit the sampled policy-accuracy pairs perfectly, as shown in Figure 4.8. At a new policy $\hat{\omega}$, the distribution of the predicted accuracy at policy $\hat{\omega}$ is given by:

$$\hat{\mathcal{A}} \sim \mathcal{N}(\hat{\mu}, \hat{\sigma}), \tag{4.10}$$

$$\hat{\mu} = \mu(\hat{\omega}) + k(\hat{\omega}, \Omega)k(\Omega, \Omega)^{-1}(\mathcal{A}(\Omega) - \mu(\Omega)), \tag{4.11}$$

$$\hat{\sigma} = k(\hat{\omega}, \hat{\omega}) - k(\hat{\omega}, \Omega)k(\Omega, \Omega)^{-1}k(\Omega, \hat{\omega}). \tag{4.12}$$

The expected improvement in accuracy at this new policy $\hat{\omega}$ is computed in closed form by:

$$\begin{aligned} \mathrm{EI}(\hat{\omega}) &= \mathbb{E}_{\hat{\mathcal{A}} \sim \mathcal{N}(\hat{\mu}, \hat{\sigma})}[\max(\hat{\mathcal{A}} - \mathcal{A}^*, 0)], \\ &= (\mathcal{A}^* - \hat{\mu})\Phi\big((\mathcal{A}^* - \hat{\mu})/\hat{\sigma}\big) + \hat{\sigma}\phi\big((\mathcal{A}^* - \hat{\mu})/\hat{\sigma}\big), \end{aligned} \tag{4.13}$$

where $\Phi, \phi$ represent the CDF and PDF of the standard normal distribution, and $\mathcal{A}^*$ is the accuracy of the best policy in $\Omega$. Therefore, the most promising policy $\omega^*$ to evaluate is the solution to the following non-linear programming:

$$\max_{\hat{\omega}} \ \mathrm{EI}(\hat{\omega}), \quad \text{s.t.} \ \ \mathcal{C}(\hat{\omega}) \leq \mathcal{T}. \tag{4.14}$$

Since both the objective and constraints[3] in (4.14) have closed-form formulas, the problem can be solved by standard constrained optimization solver, and we use sequential quadratic programming (SQP) [130]. We iterate the search process by using the obtained $\omega^*$ and its actual accuracy $\mathcal{A}(\omega^*)$ to refine the GP model, and find the next (more optimal) policy until no accuracy improvement is observed. With the final optimal pruning policy, we apply our dependency based pruning, and finetune the compressed model.

### 4.3.5 Fast accuracy evaluation.

The proposed GP search method requires evaluating the actual accuracy of different compressed models. Instead of training the models with different policies from scratch, we apply weight sharing [74] for fast accuracy evaluation. We pre-train the full model weights $\mathcal{W}$ in a way that the accuracy of any extracetd sub-model, inheriting the weights from the full model, are predictive for the accuracy

[3]FLOPs can be computed by closed-form formula of the pruning ratios

**Algorithm 5:** Omni-Dimensional Structural Learning via Joint Optimization.

**Input**: An $L$-layer model with weights $\mathcal{W} = \{\mathbf{w}^1, ..., \mathbf{w}^L\}$; pre-training iterations $T_{\text{pre}}$; target constraint $\mathcal{T}$; population size $m$; GP search iterations $T_{\text{gp}}$; finetuning iterations $T_{\text{ft}}$; training set $\mathcal{D}_{\text{tr}}$; hold-out validation set $\mathcal{D}_{\text{val}}$ ;

**Output**: Compressed model satisfying the constraint $\mathcal{T}$ and its optimal weights $\mathcal{W}^*$ ;

`/* Pre-training as described in */`

Randomly initialize the model weights $\mathcal{W}$;

**for** *each training iteration* $t \in [T_{pre}]$ **do**

  Sample a mini-batch $(x, y)$ from $\mathcal{D}_{\text{tr}}$, sample a pruning policy $\omega$ from $U_{\mathcal{T}}$, and select weights $\mathcal{W}(\omega)$ by weight sharing ;

  Compute training loss $\mathcal{L}(y|x; \mathcal{W}(\omega))$, backpropagate gradients and update $\mathcal{W}$ ;

`/* GP search as described in */`

Randomly sample $m$ different pruning policies $\{\omega_i\}_{i=1}^m$ satisfying $\mathcal{T}$, get $m$ compressed models (with weights $\mathcal{W}(\omega_i)$), evaluate their actual accuracy $\mathcal{A}(\omega_i)$ on $\mathcal{D}_{\text{val}}$, and fit a GP model with $\Omega = \{\omega_i, \mathcal{A}(\omega_i)\}_{i=1}^m$ ;

**for** *each search iteration* $t \in [T_{gp}]$ **do**

  Solve the non-linear programming Eq.(4.14) by SQP to get pruning policy $\omega_t^*$ ;

  Evaluate the actual accuracy of $\omega_t^*$ on $\mathcal{D}_{\text{val}}$ ;

  Augment $\{\omega_t^*, \mathcal{A}(\omega_t^*)\}$ to $\Omega$ and refine the GP model ;

Compress the model with the optimal policy $\omega_{T_{\text{gp}}}^*$ ;

Finetune the compressed model by $T_{\text{ft}}$ iterations ;

obtained by training them independently. Our pre-training objective is given by:

$$\min_{\mathcal{W}} \ \mathbb{E}_{(x,y)\sim\mathcal{D},\omega\sim U_{\mathcal{T}}}[\mathcal{L}(y|x; \mathcal{W}(\omega))], \tag{4.15}$$

where $\mathcal{D}$ is the training set, $U_{\mathcal{T}}$ is a constrained uniform sampling distribution[4] of pruning policies, and $\mathcal{L}$ is the loss function. $\mathcal{W}(\omega)$ means selecting weights from $\mathcal{W}$ to form the model with policy $\omega$. When reducing the embedding dimensions or self-attention heads, we select the important weights based on our local optimization metrics. Reducing the sequence length does not require removing weights from the model, thus all weights can be shared for different sequence lengths. Due to this weight sharing mechanism, we only need to train one set of weights and directly evaluate the accuracy of different pruning policies by inheriting weights from the full model. This takes much less time than training each compressed model from scratch and makes our GP search efficient.

### 4.3.6 Dependency based pruning criterion

The joint optimization searches for the optimal policy on how much to reduce from different dimensions (neurons, heads, tokens) in each transformer block. Within each dimension, we use a dependency based criterion to evaluate the redundancy, for example, which specific neurons to reduce in a layer. Intuitively, the unimportant features contribute least to the output predictions. In other words, the

[4]We sample the pruning policy repeatedly until the compressed model FLOPs satisfies the constraint $\mathcal{T}$.

output of the model has weak dependency on the unimportant features. Thus, our pruning criterion is based on the statistical dependency between the features and the output predictions of the model.

Denote the random vector of the features by $Z$ and the random vector of the model outputs by $Y$. Let $P_{Z,Y}$ be the joint distribution between the two random variables. To measure the dependence between $Z$ and $Y$, we use the cross-covariance operator [9] defined as:

$$C_{zy} := \mathbb{E}_{zy}[(\Phi(z) - \mu_z) \otimes (\Psi(y) - \mu_y)], \tag{4.16}$$

where $\Phi$ (or $\Psi$) represents a kernel mapping from the feature space (or the model output space) to a reproducing kernel Hilbert space (RKHS), with mean vector $\mu_z$ (or $\mu_y$). $\otimes$ denotes the tensor product. To summarize the degree of dependence between $Z$ and $Y$, we use the Hilbert-Schmidt norm of the cross-covariance operator $C_{zy}$, which is denoted by $\|C_{zy}\|^2_{HS}$ and is computed by the trace of $C_{zy}C_{zy}^T$. $\|C_{zy}\|^2_{HS}$ can characterize the independence between two random variables:

**Theorem 1** ($C_{zy}$ and Independence)**.** *Given RKHSs with characteristic kernels. Then, $\|C_{zy}\|^2_{HS} = 0$ if and only if $Z$ and $Y$ are independent.*

Characteristic kernels, such as Gaussian RBF kernel $k(x, x') = \exp\big(-\|x - x'\|_2^2/(2\sigma^2)\big)$, allows us to measure an arbitrary mode of dependence (including non-linear dependence) between $Z$ and $Y$. Features with high $\|C_{zy}\|^2_{HS}$ value have high dependency with the outputs of the model, indicating that the features have considerable influence to the output predictions, thus they should be retained.

To use the dependency criterion in practice, we need an empirical estimate from a batch of training samples. Denote the kernel functions by $k(z, z')$ and $l(y, y')$. Let $\mathbf{K}, \mathbf{L}$ be the Gram matrices ($\mathbf{K}_{i,i'} = k(z_i, z_{i'})$, $\mathbf{L}_{i,i'} = l(y_i, y_{i'})$) computed over the features and model outputs of $B$ training samples. An empirical estimator of $\|C_{zy}\|^2_{HS}$ is given by:

$$\widehat{\|C_{zy}\|^2_{HS}} := (B-1)^{-2}\,\mathrm{tr}(\mathbf{KCLC}), \tag{4.17}$$

where $\mathbf{C} = \mathbf{I}_B - (1/B)\mathbf{1}_B\mathbf{1}_B^T$ is the centering matrix ($\mathbf{I}_B, \mathbf{1}_B$ represent identity matrix and all-ones vector) and $\mathrm{tr}(\cdot)$ is the matrix trace operation. We show that this empirical estimator converges sufficiently:

**Theorem 2** (Bound on $\widehat{\|C_{zy}\|^2_{HS}}$.)**.** *Assume $k$ and $l$ are bounded almost everywhere by 1, and are non-negative. Then, with constants $\alpha^2 > 0.24$ and $C$, for $U > 1$ and all $\delta > 0$, with probability at*

*least* $1-\delta$ *for all* $P_{ZY}$*, we have*

$$|\widehat{\|C_{zy}\|_{HS}^2} - \|C_{zy}\|_{HS}^2| \leq \sqrt{\frac{log(6/\delta)}{\alpha^2 U}} + \frac{C}{U} \tag{4.18}$$

Our proposed dependency based pruning criterion can be applied to prune different dimensions, including attention heads, neurons, and sequence length, introduced next:

**FFN neuron reduction.** We prune neurons in the intermediate layer of FFN modules. Denote the features from the intermediate layer in the $l$-th FFN by $Z^l \in \mathbb{R}^{B\times N\times d'}$, where each neuron $j \in [d']$ has features $Z^l_{:,:,j} \in \mathbb{R}^{B\times N}$. Given a neuron pruning ratio $\kappa^l$, we retain $\lceil(1-\kappa^l)d'\rceil$ important neurons. We compute the dependency score for each neuron in the layer by $\psi^l_j = \mathrm{tr}(\mathbf{K}^l_j\mathbf{CLC})$, where $\mathbf{K}^l_j \in \mathbb{R}^{B\times B}$ is the Gram matrix defined over $Z^l_{:,:,j}$, i.e., $[\mathbf{K}^l_j]_{i,i'} = k(Z^l_{i,:,j}, Z^l_{i',:,j})$. Then, we rank the dependency scores in descending order, and identify the important neurons by $\mathrm{ArgTopK}(\{\psi^l_1 ... \psi^l_{d'}\}; \lceil(1-\kappa^l)d'\rceil)$, which gives the neuron indices with the top $\lceil(1-\kappa^l)d'\rceil$ dependency scores. The remaining bottom-ranking neurons are pruned.

**Attention head reduction.** Denote the output features of the self-attention operator in the $l$-th MHSA by $Z^l \in \mathbb{R}^{B\times N\times H\times d_h}$. Given a head pruning ratio $\zeta^l$, the head pruning procedure is similar to neuron pruning, except that each head $h \in [H]$ has output features $Z^l_{:,:,h,:} \in \mathbb{R}^{B\times N\times d_h}$. To construct the feature Gram matrix, we perform mean pooling along the embedding dimension to obtain $\tilde{Z}^l_{:,:,h} = \mathrm{mean}(Z^l_{:,:,h,:}; \mathrm{dim}{=}\text{-}1)$ (-1 means the last tensor dimension), which has a size of $\mathbb{R}^{B\times N}$. Then, we compute the Gram matrix $\mathbf{K}^l_h$ over $\tilde{Z}^l_{:,:,h}$ and the head dependency score $\psi^l_h$. We keep those heads with the top $\lceil(1-\zeta^l)H\rceil$ dependency scores and prune the other bottom-ranking heads.

**Token sequence reduction.** To achieve unstructured sequence reduction, we insert token selection layer (TSL) (no extra parameters) after the MHSA module and before the FFN module at each transformer layer. Let $Z^l \in \mathbb{R}^{B\times N\times d}$ be the input features to $l$-th TSL. Given a sequence reduction ratio $\nu^l$, TSL outputs the selected $\lceil(1-\nu^l)N\rceil$ important tokens from $Z^l \in \mathbb{R}^{B\times N\times d}$ according to indices $\mathrm{ArgTopK}(\{\psi^l_1 ... \psi^l_N\}; \lceil(1-\nu^l)N\rceil)$. That is, TSL extracts a sub-tensor from the input features, the output has size $\mathbb{R}^{B\times\lceil(1-\nu^l)N\rceil\times d}$. The token importance scores $\psi^l_n, n \in [N]$ are computed based on the Gram matrix $\mathbf{K}^l_n$ defined over the token features $Z^l_{:,n,:} \in \mathbb{R}^{B\times d}$. After TSL, the sequence length in subsequent layers becomes $\lceil(1-\nu^l)N\rceil$. And the subsequent TSLs select tokens from what are preserved by the previous TSLs.

| Tasks | Models | Datasets | Evaluation metrics |
|---|---|---|---|
| Image classification | DeiT-Small/Base<br>T2T-ViT-14 | ImageNet<br>(1.28M train data, 1K categories) | Top-1<br>accuracy |
| Object detection | RetinaNet w/ PVT backbone | COCO<br>(118K train data, 80 categories) | Mean<br>average precision |

Table 4.11: Summary of the tasks, models, and datasets on which we have evaluated the ODSL method.

### 4.3.7 Experiments

**Setup.** We conduct experiments on the ImageNet dataset [42] with representative transformer models: DeiT [233], T2T-ViT [281], which were also used by previous methods [204, 30, 193, 264, 229]. All experiments run on PyTorch framework with Nvidia A100 GPUs. We pre-train the models from scratch with Eq.(4.15), and follow the same training hyper-parameters from [233]. We conduct Gaussian process search to obtain the optimal pruning policy. We primarily use FLOPs as the computation constraints, and set the target to 40% and 60% FLOPs reductions. The size of the initial population to fit a GP model is 100, and the GP search runs for 100 iterations. We randomly sample 50k images (50 images per class) from the training set of ImageNet for accuracy evaluation during GP search. Our GP search took less than 1 hour on a single A100 GPU for all listed experiments. Based on the optimal policy, we compress the pre-trained models and perform finetuning with the same strategy as [233, 229]. For object detection, we use the RetinaNet [157] model with PVT-Small [248] backbone on COCO2017 dataset [158]. Following [248], models are trained on COCO train2017 (118k images) and evaluated on val2017 (5k images). We use the same training hyper-parameters as [248] to finetune the compressed model: AdamW optimizer with a batch size of 16, an initial learning rate of $1 \times 10^{-4}$, standard $1\times$ training schedule (12 epochs), learning rate is decayed by 10 at epoch 8 and 11. The training image is resized to have a shorter side of 640 pixels, and during testing the shorter side of the input image is fixed to 640 pixels. A summary of tasks, models, and datasets that we evaluate our ODSL on is provided in Table 4.11.

**Image classification.** Compared methods include token pruning methods (DynamicViT [204], IA-RED$^2$ [193], PatchSlim [229], Evo-ViT [264]), weight pruning methods (VTP [301], S$^2$ViTE [30]), unified compression framework UVC [279], and NAS-based pruning method [78]. The results are shown in Table 4.12. ODSL achieves noticeably higher accuracy than previous methods under same FLOPs. For example, our pruned DeiT-S model with 37% FLOPs reduction outperforms DynamicViT and Evo-ViT by 0.6% and 0.5% accuracy, respectively. On the other hand, at the same target accuracy, our method achieves higher FLOPs reduction. For example, our pruned DeiT-B model yields 60%

| Method | Top-1 | Top-1 drop | FLOPs | FLOPs reduction |
|---|---|---|---|---|
| ***DeiT-Small model*** | | | | |
| Baseline [233] | 79.8% | - | 4.6G | - |
| SPViT [78] | 78.3% | 1.5% | 3.3G | 29% |
| IA-RED$^2$ [193] | 79.1% | 0.7% | 3.1G | 32% |
| S$^2$ViTE [30] | 79.2% | 0.6% | 3.1G | 32% |
| Evo-ViT [264] | 79.4% | 0.4% | 2.9G | 37% |
| DynamicViT [204] | 79.3% | 0.5% | 2.9G | 37% |
| **ODSL (Ours)** | **79.9%** | -0.1% | 2.9G | 37% |
| UVC [279] | 78.4% | 1.4% | 2.4G | 48% |
| **ODSL (Ours)** | **79.3%** | 0.5% | 1.8G | 60% |
| ***DeiT-Base model*** | | | | |
| Baseline [233] | 81.8% | - | 17.5G | - |
| VTP [301] | 81.3% | 0.5% | 13.8G | 22% |
| IA-RED$^2$ [193] | 80.3% | 1.5% | 11.8G | 33% |
| S$^2$ViTE [30] | 82.2% | -0.4% | 11.8G | 33% |
| SPViT [78] | 81.6% | 0.2% | 11.7G | 33% |
| Evo-ViT [264] | 81.3% | 0.5% | 11.7G | 33% |
| DynamicViT [204] | 81.3% | 0.5% | 11.2G | 36% |
| **ODSL (Ours)** | **82.3%** | -0.5% | 11.2G | 36% |
| UVC [279] | 80.6% | 1.2% | 8.0G | 55% |
| **ODSL (Ours)** | **81.5%** | 0.3% | 7.0G | 60% |
| ***T2T-ViT-14 model*** | | | | |
| Baseline [281] | 81.5% | - | 4.8G | - |
| PatchSlim [229] | 81.1% | 0.4% | 2.9G | 40% |
| **ODSL (Ours)** | **81.7%** | -0.2% | 2.9G | 40% |

Table 4.12: Compare ODSL versus baselines and previous transformer model compression methods on ImageNet. Negative "Top-1 drop" means that the accuracy improves over the baseline after ODSL.

| **Model** | DeiT-S | | DeiT-B | | T2T-ViT-14 | |
|---|---|---|---|---|---|---|
| (**FLOPs reduction**) | Full (0%) | Pruned (37% / 60%) | Full (0%) | Pruned (36% / 60%) | Full (0%) | Pruned (40%) |
| **Top-1(%)** | 79.8 | 79.9 / 79.3 | 81.8 | 82.3 / 81.5 | 81.5 | 81.7 |
| **Throughput (img/s)** | 2773 | 4050 / 5523 | 1239 | 1792 / 2649 | 1940 | 2527 |

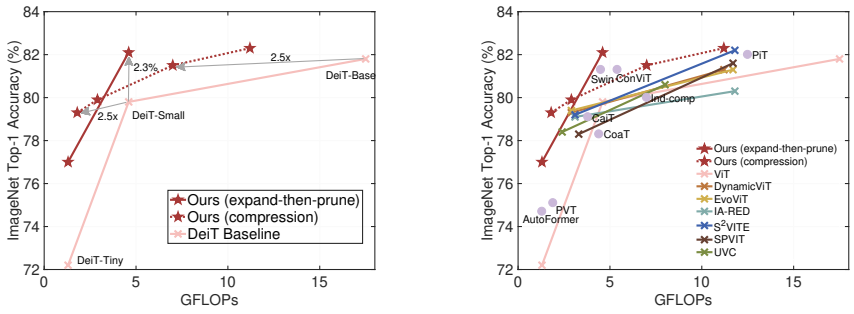
Table 4.13: Compare the throughput of our compressed models with baseline models, measured on one Nvidia A100 GPU.

FLOPs reduction with 81.5% top-1 accuracy, compared to DynamicViT and Evo-ViT yielding less than 36% FLOPs reduction. These results clearly evidence the advantage of omni-dimensional structural learning on models that have heterogeneous modules like transformer. We also compare the throughput of our compressed models over baselines on a single Nvidia A100 GPU with a fixed batch size of 256. A shown in Table 4.13, our compressed models achieve $1.3\times \sim 2.2\times$ throughput improvement without significant accuracy loss.

Apart from compressing the model for faster inference, ODSL can improve existing models for higher accuracy, as shown in Table 4.14. More exactly, we apply ODSL to a scaled-up DeiT-Ti model (width scaled by $2\times$), with the goal of reducing its FLOPs to the same level as the original DeiT-Ti. Notably, the obtained model achieves 77% top-1 accuracy at 1.3 GFLOPs, outperforming the original DeiT-Ti (trained with longer training epochs) by 3.1%. Same observation holds on DeiT-S, where we

| Method | ∼1GFLOPs | | ∼4GFLOPs | |
|---|---|---|---|---|
| | **Top-1** | **FLOPs** | **Top-1** | **FLOPs** |
| Baseline | 72.2% | 1.3G | 79.8% | 4.6G |
| $\text{Baseline}_{2\times\text{training}}$ | 73.9% | 1.3G | 81.0% | 4.6G |
| ***Model compression*** | | | | |
| Manifold [124] | 75.1% | 1.3G | 81.5% | 4.6G |
| UP-DeiT [277] | 75.8% | 1.3G | 81.6% | 4.6G |
| NViT-DeiT [269] | 76.2% | 1.3G | 82.1% | 4.6G |
| ***NAS*** | | | | |
| GLiT [25] | 76.3% | 1.4G | 80.5% | 4.4G |
| AutoFormer [28] | 74.7% | 1.3G | 81.7% | 5.1G |
| **ODSL (Ours)** | **77.0%** | 1.3G | **82.1%** | 4.6G |

Table 4.14: Compare ODSL with NAS and other model compression strategies (e.g., knowledge distillation) on ImageNet.




(a) Mulit-dimensional compression of DeiT models on ImageNet.

(b) Compare with state-of-the-art transformers and pruning methods.

Figure 4.9: Compare multi-dimensional ViT compression with state-of-the-art transformers and pruning methods.

achieve 82.1% top-1 accuracy, improving the baseline by 1.1%. These results suggest that heavily compressed large models can achieve higher accuracy than directly training small models. We also compare ODSL results with NAS methods [28, 25] thatt are designed for transformer models. In Table 4.14, our method achieves higher accuracy than searching architecture from scratch.

**Object detection.** PVT-Small is an efficient transformer models. It has progressive sequence shrinking strategy and also performs embedding dimension shrinking in the self-attention modules Accordingly, we modify our sequence reduction strategy for PVT-Small. The token selection layer no longer discard the unimportant tokens. Instead, they are directly skipped without participating any computations. The selected important tokens participate the computations in the MHSA and FFN modules, and the output will be concatenated with the skipped unimportant tokens. The results are shown in Table 4.15. Our compressed model achieves almost the same AP values compared to the baseline model, while yielding 30% FLOPs reduction.

| Method | FLOPs reduction | AP | $AP_{50}$ | $AP_{50}$ | $AP_S$ | $AP_M$ | $AP_L$ |
|---|---|---|---|---|---|---|---|
| PVT-Small [248] | 0% | 38.7 | 59.3 | 40.8 | 21.2 | 41.6 | 54.4 |
| **ODSL (Ours)** | 30% | 38.6 | 58.8 | 40.6 | 21.0 | 41.5 | 54.4 |

Table 4.15: Results of RetinaNet with PVT-Small backbone on COCO2017 data for object detection task using ODSL.

| Model | Neuron | Head | Sequence | Top-1 | FLOPs |
|---|---|---|---|---|---|
| DeiT-B | - | - | - | 81.8% | 17.5G |
| | ✓ | - | - | 79.4% | 7.4G |
| | - | - | ✓ | 79.5% | 7.0G |
| | ✓ | ✓ | - | 80.4% | 7.1G |
| | ✓ | - | ✓ | 80.7% | 7.0G |
| | - | ✓ | ✓ | 80.2% | 7.0G |
| | ✓ | ✓ | ✓ | **81.5%** | 7.0G |

Table 4.16: Ablation study on the effectiveness of multi-dimensional ViT compression. "✓" means that pruning is conducted along the corresponding dimension, while "−" means no pruning along the dimension. In the last row, jointly pruning along the neuron, head and sequence dimensions achieves the best accuracy. Results are obtained with DeiT-B model on ImageNet.

### 4.3.8 Analyses

**Ablation study.** We compare OSDL with neuron, head or token pruning individually for DeiT-B on ImageNet in Table 4.16. Firstly, self-attention head pruning [181, 239] alone cannot yield significant FLOPs reduction, since the FLOPs of all the MHSA modules only account for 40% of the total FLOPs. Secondly, excessive pruning of whichever structures causes unacceptable accuracy loss, even for the fine-grained token pruning. In contrast, ODSL (last row of Table 7.7) achieves more FLOPs reduction with better accuracy. Searching the optimal policy to balance the FLOPs reduction from different structures in the model is of vital importance if we aim to achieve significant acceleration.

**Dimensions are complementary to each other.** In Figure 4.10(left), we visualize the searched policy by plotting the layer-wise pruning ratios of neurons, attention heads, and tokens on the DeiT-B model. We observe that most of the neuron and head (structured pruning dimensions) come from the shallower layers, while most of the token pruning (unstructured pruning dimension) occur at deeper layers. Thanks to the complementary nature across different pruning dimensions, we can reduce the redundancy from all layers of the model without compromising the model accuracy too much.

**Heuristics obtained from OSDL.** We also find that for transformer models, deeper layers tend to have more redundancy, reflected by the increased layer-wise FLOPs reduction in Figure 4.10(right). This suggests that the uniform FLOPs design principle (i.e., every transformer block has the same FLOPs) in existing transformer models may not be necessary.

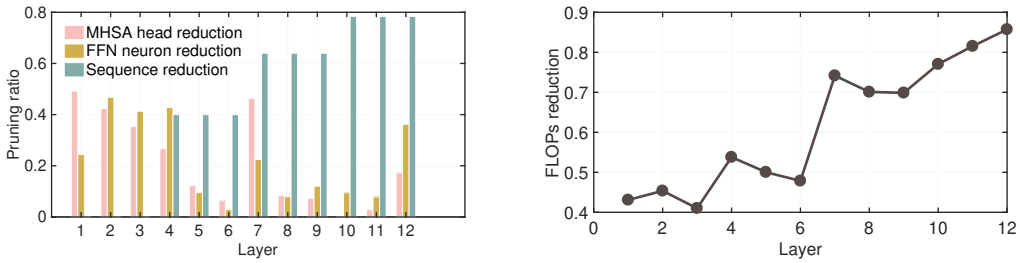



Figure 4.10: **Left**: Layer-wise pruning ratios of neurons, attention heads, and tokens in our search ODSL policy. **Right**: Layer-wise FLOPs reduction. Results are obtained by using DeiT-B model on ImageNet.

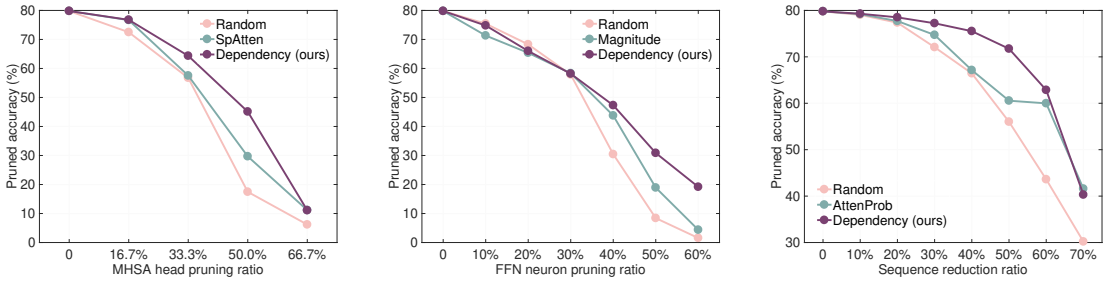



Figure 4.11: Compare our dependency criterion versus other metrics for pruning each dimension individually. The plots show the pruned accuracy (w/o finetuning) versus pruning ratios along each dimension. Results are obtained with DeiT-S on ImageNet.

**Advantage of dependency-based criterion.** In Figure 4.11, we compare our dependency criterion with previous metrics for a specific pruning dimensions: SpAtten [246] for attention head pruning, attention probability [246] (AttenProb) for. token pruning, and row-wise norm of the weight matrix (Magnitude) for neuron pruning. All criteria (including random selection) perform well when the pruning rate is small ($< 20\%$), suggesting that the redundancy indeed exists in each dimension. However, our dependency based pruning achieves relatively higher accuracy at larger pruning rates. In Figure 4.13, by visualizing the attention-maps produced by all the heads in DeiT-B model, we observe that dependency based pruning indeed removes the redundant heads. In Figure 4.12, by visualizing the attention-maps produced by the top-ranking and bottom-ranking heads in the last block of the DeiT-B model, we see that our proposed dependency criterion can identify the heads that are more important to the model prediction.

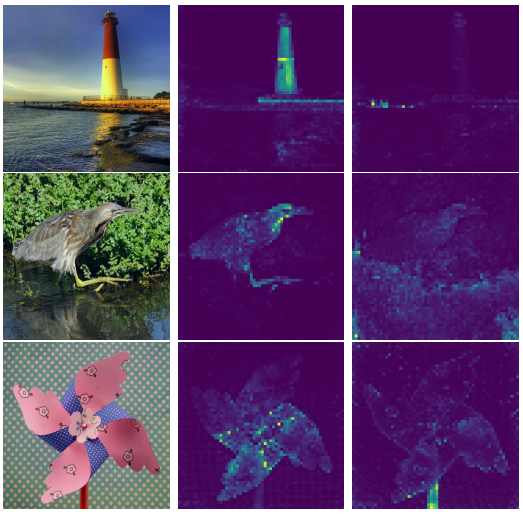


**Image** **High dep. head** **Low dep. head**

Figure 4.12: Examples of the attention-maps produced by top-ranking and bottom-ranking attention head in the last block of DeiT-B model for difference input images.

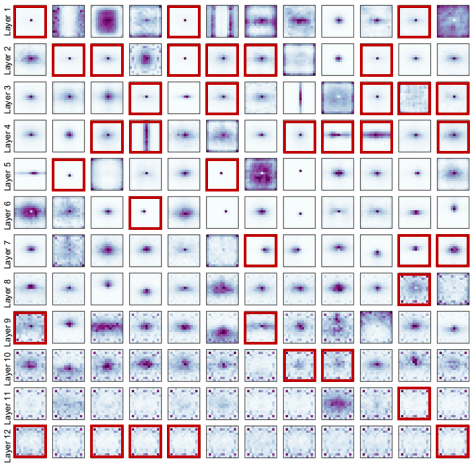



Figure 4.13: Attention-maps (averaged over 256 images) produced by all heads in the DeiT-B model. Red box means the head is pruned based on our dependency criterion. Number of pruned heads follow the policy in Figure 4.10.

**GP search process.** is visualized in Figure 4.14. Our GP search firstly randomly samples 100 populations (pruning policies that satisfy the computation target) to fit GP model, which are shown on the left of the figure. On the right, the plot shows the validation accuracy of the pruning policy obtained by solving Eq.(4.14) at each search iteration. As the search process iterates, the obtained pruning policy gradually improves in terms of the top-1 accuracy until convergence. And the final policy achieves notably higher accuracy than the best result in the initial random sampling.

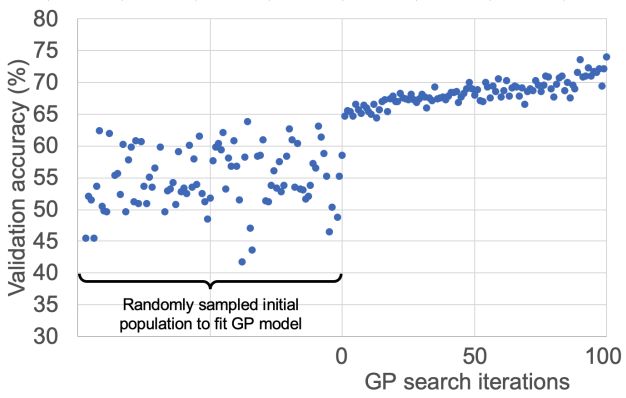



Figure 4.14: GP search process with DeiT-B model on ImageNet by plotting the validation accuracy over search iterations.

## 4.4 Conclusion

In this chapter, we have introduced two approaches for neural structural learning to improve the model efficiency in both training and inference.

The first CHEX method unifies pruning (regressive based) and regrowing (progressive) into one framework CHEX (1) dynamically adjusts the channel importance based on a periodic pruning and regrowing process, which prevents the important channels from being prematurely pruned; (2) dynamically re-allocates the number of channels under a global sparsity constraint to search for the optimal sub-model structure during training. We design the components in the pruning and regrowing process by proposing a column subset selection based criterion for pruning and a channel orthogonality based importance sampling for regrowing. This enables us to obtain a sub-model with high accuracy in only one training pass from scratch, without pre-training a large model or requiring extra fine-tuning. Experiments on a wide range of deep learning tasks demonstrate that CHEX can effectively reduce the computation demands of diverse deep architectures while achieving superior accuracy compared to the state-of-the-art methods.

The second OSDL approach can successfully compress more complicated architectures, e.g., the Transformer, which contain heterogeneous building structures. We propose OSDL with joint optimization, which compresses the attention heads, neurons, and tokens jointly in the transformer model. We propose an omni-potent criterion based on the feature-output dependency to rank the feature importance in individual module. Then, we learn the optimal compression policy by casting multi-dimensional compression as an constrained non-linear optimization and using Gaussian process search with expected improvement to solve it. Our results on ImageNet and COCO datasets using various transformer models outperform previous state-of-the-art model compression and NAS-on-Transformer approaches.

# Chapter 5

# Dynamic Neural Networks

## 5.1 Prologue

With the emergence of many successful deep learning models and the neural architecture search methods, we are able to train deeper, more accuracy and more efficient models to continuously advance the performance on different tasks. However, most of the current deep models perform inference in a static manner: the architecture and the model parameters are fixed once trained. This may limit their representation power and efficiency.

Dynamic neural networks, however, can adapt the model structures parameters to the inputs, and thus have several advantages over the static models. Dynamic neural networks can allocate computations on demand at test time by sparsely activating part of the network conditioned on the input. Therefore, the model can control the model capacity and inference cost based on the hardness of different inputs. Due to the input-adaptive model parameters, dynamic neural network enjoy enlarged parameter space and thus improved representation ability. We will show that the model capacity is greatly boosted by applying input-conditioned weights with very little increase of the actual parameters or FLOPs. Dynamic neural networks can inherit existing architecture design, optimization and data processing principles, and can be applied seamlessly to a wide range of applications.

In this chapter, we introduce two dynamic neural network approaches, covering both dynamic parameters and dynamic architectures. In section 5.2, we propose a parameter efficient dynamic convolution operator (dubbed as PEDConv) that learns to reparameterize the base convolution weights in both spatial and channel dimensions, through a input-dependent tensor decomposition.

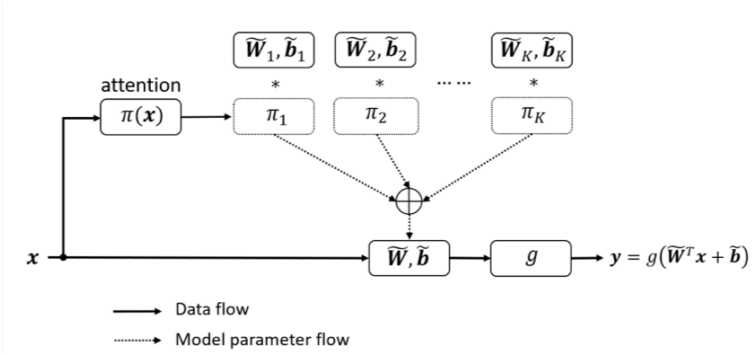
PEDConv significantly boosts the accuracy when substituting standard convolutions on a plethora of prevalent deep learning tasks, including ImageNet classification, COCO object detection, ADE20K semantic segmentation, and adversarial robustness. Meanwhile, PEDConv almost maintain the number of parameters and the computational cost compared to the baseline models. In section 5.3, a dynamic architecture with input-conditioned spatial resolution and architecture width is proposed. To achieve dynamic spatial resolution, we propose to perform convolution on scaled-down feature-maps where the down-scaling factor is adaptive to different input images. To achieve dynamic architectural width, we introduce a low-cost channel saliency predictor for each convolution to dynamically skip the computation of unimportant channels based on the Gumbel-Softmax. To better capture the feature-maps information and facilitate input-adaptive decision, we employ classic image processing metrics, *e.g.,* Spatial Information, to guide the saliency predictors. The proposed method can be readily applied to a variety of SR deep networks and trained end-to-end with standard super-resolution loss, in combination with a sparsity criterion.

## 5.2 Dynamic parameters

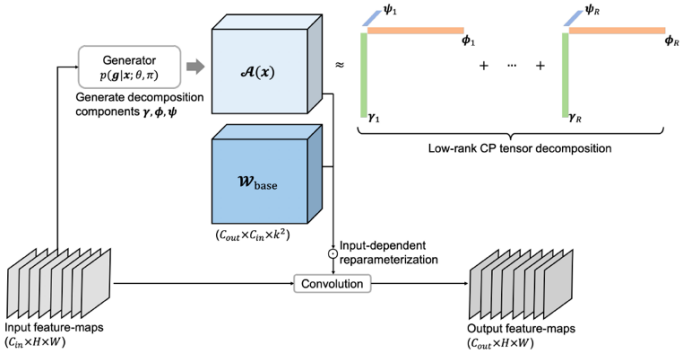
### 5.2.1 Convolution with input-adaptive parameters

The powerful representation ability of deep CNNs rely on using the different convolution kernels to extract diverse information across channels and layers. However, current model designs apply the same set of weights to process different input images, which may limit the ability to adapt to diverse visual patterns. More effective information are captured by increasing the model depth and width, causing high computational cost and large memory footprint. In contrast, neural networks with dynamic parameters adapt the model weights to different input examples. Given an input $x$, the output of a model with dynamic parameters is represented by $y = \mathcal{F}(x, \Theta|x)$, meaning that the model weights $\Theta$ are conditioned on the input $x$. Dynamic parameter designs can effectively improve the representation power of the deep models with negligible increase of computational cost.

Previous works on dynamic parameters [37, 267] use $K$ parallel convolution weights $\{\mathbf{W}_k\}_{k=1}^{K}$ for each layer in the model. These weights are linearly combined $\mathbf{W}(\mathbf{x}) = \sum_k \alpha_k(\mathbf{x}) \cdot \mathbf{W}_k$ for each input example $\mathbf{x}$, based on the input-dependent attention $\boldsymbol{\alpha}(\mathbf{x})$. These aggregation-based methods can increase the model capacity while maintaining the inference efficiency, but also face challenges in parameter/storage inefficiency and learning difficulty: (1) The use of $K$ parallel weights for each layer brings in considerable parameter and storage overheads ($K = 32$ in [267]); (2) the over-




(a) Previous dynmiac parameters methods [37, 267]




(b) Proposed PEDConv

Figure 5.1: Comparing the proposed PEDConv with previous methods on dynamic parameters.

parameterized model is prone to overfitting, thus requires meticulously designed training techniques and heavy regularizations [267]; (3) the scalar attention $\alpha_k$ multiplied to the convolution weight $\mathbf{W}_k$ imposes optimization challenge: if the attention value is too small, the corresponding weight will be insufficiently trained. Hence, existing method [37] applies various normalization techniques to the attention values for training models with dynamic parameters.

To rectify the above limitations, we propose a novel parameter efficient dynamic convolution called PEDConv to achieve dynamic parameters in current deep models. A comparison with previous methods is provided in Figure 5.1. PEDConv dynamically reparameterizes each layer's weights along all dimensions, both spatial-wise and channel-wise, for different inputs. The core component of PEDConv is a Canonical Polyadic tensor decomposition-based reparameterization applied to the base weights[1], where the low-rank decomposition components are conditioned on the input. PEDConv can capture both generic information shared among inputs by optimizing the base weights, and input specification by learning the dynamic reparameterization. To generate the input-dependent decomposition components, we propose to learn a conditional distribution of the input examples, together with mutual information maximization between the generated components and input to

[1]Base weights refer to parameters shared among inputs, while dynamic parameters are input-dependent.

enforce the input-dependency. PEDConv is applied to substitute convolutions in diverse CNN architectures including ResNet, MobileNet, EfficientNet with significant accuracy improvements and negligible computation cost increase. To demonstrate the generality of PEDConv, we conduct experiments on a wide variety of deep learning tasks including classification on ImageNet, object detection on COCO, and semantic segmentation on ADE20K. PEDConv achieves superior accuracy while requiring considerably fewer parameters and FLOPs compared to the state-of-the-art methods. For example, PEDConv applied to ResNet-50 achieves 80.5% top-1 accuracy on ImageNet, improving previous best dynamic parameter approach by 1.9%. When we extend the sample-wise dynamic PEDConv to spatial-wise dynamic model variant, we achieve 79.3% top-1 accuracy on ResNet-50 while reducing 44% FLOPs compared to the baseline model. Moreover, we find that PEDConv can improve the adversarial robustness compared to the current static convolution model.

### 5.2.2 Parameter-efficient dynamic convolution

A straight-forward strategy to achieve dynamic parameters is by reparameterizing the weights for different inputs: $\mathcal{A}(\mathbf{x}) \odot \mathbf{W}_{\text{base}}$, where each element in $\mathcal{A}(\mathbf{x}) \in \mathbb{R}^{C_{out} \times C_{in} \times k \times k}$[2] is conditioned on input $\mathbf{x}$ and $\odot$ represents the Hadamard product. This formulation can be viewed as leveraging "slow" weights $\mathbf{W}_{\text{base}}$ that are shared across inputs and capture general information, and "fast" weights $\mathcal{A}(\mathbf{x})$ that learn specific information for adaptation to each individual input. Moreover, the direction (in terms of sign) and magnitude of adaptation to each element in the base weights can vary per-input basis, However, generating $\mathcal{A}(\mathbf{x})$ can introduce considerable parameters and computation overheads, since $\mathcal{A}(\mathbf{x})$ has the same shape as the base weight. To circumvent this problem, we propose a tensor decomposition based reparameterization for parameter-efficient dynamic convolution (PEDConv), formulated as:

$$\mathbf{W}(\mathbf{x}) = \mathbf{W}_{base} \ \times_1 \ \texttt{diag}(\boldsymbol{\gamma}(\mathbf{x})) \ \times_2 \ \texttt{diag}(\boldsymbol{\phi}(\mathbf{x})) \ \times_3 \ \texttt{diag}(\boldsymbol{\psi}(\mathbf{x})), \tag{5.1}$$

where $\boldsymbol{\gamma}(\mathbf{x}) \in \mathbb{R}^{C_{out}}$, $\boldsymbol{\phi}(\mathbf{x}) \in \mathbb{R}^{C_{in}}$, $\boldsymbol{\psi}(\mathbf{x}) \in \mathbb{R}^{k^2}$, $\mathbf{W}_{base} \in \mathbb{R}^{C_{out} \times C_{in} \times k \times k}$. The operator $\times_i$ denotes the $i$-mode product. $\texttt{diag}(\cdot)$ converts a vector to a diagonal matrix. $\mathbf{W}(\mathbf{x})$ represents model weights that are conditioned on $\mathbf{x}$.

Eq.(5.1) performs the rank-1 decomposition of $\mathcal{A}(\mathbf{x})$ with three components $\boldsymbol{\gamma}(\mathbf{x})$, $\boldsymbol{\phi}(\mathbf{x})$, and $\boldsymbol{\psi}(\mathbf{x})$, capturing the input dynamics for the input/output channels and each kernel. The decomposition can also be expressed by: $\mathcal{A}(\mathbf{x}) \approx \boldsymbol{\gamma}(\mathbf{x}) \otimes \boldsymbol{\phi}(\mathbf{x}) \otimes \boldsymbol{\psi}(\mathbf{x})$, where $\otimes$ stands for the outer product of vectors.

[2]We assume a 2D convolution layer with output channels $C_{out}$, input channels $C_{in}$, and kernel size $k \times k$.

The $k \times k$ kernels are not decomposed since the kernel size is usually small (e.g., $3 \times 3$). Therefore, our reparameterization formula can be equivalently computed by the Hadamard product between the input-dependent decomposition and the base weights:

$$\mathbf{W}(\mathbf{x}) = \mathbf{W}_{base} \odot (\boldsymbol{\gamma}(\mathbf{x}) \otimes \boldsymbol{\phi}(\mathbf{x}) \otimes \boldsymbol{\psi}(\mathbf{x})). \tag{5.2}$$

Due to the tensor decomposition, we can grealy reducee the parameter overhead for generating $\mathcal{A}(\mathbf{x})$. We only need $C_{out} + C_{in} + k^2$ parameters to reparameterize each element of the base weights with size $C_{out} \times C_{in} \times k^2$. Applying PEDConv in a layer can be represented by:

$$Y_{t,w',h'} = \sum_{s=1}^{C_{in}} \sum_{j=1}^{k} \sum_{i=1}^{k} \Big( \gamma(\mathbf{x})_t \phi(\mathbf{x})_s \psi(\mathbf{x})_{ji} [\mathbf{W}_{\text{base}}]_{t,s,j,i} \Big) X_{s,w_j,h_i}, \tag{5.3}$$

where $w_j = (w' - 1)\Delta + j - p$, $h_i = (h' - 1)\Delta + i - p$, $\Delta$ is stride and $p$ is zero-padding. Following the standard design in CNN, we use batch normalization and an activation function (e.g., ReLU) after PEDConv when substituting it for static convolutions.

**Learning decomposition components.** For simplicity, we combine the decomposition components as a vector $\mathbf{g}(\mathbf{x}) = [\boldsymbol{\gamma}(\mathbf{x}); \boldsymbol{\phi}(\mathbf{x}); \boldsymbol{\psi}(\mathbf{x})] \in \mathbb{R}^{C_{out}+C_{in}+k^2}$. To generate $\mathbf{g}(\mathbf{x})$ for different $\mathbf{x}$, we propose to learn the conditional prior distribution, denoted by $p(\mathbf{g}|\mathbf{x}; \boldsymbol{\theta}, \boldsymbol{\pi})$, where $\theta$ represents the base convolution weights and $\pi$ refers to the weights of the generator producing $\mathbf{g}$. When training the model with the conditional distribution, the marginal log-likelihood is obtained as: $\log p(\mathbf{Y}|\mathbf{X}; \boldsymbol{\theta}, \boldsymbol{\pi}) = \sum_{i=1}^{N} \log \mathbb{E}_{\mathbf{g}_i \sim p(\mathbf{g}_i|\mathbf{x}_i;\boldsymbol{\theta},\boldsymbol{\pi})}[p(\mathbf{y}_i|\mathbf{x}_i, \mathbf{g}_i; \boldsymbol{\theta})]$. We consider a lower bound derived by the variational inference method, which is the expected loss over the conditional distribution:

$$\log p(\mathbf{Y}|\mathbf{X}; \boldsymbol{\theta}, \boldsymbol{\pi}) \geq \sum_{i=1}^{N} \mathbb{E}_{\mathbf{g}_i \sim p(\mathbf{g}_i|\mathbf{x}_i;\boldsymbol{\theta},\boldsymbol{\pi})}[\log p(\mathbf{y}_i|\mathbf{x}_i, \mathbf{g}_i; \boldsymbol{\theta})]. \tag{5.4}$$

To evaluate the gradient w.r.t. $\theta$ and $\pi$, we use the reparametrization trick [129]. We assume the conditional distribution $p(\mathbf{g}|\mathbf{x}; \boldsymbol{\theta}, \boldsymbol{\pi})$ has the form of Gaussian with mean $\mu$ and diagonal covariance $\texttt{diag}(\boldsymbol{\sigma}^2)$. At each layer, the generator module (denoted by $G_\pi$) takes as input the preceding layer's output feature-maps (denoted by $X$), and generate the distribution parameters (mean $\boldsymbol{\mu}$ and standard deviation $\boldsymbol{\sigma}$) of the conditional distribution. The actual decomposition components $\mathbf{g}$ are sampled from this conditional distribution using the reparameterization trick:

$$[\boldsymbol{\mu}_i, \boldsymbol{\sigma}_i] = G_\pi(\mathbf{X}_i), \quad \mathbf{g}_i \sim p(\mathbf{g}_i|\mathbf{x}_i; \boldsymbol{\theta}, \boldsymbol{\pi}) = \mathcal{N}(\boldsymbol{\mu}_i, \texttt{diag}(\boldsymbol{\sigma}_i^2)) = \boldsymbol{\mu}_i + \boldsymbol{\sigma}_i \odot \boldsymbol{\epsilon}, \tag{5.5}$$

where we use subscript $i$ to emphasize per-input basis. $\boldsymbol{\epsilon} \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$ is for reparameterization trick so that our training objective (i.e., Monte Carlo estimate [129] of the expectation in Eq.(5.4)) is differentiable w.r.t. $\theta$ and $\pi$ with the presence of sampling operation. Following [219], we perform a deterministic inference without sampling $\mathbf{g} = \mathbb{E}[\mathbf{g}|\mathbf{x}]$ at testing time.

**Generator design.** The decomposition components $\mathbf{g}(\mathbf{x})$ are generated based on the input feature-maps $\mathbf{X}$ in each layer. To make the generator compact and efficient, we employ a bottleneck two-layer MLP. Firstly, global average pooling (GAP) is applied to obtain the global spatial information of the input feature-maps. Then, two fully connected layers with a ReLU activation function in-between are applied to generate the distribution parameters for the conditional distribution in Eq.(5.5):

$$[\boldsymbol{\mu}, \boldsymbol{\sigma}] = G_\pi(\mathbf{X}) = \mathbf{W}_{\text{fc1}} \cdot (\mathbf{W}_{\text{fc2}} \cdot \frac{1}{HW} \sum_{i \in H, j \in W} \mathbf{X}_{c,i,j})_+. \tag{5.6}$$

We denote the two-layer MLP generator by $G_\pi(\cdot)$ with weights $\pi$, that takes as input the GAP feature of corresponding layer, and output mean $\mu$ and std $\sigma$ of $p(\mathbf{g}|\mathbf{x}; \boldsymbol{\theta}, \boldsymbol{\pi})$. $\pi$ represents the learnable parameters $\mathbf{W}_{\text{fc1}} \in \mathbb{R}^{(C_{out}+C_{in}+k^2)\times C_{in}/r}, \mathbf{W}_{\text{fc2}} \in \mathbb{R}^{C_{in}/r \times C_{in}}$, where $r$ here is the reduction factor in the bottleneck MLP. $\mathbf{X} \in \mathbb{R}^{C_{in} \times H \times W}$ denotes the input feature-maps, and $(\cdot)_+$ is ReLU activation. After computing the vector $\mathbf{g}(\mathbf{x})$ by Eq.(5.5),(5.11), we can obtain the decomposition components $\boldsymbol{\gamma}(\mathbf{x}), \boldsymbol{\phi}(\mathbf{x}), \boldsymbol{\psi}(\mathbf{x})$ in Eq.(5.1) by chunking $\mathbf{g}(\mathbf{x})$.

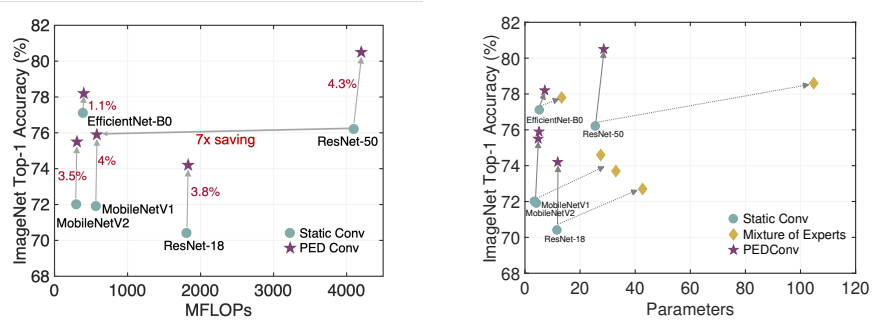
**Input-dependency by mutual information maximzation.** To enforce the input-dependency, we also propose to maximize the mutual information (MI) between the generated $\mathbf{g}$ and the input data during training. This MI objective is given by: $\max I((X, Y); A)$, where $A$ represents the dynamic weights, $X, Y$ are the input data and output label, respectively. The information maximization only occurs at training time for learning the model parameters, which is why we can use the label information in the MI objective. Directly computing MI is intractable since the true posterior distributions are unknown. Therefore, we use Variational Information Maximization [1] to derive a lower bound, which is given by:

$$\begin{aligned} I((X, Y); A) &= I(X; A) + I(Y; A|X), \\ &\geq \underbrace{\mathbb{E}_{X \sim p(X)}[\log p_{\theta'}(X|A = g_\theta(X))]}_{\text{local auto-encoder loss}} + \underbrace{\mathbb{E}_{Y \sim p(Y|X)}[\log p_w(Y|X, A = g_\theta(X))]}_{\text{global cross-entropy loss}}. \end{aligned} \tag{5.7}$$

We omit the constant entropy terms and the subscript of the expectation for clarity. The second term in Eq.(5.7) maximizes the log likelihood of label for the training data, which is equivalent to minimizing the cross-entropy loss in classification. This task loss optimizes the base convolution weights and the generator module weights. For the first term, $p_{\theta'}(X|A)$ is the approximated posterior, which is equivalent to the decoding term in the auto-encoder. We assume this decoding term takes the form of Gaussian. A known fact is that maximizing the log likelihood of Gaussian is equivalent to minimizing the MSE reconstruction loss. Specifically, we instantiate the decoding term $p_{\theta'}(X|A)$ by a decoder MLP with weights $\theta'$. This decoder MLP has two fully-connected layers with ReLU in-between, and it only appears at training time for MI maximization. The input of the decoder MLP is $A$, which is mapped to the reconstructed GAP input by the decoder MLP. During training, in addition to the cross-entropy loss, we also minimize the MSE based auto-encoder loss as an regularization term to encourage input-dependency. This MSE loss optimizes the generator module weights and decoder MLP weights. With this training objective, the generated decomposition components for PEDConv will be able to carry the information about each specific input data while optimizing the task loss.

**Complexity analysis.** Given that the spatial dimension is reduced for generating the input-dependent decomposition components, the computation cost of PEDConv mainly comes from the convolution operation. Specifically, the FLOPs complexity of PEDConv can be calculated by $(HWC_{out}C_{in}k^2)+((2C_{in}+C_{out}+k^2)C_{in}/r+2C_{out}C_{in}k^2)$, where the first term counts the convolution operation while the second term counts the decomposition components generator and application. The second term has much less computation compared to the convolution operation, making PEDConv almost as efficient as the static convolution. In terms of parameter complexity, static convolution and vanilla dynamic convolution [37, 267] require $C_{out}C_{in}k^2$ and $KC_{out}C_{in}k^2$ $(K \geq 4)$ parameters, respectively. In contrast, PEDConv requires $C_{out}C_{in}k^2$ for the base convolution weights, and an additional $(2C_{in} + C_{out} + k^2)C_{in}/r$ parameters are required by the bottleneck MLP for generating the decomposition components. Since the second term is relatively negligible, the proposed PEDConv has much less parameter complexity than the aggregation based dynamic convolutions that linearly combines multiple weights with scalar type attentions. The computatation and complexity analysis are supported by our results in Figure 5.2.

**Generalized rank-$R$ decomposition in PEDConv.** Eq.(5.2) gives a special case of rank-1 decomposition of $\mathcal{A}(\mathbf{x})$. A generalized rank-$R$ Canonical Polyadic tensor decomposition can be




(a) PEDConv has almost same FLOPs as static baseline.

(b) PEDConv has much less parameter overhead compared to previous method.

Figure 5.2: PEDConv boost the accuracy by 1% $\sim$ 4% of different CNN backbones on ImageNet while maintain the FLOPS as static baseline and requires much fewer parameters than previous method.

| **Tasks** | **Models** | **Datasets** | **Eval metrics** |
|---|---|---|---|
| Image classification | ResNet-18/50/101, MobileNetV1/V2, EfficientNet | ImageNet (1.28M train data, 1K categories) | Top-1 accuracy |
| Object detection | SSD | COCO (118K train data, 80 categories) | Mean average precision |
| Semantic segmentation | UperNet, PSPNet | ADE20K (25K train data, 3K categories, 193K parts) | Mean intersection over union |
| Adversarial learning | ResNet-50 | ImageNet (1.28M train data, 1K categories) | Clean/Robust accuracy |

Table 5.1: Summary of the tasks, models, and datasets on which we have evaluated PEDConv.

expressed as the sum of $R$ rank-1 tensors that are formulated as the outer product of rank-1 components: $\mathcal{A}(\mathbf{x}) \approx \sum_{r=1}^{R} \boldsymbol{\gamma}_r(\mathbf{x}) \otimes \boldsymbol{\phi}_r(\mathbf{x}) \otimes \boldsymbol{\psi}_r(\mathbf{x})$. $R$ influences the parameter efficiency, where the smaller the $R$ is, the more reduced the parameters. PEDConv with the rank-$R$ decomposition is given as: $\mathbf{W}(\mathbf{x}) = \sum_{r=1}^{R} \mathbf{W}_{\text{base}} \times_1 \texttt{diag}(\boldsymbol{\gamma}_r(\mathbf{x})) \times_2 \texttt{diag}(\boldsymbol{\phi}_r(\mathbf{x})) \times_3 \texttt{diag}(\boldsymbol{\psi}_r(\mathbf{x}))$.

### 5.2.3 Experiments

To verify the efficacy and generality of PEDConv, we evaluate on a variety of deep learning tasks with diverse CNN architectures. We use rank-1 CP decomposition for PEDConv in Eq.(5.1). We set the reduction factor in the bottleneck MLP to $r = 16$, and the loss balance factor in Eq.(5.7) to $\lambda = 1e^{-3}$. To keep our method simple and generic, these hyper-parameters are kept constant for all experiments. Table 5.1 provides a summary on the tasks, models, and datasets on which we have evaluated PEDConv.

**Image classification.** We evaluate PEDConv on ImageNet [42] using ResNet, MobileNet, and EfficientNet architectures, where all the convolution layers except the first one are replaced by

| Model | Method | Params | FLOPs | Top-1 (%) |
|---|---|---|---|---|
| MobileNetV1 | Baseline | 4.2M | 569M | 71.9 |
| | DSNet [140] | - | 565M | 74.5 |
| | DR-Conv [27] | - | 610M | 75.5 |
| | CondConv [267] | 33.0M | 600M | 73.7 |
| | **PEDConv** | 5.1M | 579M | **75.9** |
| MobileNetV2 | Baseline | 3.5M | 300M | 72.0 |
| | DK [61] | 11.1M | 760M | 74.8 |
| | CA [96] | 4.0M | 310M | 74.3 |
| | DCD [148] | 5.5M | 326M | 75.2 |
| | DY-Conv [37] | 11.1M | 313M | 75.2 |
| | CondConv [267] | 27.5M | 329M | 74.6 |
| | **PEDConv** | 4.8M | 307M | **75.5** |
| EfficientNet-B0 | Baseline | 5.3M | 391M | 77.1 |
| | CA [96] | 5.4M | 400M | 76.9 |
| | CondConv [267] | 13.3M | 402M | 77.8 |
| | **PEDConv** | 7.2M | 400M | **78.2** |

(a) Light-weight CNN models.

| Model | Method | Params | FLOPs | Top-1 (%) |
|---|---|---|---|---|
| ResNet-18 | Baseline | 11.7M | 1.81G | 70.4 |
| | SE [116] | 11.8M | 1.81G | 70.6 |
| | CBAM [257] | 11.8M | 1.82G | 70.7 |
| | DCD [148] | 14.0M | 1.83G | 73.1 |
| | DY-Conv [37] | 42.7M | 1.85G | 72.7 |
| | **PEDConv** | 11.9M | 1.83G | **74.2** |
| ResNet-50 | Baseline | 25.6M | 4.1G | 76.2 |
| | SE [116] | 28.1M | 4.1G | 77.5 |
| | CBAM [257] | 28.1M | 4.1G | 77.3 |
| | DK [61] | 81.2M | 4.6G | 78.5 |
| | SK [146] | 27.5M | 4.5G | 79.2 |
| | DCD [148] | 30.7M | 4.2G | 77.9 |
| | WeightNet [174] | 31.1M | 4.2G | 77.5 |
| | CondConv [267] | 104.8M | 4.2G | 78.6 |
| | **PEDConv** | 28.6M | 4.2G | **80.5** |

(b) Non-compact CNN models.

Table 5.2: Results of comparing PEDConv with state-of-the-arts on ImageNet. PEDConv outperforms previous dynamic convolutions [37, 267] in terms of higher accuracy and fewer parameters/FLOPs.

PEDConv. All models are trained by the SGD optimizer with a momentum of 0.9 and a batch-size of 512. When training ResNets, we set the weight decay to 1e-4. The initial learning rate is set to 0.512, and decays by a cosine scheduler for total 120 epochs. When training MobileNets, we set the weight decay to 4e-5. The initial learning rate is set to 0.1 and decays by cosine scheduler for total 250 epochs. When training EfficientNet, we follow the same procedure as [228]. We evaluate single-crop top-1 accuracy, and use input resolution $224 \times 224$ in both training and testing phases. Standard data augmentations described in PyTorch official repository are adopted.

The results are reported in Table 5.2, including comparisons with state-of-the-art methods. As shown, with comparable FLOPs and parameters, PEDConv consistently boosts the accuracy over baseline models. For example, PEDConv-MobileNetV2/ResNet-50 acheives 3.5%/4.3% accuracy gain over MobileNetV2/ResNet-50 with only 2% extra FLOPs, respectively. Moreover, PEDConv requires noticeably fewer model parameters compared with the state-of-the-art methods. On MobileNetV2, PEDConv only requires 43% of the parameters compared to DY-Conv and 17% of the parameters compared to CondConv, while achieving better top-1 accuracy. On ResNet-18/50, PEDConv only requires 28% of the parameters compared to DY-Conv/CondConv, while improving the accuracy by 1.5%/1.9%, respectively. In addition, on more powerful NAS-based mobile network EfficientNet-B0, PEDConv still achieves 1.1% accuracy gain over the baseline model. These results demonstrate the effectiveness, efficiency and compactness of the proposed PEDConv dynamic parameter design.

**Object detection.** We further apply PEDConv to SSD [163] on COCO dataset [158] for object detection tasks. Following [163], we train SSD on the COCO train2017 split containing about 118k

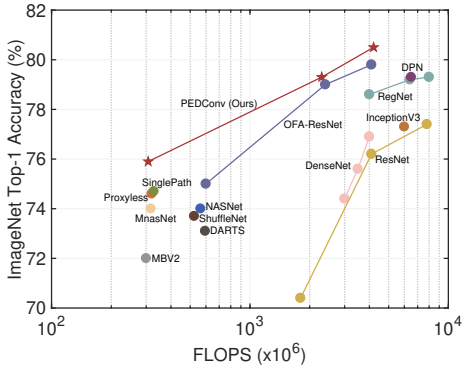



Figure 5.3: Compare PEDConv with NAS and advanced CNN architectures on ImageNet. PEDConv on simple backbones such as MobileNet and ResNet can already compete with the other advanced methods.

| Detector | Backbone | Params | FLOPs | $AP^{bbox}$ | $AP^{bbox}_{50}$ | $AP^{bbox}_{75}$ | $AP^{bbox}_{S}$ | $AP^{bbox}_{M}$ | $AP^{bbox}_{L}$ |
|---|---|---|---|---|---|---|---|---|---|
| SSD300 | ResNet-50 | 22.9M | 20.2G | 25.2 | 42.7 | 25.8 | 7.3 | 27.1 | 40.8 |
| | PEDConv-ResNet-50 | 24.4M | 20.3G | **29.3** | 48.2 | 30.4 | 9.4 | 31.9 | 47.9 |

Table 5.3: Results of PEDConv applied to SSD300 on COCO object detection.

images and test on the val2017 split containing 5k images. The input size is fixed to $300 \times 300$. We adopt SGD optimizer with a momentum of 0.9, a batch-size of 64, and a weight decay of 5e-4. The model is trained with a learning rate of 1e-3 for 160k iterations, which decays by 10 and 100 times, and continue to train for 40k iterations each.

As shown in Table 5.3, PEDConv achieves considerable performance gain across all evaluation metrics, e.g., 4.1% higher in mAP with only 6% parameters overhead and nearly the same computational cost. when examining the gains on different AP scores for small/medium/large objects, we find that the largest improvement appears on $AP_L$, where PEDConv surpasses the baseline by 7%. We conjecture that this improvement may arise from the power of PEDConv in adapting the convolution filters to diverse visual patterns and prioritizing the most informative elements w.r.t. different inputs.

**Semantic segmentation.** We also conduct experiments on the semantic segmentation. We employ the segmentation frameworks UperNet [260] and PSPNet [295] to validate PEDConv on the ADE20K dataset [298], which contains around 25K images for training and 2K images for testing. We follow the same training procedure as [260, 295] to train UperNet and PSPNet. The performance of the models are evaluated by the mean of the Intersection over Union (IoU) averaged over all semantic categories in the testing set. As shown in Table 5.4, PEDConv consistently improves mean IoU over the baseline models on both UperNet and PSPNet, achieving 2.4% and 1.3% gains, respectively. Consistent with our findings before, PEDConv incurs negligible computation overhead, maintaining

| Segmentor | Backbone | Params | FLOPs | Mean IoU (%) |
|---|---|---|---|---|
| UperNet | ResNet-50 | 64.1M | 156.2G | 40.4 |
| | PEDConv-ResNet-50 | 72.3M | 156.3G | **42.8** |
| PSPNet | Dilated ResNet-50 | 51.5M | 186.8G | 41.3 |
| | Dilated PEDConv-ResNet-50 | 59.7M | 186.9G | **42.6** |

Table 5.4: Results of PEDConv applied to UperNet and PSPNET on ADE20K semantic segmentation.

| Training | Model | Method | Params | FLOPs | Evaluated Against | | |
|---|---|---|---|---|---|---|---|
| | | | | | Natural Images (%) | PGD-10 (%) | PGD-50 (%) |
| Adversarial (Free $m = 4$ [213]) | ResNet-50 | Baseline | 25.6M | 4.1G | 60.2 | 32.8 | 31.9 |
| | | PEDConv | 28.6M | 4.2G | **64.2** | **35.6** | **34.8** |

Table 5.5: Validation accuracy and robustness of ResNet-50 with standard convolution and the proposed PEDConv. Both models are trained with adversarial training to resist $\ell_\infty$ $\epsilon = 4$ attacks.

the inference efficiency compared to convolutions.

**Adversarial robustness.** We investigate the robustness of CNNs with PEDConv to adversarially perturbed images (i.e., adversarial examples). We follow the adversarial training method [213] to train robust models on ImageNet datasett, given that adversarial training is one of the few defenses against adversarial attacks that withstands strong attacks. Table 5.5 summarizes the results of defending PGD-10/50 attacks with $\ell_\infty \epsilon = 4$. PEDConv significantly improves both the natural and robust accuracy when facing the PGD attacks , compared with the baseline ResNet-50 model.

To explain the improvement of robustness, we conjecture that PEDConv enhances the model capacity since it can adapt the model weights to different input images. Our results in Table 5.2(b) show that PEDConv-ResNet-50 achieves superior accuracy compared to much larger capacity ResNets. It has been studied in [177] that model capacity is a crucial factor to enhance the robustness and the ability to train against strong adversaries in adversarial training, since separating adversarial examples would require more complicated decision boundary. Our results suggest that dynamic parameter may be another efficient strategy to enhance model capacity and thus the adversarial robustness.

### 5.2.4 Analyses

**Different formulations of PEDConv.** Table 5.6(a) compares of the default formulation of PEDConv in Eq.(5.1) with three variants on ImageNet dataset using ResNet-18/50 architectures. $\boldsymbol{\gamma}(\mathbf{x})$ denotes dynamic parameters for the output channels only $\mathbf{W}(\mathbf{x}) = \mathbf{W}_{\text{base}} \times_1 \boldsymbol{\gamma}(\mathbf{x})$. $\boldsymbol{\phi}(\mathbf{x})$ denotes dynamic parameters for the input channels only $\mathbf{W}(\mathbf{x}) = \mathbf{W}_{\text{base}} \times_2 \boldsymbol{\phi}(\mathbf{x})$. $\boldsymbol{\phi}(\mathbf{x})$ denotes dynamic parameters for the kernels only $\mathbf{W}(\mathbf{x}) = \mathbf{W}_{\text{base}} \times_3 \boldsymbol{\psi}(\mathbf{x})$. All three variants enhance the

| $\gamma(\mathbf{x})$ | $\phi(\mathbf{x})$ | $\psi(\mathbf{x})$ | Variational | ResNet-18 | | | ResNet-50 | | |
|---|---|---|---|---|---|---|---|---|---|
| | | | | Params. | FLOPs | Top-1 (%) | Params. | FLOPs | Top-1 (%) |
| | | | | 11.7M | 1.8G | 70.4 | 25.6M | 4.1G | 77.0 |
| ✓ | | | | 11.8M | 1.8G | 72.7 | 27.9M | 4.1G | 79.1 |
| | ✓ | | | 11.8M | 1.8G | 73.0 | 27.4M | 4.1G | 78.7 |
| | | ✓ | | 11.7M | 1.8G | 73.1 | 26.7M | 4.1G | 78.3 |
| ✓ | ✓ | ✓ | | 11.9M | 1.8G | 73.7 | 28.6M | 4.2G | 79.5 |
| ✓ | ✓ | ✓ | ✓ | 11.9M | 1.8G | 74.2 | 28.6M | 4.2G | 80.5 |

(a)

| Rank | Params. | FLOPs | Top-1 (%) |
|---|---|---|---|
| $R = 1$ | 11.9M | 1.8G | 74.2 |
| $R = 4$ | 12.4M | 1.9G | 74.7 |
| $R = 8$ | 13.0M | 2.0G | 74.9 |

(b)

| PW | DW | Params. | FLOPs | Top-1 (%) |
|---|---|---|---|---|
| ✓ | | 4.8M | 579M | 75.3 |
| | ✓ | 4.6M | 569M | 75.0 |
| ✓ | ✓ | 5.1M | 579M | 75.9 |

(c)

| Layers | Params. | FLOPs | Top-1 (%) |
|---|---|---|---|
| Shallow | 4.3M | 570M | 74.0 |
| Deep | 5.1M | 578M | 75.0 |
| All | 5.1M | 579M | 75.9 |

(d)

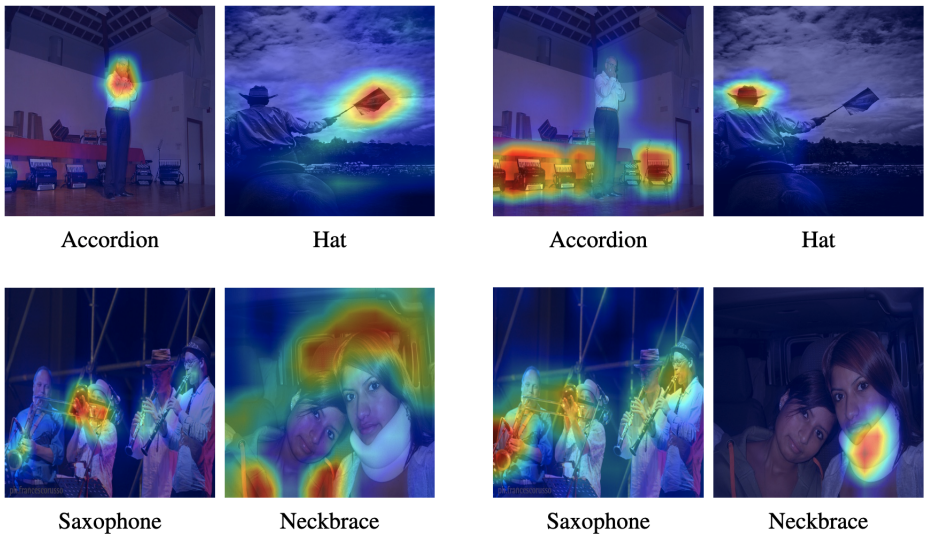
Table 5.6: (a) Study of different formulations of PEDConv applied to ResNet-18/50 on ImageNet. (b) Impact of decomposition rank for PEDConv applied to ResNet-18 on ImageNet. (c) PEDConv applied at different types of convolutions in MobileNetV1 on ImageNet. (d) PEDConv applied at different layer depth of MobileNetV1 on ImageNet.

accuracy compared to the static convolution, with kernel-wise dynamic parameter strategy shows the best parameter efficiency, due to the small kernel sizes. The combination of three components brings in additional accuracy improvement. These results necessitate our proposed omni-dimensional dynamic parameters. Moreover, we also study the effect of learning the conditional distribution with Eq.(5.5) versus deterministic generation of the decomposition components. We find that incorporating stochasticity improve the accuracy noticeably, i.e., 0.5%/1% gain on ResNet-18/50, respectively.

**Decomposition rank.** We investigate the impact of the rank values in the CP decomposition. As shown in Table 5.6(b), increasing the rank value improves the accuracy, at the cost of slightly more model parameters and computation cost.

**PEDConv at different layers.** Table 5.6(c) compares the results of replacing different types of convolution layers (PW: pointwise $1 \times 1$ convolution, DW: depthwise convolution) by PEDConv. We use MobileNetV1 on ImageNet dataset as an example. Applying PEDConv to any type of layer improves the accuracy, while using it for all types of layers yields the best accuracy. We find that substituting PEDConv for pointwise convolution layers yields better accuracy than for depthwise convolution layers, while the latter case is more parameter efficient.

**PEDConv at different depths.** We analyze the impact of applying PEDConv at different depths (shallow vs. deep layers) on ImageNet using MobileNetV1. As shown in Table 5.6(d), applying PEDConv in deeper layers is more beneficial than applying in shallower layers. We conjecture that deeper layers encode more context information, providing more clues to adapt the model weights to




(a) Static ResNet-50 (b) PEDConv ResNet-50

Figure 5.4: Compare the class-activation maps from the static resnet-50 and the pedconv model on ImageNet samples.

different examples.

**Visualization.** To further evidence the effectiveness of PEDConv, we adopt Grad-CAM [212] as the tool to visualize the attention-map produced by ResNet50 with PEDConv for classification on ImageNet. Grad-CAM generates a heatmap showing which region of the input image influence most to the model's prediction. The results are shown in Figure 5.4. The attention-maps produced by ResNet50 with PEDConv can more precisely locate the target objects, while the static model tends to only focus on the central regions thus making wrong decisions when the object locates at the periphery.

## 5.3 Dynamic architectures

### 5.3.1 Efficient inference by input-adaptive model structure

Considering that different inputs may have different computational demands, it is natural to performance inference with dynamic architectures adapted to each input. Neural networks with dynamic architectures can be most customized to the sample distribution of the dataset by allocating appropriate computation to process each input. For example, it is intuitive to use less computations on "easy" examples while preserving the representation power on "hard" examples. These properties lead to better efficiency-accuracy trade-off compared to static models that are indifferent to inputs with

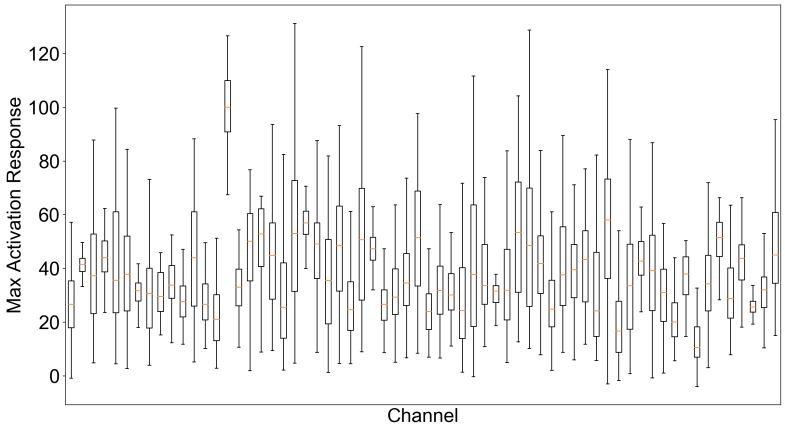
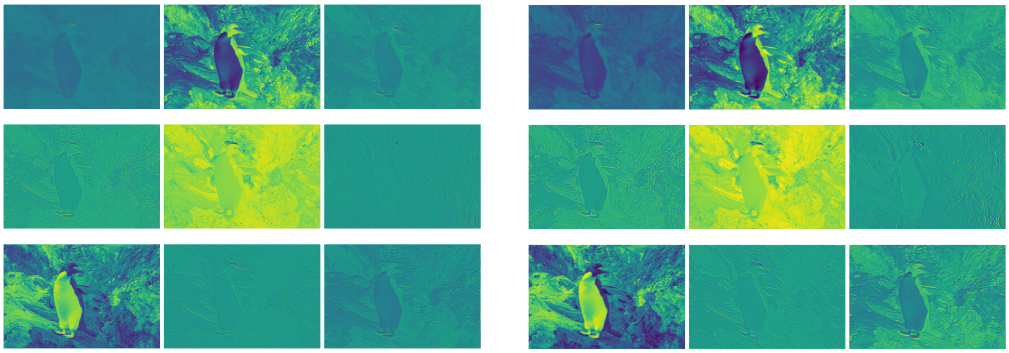



Figure 5.5: Variation of the maximum activation response for all channels in the first convolution layer of EDSR on 100 DIV2K validation images.



(a) $\text{Conv}(\mathbf{X}_{\text{in}})$ (b) $f_{\uparrow 2}(\text{Conv}(f_{\downarrow 2}(\mathbf{X}_{\text{in}})))$

Figure 5.6: Illustrations of spatial redundancy in SR networks. We take the first convolution layer in EDSR-baseline (x4 SR) as an example. (a): output feature-maps generated by the standard convolution. (b): output feature-maps generated by (i) down-scale the spatial resolution of input feature-maps, (ii) perform convolution, (iii) up-scale the output to the original spatial size.

identical computations.

In this section, we introduce a novel dynamic neural network applied on a fundamental computer vision task, single image super-resolution (SISR). SISR aims to recover a high-resolution (HR) image from a single low-resolution (LR) image. Since these exists multiple HR images that can be downsampled to the same LR input, the SISR problem is ill-posed. Recently, SISR systems based on deep convolutional neural networks [50, 153, 289, 294] have achieved substantial progress Nevertheless, these SISR models are computationally more expensive, and require excessively larger memory footprints, compared to image classification CNNs. For example, the FLOPs requires to process a $224 \times 224$ color image using the standard RDN [294] for $\times 4$ SISR is 1140G, which is $278\times$ of the and ResNet-50 model for classification. Moreover, the memory footprint of the SISR models can be two order of magnitude larger than the classification models.

Previous model compression algorithms like channel pruning [141, 166, 173, 91] can speedup the inference, reduce the storage and memory cost from intermediate feature-maps. However, the representation power of the model is permanently reduced after compression. Previous works neglect the diverse capacity demands from different images, thus the compressed model may not regain its accuracy after finetuning. In addition, previous works do not take into consideration that the model activation patterns exhibit high variation with respect to different inputs, which is a more significant phenomena on SISR task. In Figure 5.5, we plot the variation of the maximum activation response for all channels in the first convolution layer of EDSR model [153] on 100 DIV2K [3] validation images. We observe that some input images elicit very high activation values, whereas the other images elicit little response on the same set of channels. This suggests that the relative importance of each channel can vary greatly for different input images (i.e., high input-dependency). Last but not least, the spatial redundancy in the intermediate feature-maps is rarely considered by most of the model compression algorithms, but is an important problem in SISR to process high-resolution images. To illustrate the spatial redundancy in SISR models, we visualize the output feature-maps generated by the first-layer convolution in EDSR model on one DIV2K image in Figure 5.6(a). In Figure 5.6(b), we downsample the input feature-maps by $2\times$, then apply the optimized convolution layer, and finally upscale the output to original spatial size. Although the convolution is performed on spatially reduced feature-maps in Figure 5.6(b), the outputs have little visual discrepancy compared to Figure 5.6(a). This suggests that spatial redundancy can be reduced with minimal information loss in SISR models.

We propose a novel dynamic super-resolution (DSR) architecture that learns to jointly reduce the spatial-wise and channel-wise redundancy for different input images, as depicted in Figure 5.7. Spatial-wise, our DSR achieves input-adaptive resolution by performing convolution on downsampled feature-maps. A router module is inserted into each residual block and is trained to allocate different spatial resolutions of the input feature-maps for difference inputs. Channel-wise, our DSR achieves dynamic width by learning a channel saliency predictor for different inputs. Instead of permanently pruning unimportant channels, we accelerate the convolution by selectively computing a subset of channels identified by the saliency predictor, while skipping the computation of on unimportant channels.. The binary decisions on the channel computation are made trainable by employing the Gumbel-Softmax technique. Both spatial resolution routers and channel saliency predictors take the conditioning information computed from the feature-maps as input. We employ classic image processing metrics, e.g., Spatial Information (SI) [121], as the feature-maps statistics to guide the routers and saliency predictors. Compared with naive global average/maximum pooling, SI

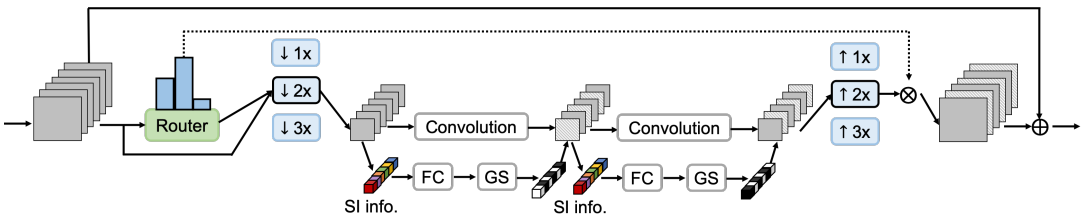



Figure 5.7: Residual block structure in the proposed dynamic architecture. We propose both dynamic spatial resolution and dynamic architectural width strategies. The router assigns input-adaptive decisions regarding the feature-maps resolution. The spatial information (SI), fully-connect layer (FC) and gumbel-softmax (GS) modules constitute a predictor for controlling the width.

can capture the high-frequency spatial details, which makes it more suitable on the SISR task. DSR model is trained end-to-end with gradient-based optimization using standard super-resolution loss, in conjunction with a sparsity criterion which drives the model to satisfy a computation constraint. Experiments on four SISR benchmarks demonstrate that DSR can save 2× FLOPs on both lightweight and non-compact SISR models while maintaining the quantitative/qualitative performance. In addition, we establish a new series of efficient models based on DSR with superior PSNR-FLOPs Pareto frontier compared to the SOTA NAS-based methods.

### 5.3.2 Dynamic spatial resolution.

The proposed DSR model is composed of residual blocks show in Figure 5.7. When an input image comes in, the router module in each block decides a spatial resolution. The input feature-maps are down-sampled to the chosen resolution and processed by the convolution layers in the block. The output feature-maps are up-sampled back to the original spatial size and added to the input feature-maps to form the residual learning. In each convolution layer, the Spatial Information (SI) of the input feature-maps are computed and fed into a MLP module to predict the channel importance scores. Gumbel-Softmax is used to selectively execute the important channels to produce the output feature-maps, while the computation on the unimportant channels are skipped.

Most of the SISR models are built upon residual blocks (RB): $\mathbf{X}_{b+1} = \mathcal{F}(\mathbf{X}_b) + \mathbf{X}_b$, where $\mathbf{X}_b$ is the input feature-maps to the $b$-th RB, and $\mathcal{F}(\cdot)$ is the residual function composed of two convolution layers with a non-linear function in-between. Our findings on the spatial redundancy in SISR models suggests that performing convolution on low-resolution representations together with parameter-free upsampling may be sufficient to reconstruct the final high-resolution image for some input images. Therefore, our DSR model adapts the feature resolution in $\mathcal{F}$ to different inputs to reduce the spatial redundancy and improve the inference efficiency.

A lightweight router network receives the feature-maps $\mathbf{X}_b$ of a single image and decides the

spatial resolution for the following convolution layers. The router network consists of a Spatial Information (SI) operator and a fully-connected (FC) layer with softmax activation. The SI operator [121] measures the amount of spatial details in the feature-maps by calculating the standard deviation over the pixels after Sobel-filtering: $\mathrm{SI}(\mathbf{X}_b) = \mathrm{Std}[\mathrm{Sobel}(\mathbf{X}_b)]$. The output of the SI operator is converted into the routing probability distribution:

$$\mathbf{h}_b = \mathbf{W}_b^{\mathrm{router}} \cdot \mathrm{SI}(\mathbf{X_b}), \tag{5.8}$$

$$p_i(\mathbf{X}_b) = \frac{\exp([\mathbf{h}_b]_i)}{\sum_{j=1}^{N} \exp([\mathbf{h}_b]_j)}, \tag{5.9}$$

where $\mathbf{W}_b^{\mathrm{router}}$ denotes the parameters of the FC layer, $\mathbf{h}_b$ is the pre-softmax logits, $p_i(\mathbf{X}_b)$ is the probability to choose spatial resolution $i$, and $N$ is the number of available resolutions.

After the spatial resolution $i \in [N]$ is chosen by the router, we down-sample the input feature-maps by a factor of $i$, e.g., $i = 2$ means the spatial size becomes $1/4$ of the original size. The down-sampled feature-maps are fed into the residual function $\mathcal{F}$. Our DSR model also perform adaptive channel selection within $\mathcal{F}$ in each convolution layer to achieve dynamic architectural width. The output feature-maps from $\mathcal{F}$ are finally up-sampled to the original spatial size and are added to the original input feature-maps to form the residual learning. We adopt average pooling for down-sampling and nearest-neighbour interpolation for up-sampling.

Based on the softmax distribution in Eq.(5.9), the spatial resolution corresponding to the maximum probability is selected. The entire block operated on the selected resolution. $i$ is described by:

$$\mathbf{X}_{b+1} = p_i(\mathbf{X}_b) f_{\uparrow i}(\mathcal{F}(f_{\downarrow i}(\mathbf{X}_b))) + \mathbf{X}_b, \tag{5.10}$$

where $i = \text{argmax} \ \{p_j(\mathbf{X}_b), j = 1, ..., N\}$, $f_{\uparrow i}, f_{\downarrow i}$ represent the up-sampling and down-sampling operation, respectively. We multiply the probability value to the residual output in order to make the router trainable end-to-end.

### 5.3.3 Dynamic architectural width.

In each convolution layer, our DSR model introduces a channel saliency predictor to perform conditional computation with dynamic width: the neurons or channels are selectively activated conditioned on the input. Inspired by squeeze-and-excitation [116], we design our saliency predictor as a two-layer MLP. We use the SI operator to extract the global information from the feature-maps. Then, two fully-connected (FC) layers with ReLU activation in-between and Sigmoid activation at

the end are applied to generate the channel importance scores, where the first FC has a reduction ratio of $r$ (we set $r = 16$). In the $l$-th convolution layer, the channel important scores $\mathbf{g}^l \in \mathbb{R}^{C^l}$ are calculated by:

$$\mathbf{g}^l = \delta(\mathbf{W}^l_{\text{fc1}} \cdot \sigma(\mathbf{W}^l_{\text{fc2}} \cdot \text{SI}(\mathbf{X}^{l-1}))), \tag{5.11}$$

where $\mathbf{X}^{l-1} \in \mathbb{R}^{C^{l-1} \times H^{l-1} \times W^{l-1}}$ denotes the input feature-maps, $\mathbf{W}^l_{\text{fc1}} \in \mathbb{R}^{C^{l-1} \times C^{l-1}/r}, \mathbf{W}^l_{\text{fc2}} \in \mathbb{R}^{C^{l-1}/r \times C^l}$ are the parameters of the FC layers, $\delta, \sigma$ represent Sigmoid and ReLU activation. ($C, H, W$ represent the channel number, spatial height and width, respectively.) The computation cost of this predictor module is cheap, requiring only $C^{l-1}H^{l-1}W^{l-1} + {C^{l-1}}^2/r + C^l C^{l-1}/r$ Multi-Adds. This is relatively negligible compared to the computation cost of convolution, requiring $C^{l-1}C^l H^l W^l k^2$ Multi-Adds for kernel size $k$.

To decide which channels are computed, we treat $\mathbf{g}^l_i$ as the probability to estimate how likely each channel is selected, since Sigmoid squeezes $\mathbf{g}^l_i$ to $[0, 1]$. (We use subscript $i$ to denote $i$-th element in the vector.) Directly sampling channels from a Bernoulli distribution using $\mathbf{g}^l_i$ as the probability is a reasonable option. However, this sampling procedure is not differentiable. Therefore, we employ the Gumbel-Softmax (GS) technique [122], which uses a differentiable sampling procedure to approximate the categorical distribution. GS converts the soft probabilities into hard decisions while enabling back-propagation to update the channel saliency predictor. GS defines a continuous and differentiable approximation as:

$$z_i = \frac{\exp((\log(\pi_i) + \xi_i)/\tau)}{\sum_{j=1}^{n} \exp((\log(\pi_j) + \xi_j)/\tau)}, \tag{5.12}$$

where $\xi_i$ are noise samples drawn from Gumbel(0,1) distribution, and $\tau$ is a temperature constant. The output of GS becomes a Bernoulli sample as $\tau$ approaches to 0. Since channel selection is binary ($n = 2$ in Eq.(5.12)), we further simplify the GS formulation. Let $\pi_1 = \mathbf{g}^l_i$ be the probability to choose $i$-th channel in $l$-th layer, and $\pi_2 = 1 - \mathbf{g}^l_i$ be the probability to skip the channel. Denote the pre-Sigmoid vector in Eq.(5.11) by $\hat{\mathbf{g}}^l$. Substituting $\pi_1, \pi_2$ into Eq.(5.12) reduces the GS formulation into:

$$z_1 = \sigma((\hat{\mathbf{g}}^l_i + \xi_1 - \xi_2)/\tau). \tag{5.13}$$

In our experiments, we set the GS temperature $\tau = 1$. We adopt the straight-through estimator [13] to generate the hard decisions during forward pass, and gradients are computed from soft samples

during backward pass:

$$\mathbf{m}_i^l = \begin{cases} \mathbb{1}_{z_1 > 0.5} = \mathbb{1}_{(\hat{\mathbf{g}}_i^l + \xi_1 - \xi_2)/\tau > 0} & \text{forward} \\ z_1 & \text{backward} \end{cases}, \tag{5.14}$$

where $\mathbf{m}_i^l$ is the decision for $i$-th channel. During inference, we do not add Gumbel noise (i.e. $\xi_1, \xi_2$=0) when generating the hard decisions. After $\mathbf{m}^l \in \mathbb{R}^{C^l}$ is obtained, the convolution is conducted on the selected channels only:

$$\mathbf{X}^l = f(\hat{\mathbf{W}}^l, \mathbf{X}^{l-1}), \quad \hat{\mathbf{W}}^l = \mathbf{W}^l[\boldsymbol{\nu}^l, :, :, :], \quad \boldsymbol{\nu}^l = \{i | \mathbf{m}_i^l = 1\}, \tag{5.15}$$

where $f(\cdot, \cdot)$ represents a convolution operation, $\mathbf{W}^l \in \mathbb{R}^{C^l \times C^{l-1} \times H^l \times W^l}$ denotes the convolution weights. Since we only activate the selected channels, the output feature-maps $\mathbf{X}^l$ has a dimension of $(1 - \eta^l)C^l \times H^l \times W^l$, where $\eta^l$ is the channel sparsity computed by $1 - |\boldsymbol{\nu}^l|/C^l$. This implies that the subsequent $(l+1)^{\text{th}}$ layer can also make use of the input-side sparsity $\eta^l$. Every convolution layer can exploit both input and output channel sparsity.

### 5.3.4 Sparsity loss and training strategy.

Without additional constraints, DSR model becomes accuracy driven, and the learned architecture will process each input image at the original resolution using all channels. Apart from the standard super-resolution loss, we add a sparsity loss to the training objective. We define a computation constraint $\kappa \in (0, 1)$ to represent the relative amount of desired computation. For example, $\kappa = 0.5$ indicates that *on average* 50% FLOPs of the full model would be activated. Suppose there are $B$ residual blocks in model, each having FLOPs $\mathbb{F}_b$ at full resolution and width. With dynamic resolution and width, the averaged FLOPs depends on the chose resolution $i_b$ in Eq.(5.10) and the channel sparsity $\eta_b^1, \eta_b^2$ (two convolutions per block) in Eq.(5.15). The FLOPs of $b$-th RB can be calculated by: $\mathbb{F}_{\text{b,sp}} = \frac{\eta_b^1 + \eta_b^1 \eta_b^2}{2(i_b)^2} \mathbb{F}_b$. To satisfy the computation constraint, we design a sparsity loss $\mathcal{L}_{\text{sp}}$ as follows:

$$\mathcal{L}_{\text{sp}} = \log(|\frac{\sum_{b=1}^{B} \mathbb{F}_{\text{b,sp}}}{\sum_{b=1}^{B} \mathbb{F}_{\text{b}}} - \kappa| + 1). \tag{5.16}$$

During training, this sparsity loss is averaged over images in each mini-batch, and the model learns to distribute the computational budget among different blocks and images.

To make the reconstructed high-resolution images $\mathbf{I}_{\text{SR}}$ have similar visual quality to the ground-

truth $\mathbf{I}_{\text{GT}}$, we use $\ell_1$ distance as the super-resolution loss:

$$\mathcal{L}_{\text{sr}} = \|\mathbf{I}_{\text{SR}} - \mathbf{I}_{\text{GT}}\|_1. \tag{5.17}$$

The final optimization objective involves two kinds of losses, $\mathcal{L}_{\text{sr}}$ for reconstructing high quality images and $\mathcal{L}_{\text{sr}}$ for constraining the computation cost of the SR network:

$$\mathcal{L}_{\text{total}} = \mathcal{L}_{\text{sr}} + \alpha\mathcal{L}_{\text{sp}}. \tag{5.18}$$

Optimizing Eq.(5.18) from scratch produces relatively lower PSNR at target computation constraint, since learning dynamic architectures and model weights jointly causes training instability, i.e., the decision functions are hard to converge into a stable situation. To remedy the training issue, we propose a two-stage training strategy:

1. **Pre-training stage**: Train a static SISR architecture from scratch without routers (Eq.(5.8)) and salience predictors (Eq.(5.11)). The training loss involves $\mathcal{L}_{\text{sr}}$ only. This supervised pre-training is able to leverage the labeled data to initialize the model weights, paving way for the searching stage.

2. **Searching stage**: Construct dynamic architecture by adding routers and salience predictors. Train the model weights with routers and predictors using joint loss Eq.(5.18), so that the DSR model can learn to adapt the spatial resolution and architectural width to different inputs while satisfying the FLOPs constraint.

### 5.3.5 Experiments

**Datasets and evaluation metrics.** Following [187], we use the 800 RGB images from the DIV2K dataset [3] to train our DSR models. In order to compare with previous methods, we test the model on four benchmarks: Set5 [15], Set14 [287], BSD100 [180], and Urban100 [118]. The low-resolution images are generated by downsampling the HR images by the bicubic kernel. We use standard evaluation metrics from previous works: PSNR and SSIM between the reconstructed images and the ground-truth on Y-channel of the YCbCr color space.

**Implementations.** We inherit the state-of-the-art SISR architectures, e.g., EDSR [153], CARN [4], and RDN [294], as the backbones of our DSR models, but replace the building blocks by our residual blocks with dynamic resolution and width. We set $\alpha = 1$ in Eq.(5.18). We use a batch-size

| Tasks | Backbones | Datasets | Eval metrics |
|---|---|---|---|
| Single image super resolution | EDSR, CARN, RDN | DIV2K (for training)<br>Set5, Set14, BSD100, Urban100 (for testing) | PSNR, SSIM |

Table 5.7: Summary of the tasks, backbone, and datasets on which we have evaluated the DSR model.

of 16 and an input patches of size $48 \times 48$ cropped from the low-resolution inputs. Standard data augmentations are employed, including random rotation by $90°$ and random horizontal flipping. In addition, all input images are substracted by the mean values computed over all DIV2K images. We adopt the ADAM optimizer with $\beta_1 = 0.9, \beta_2 = 0.999, \epsilon = 10^{-8}$. The initial learning rate is $10^{-4}$ and gets halved every 200 epochs for a total of 1000 training epochs. Table 5.7 provides a summary of the datasets and backbones that we use.

**Comparison with model compression methods.** In Table 5.8, we compare the PSNR/SSIM of our DSR models against other model compression algorithms for $\times 4$ SISR. Among various competitors, C-SGD [46] and DI [101] are the state-of-the-art static channel pruning methods; GhostDW [77] and GhostSR [187] replace the standard convolutions by cheaper depth-wise convolutions or shift operations to reduce FLOPs. When using the lightweight CARN backbone, our DSR model achieves $2\times$ FLOPs saving while slightly outperforming the baseline on three out of four benchmarks. We conjecture that DSR model maintains the representation ability by dynamically adjusting the resolution and architectural width for different inputs. Due to the two-stage training strategy, DSR model can leverage the pre-training information to learn more effective decision functions in the searching stage. On the other hand, the competitors achieve very limited ($\sim$20%) FLOPs savings with PSNR loss. For example, the PSNR of C-SGD drops by 0.1dB on the Urban100 dataset. This is because CARN is already compact model. Static channel pruning permanently reduces the model capacity and representation power. Compared with [77, 187], DSR models also achieve better PSNR. When using the RDN backbone, our DSR models can reduce nearly half of FLOPs while yielding higher PSNR/SSIM than depth-wise convolution based models [187]. Since DSR models can minimize the spatial-wise and channel-wise redundancy jointly, we achieve better PSNR-FLOPs trade-off than other methods that only focus on channel-wise reduction. Figure 5.11 compares the visual quality of the reconstructed high-resolution images from DSR models versus the baseline models. As observed, the details and textures of the images from DSR models are almost same as the baseline model, even though the DSR models have 50% less FLOPs. In Figure 5.8, we compare with state-of-the-art dynamic layer/channel pruning methods, e.g., SkipNet [249] and FBS [64]. We plot the PSNR against a range of model FLOPs and our DSR models consistently outperform the two competitors.

| Scale | Method | FLOPs reduction | Set5 PSNR/SSIM | Set14 PSNR/SSIM | BSD100 PSNR/SSIM | Urban100 PSNR/SSIM |
|---|---|---|---|---|---|---|
| x4 | EDSR-baseline [153] | 0 | 32.14 / 0.893 | 28.59 / 0.782 | 27.59 / 0.737 | **26.12** / **0.787** |
| | DI [101] | 50% | 32.04 / 0.891 | 28.50 / 0.779 | 27.52 / 0.735 | 25.90 / 0.780 |
| | **DSR (Ours)** | 50% | **32.25** / **0.894** | **28.63** / **0.782** | **27.59** / **0.737** | 26.04 / 0.784 |
| | CARN [4] | 0 | 32.16 / 0.894 | 28.59 / 0.781 | **27.58** / 0.736 | 26.03 / 0.783 |
| | C-SGD [46] | 22% | 32.08 / 0.893 | 28.52 / 0.780 | 27.56 / 0.735 | 25.93 / 0.780 |
| | Ghost-DW [77] | 19% | 32.14 / 0.894 | 28.59 / 0.781 | 27.57 / 0.735 | 26.02 / 0.782 |
| | GhostSR [187] | 20% | 32.11 / 0.893 | 28.57 / 0.780 | 27.56 / 0.735 | 25.98 / 0.781 |
| | **DSR (Ours)** | 50% | **32.20** / **0.894** | **28.61** / **0.781** | 27.57 / **0.736** | **26.05** / **0.783** |
| | RDN [294] | 0 | **32.47** / **0.899** | **28.81** / **0.787** | **27.72** / **0.742** | **26.61** / **0.803** |
| | GhostSR [187] | 47% | 32.32 / 0.897 | 28.70 / 0.784 | 27.66 / 0.740 | 26.37 / 0.795 |
| | **DSR (Ours)** | 47% | 32.39 / 0.897 | 28.78 / 0.785 | 27.68 / 0.740 | 26.41 / 0.795 |

Table 5.8: Quantitative results of our DSR models compared with baseline models and other model compression methods for $\times 4$ SISR. DSR models achieve better PSNR/SSIM while reducing more FLOPs compared with other approaches.

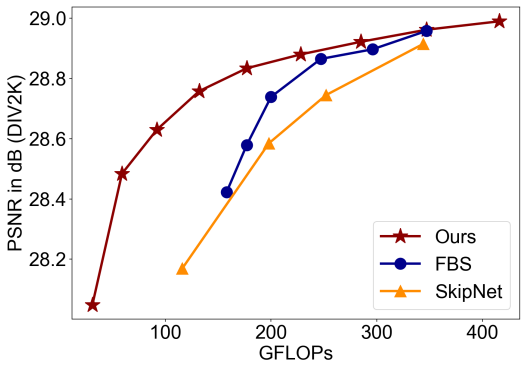



Figure 5.8: Comparison with SOTA dynamic layer/channel pruning methods, e.g., SkipNet [249], FBS [64], on the DIV2K validate set.

**Comparison with super-efficient SISR architectures.** In Table 5.9 and Figure 5.9, we compare with super-efficient SISR architectures. Since we aim to obtain highly compact models, we establish a series of DSR models with extremely large FLOPs saving by using the EDSR as the backbone. Our super-efficient models, DSR-L/-M/-S have 6.6, 3.6, and 1.9 GFLOPs, respectively. To the best of our knowledge, SESR [16] is the current best architecture in terms of PSNR-FLOPs trade-off. Notably, DSR-L/S achieves higher PSNR/SSIM than SESR-XL/M11 on all benchmarks under the same FLOPs. Compared with FEQE-P [240], DSR-M achieves competitive PSNR/SSIM while requiring 55% less FLOPs. Moreover, DSR-M outperforms the NAS-based TPSR-NoGAN [137] noticeably under the same FLOPs. For example, on Set5 dataset, DSR-M achieves 0.4dB gain over TPSR-NoGAN. Finally, we find that both DSR-L and DSR-M achieve better PSNR than VDSR [128], but have 93$\times$ and 170$\times$ FLOPs savings, respectively. Figure 5.12 shows the image quality of different super-efficient SISR architectures. Reconstructed images from our DSR models show higher quality with sharper edges and less artifacts than the other approaches.

| Scale | Method | FLOPs | Set5 PSNR / SSIM | Set14 PSNR / SSIM | BSD100 PSNR / SSIM | Urban100 PSNR / SSIM |
|---|---|---|---|---|---|---|
| x4 | VDSR [128] | 613G | 31.35 / 0.884 | 28.02 / 0.767 | 27.29 / 0.725 | 25.18 / 0.752 |
| | SESR-XL [16] | 6.6G | 31.54 / 0.887 | 28.12 / 0.771 | 27.31 / 0.728 | 25.31 / 0.760 |
| | **DSR-L (Ours)** | 6.6G | **31.74** / **0.887** | **28.31** / **0.773** | **27.36** / **0.729** | **25.46** / **0.763** |
| | FEQE-P [240] | 5.6G | **31.53** / 0.882 | **28.21** / **0.771** | **27.32** / **0.727** | **25.32** / **0.758** |
| | FSRCNN [51] | 4.6G | 30.71 / 0.866 | 27.59 / 0.754 | 26.98 / 0.713 | 24.62 / 0.726 |
| | TPSR-NoGAN [137] | 3.6G | 31.10 / 0.878 | 27.95 / 0.766 | 27.15 / 0.721 | 24.97 / 0.746 |
| | **DSR-M (Ours)** | 3.6G | **31.53** / **0.884** | 28.19 / 0.770 | 27.29 / 0.727 | 25.24 / 0.756 |
| | SESR-M11 [16] | 1.9G | 31.27 / **0.881** | 27.94 / 0.766 | **27.20** / **0.723** | 25.00 / **0.747** |
| | **DSR-S (Ours)** | 1.9G | **31.31** / 0.879 | **28.04** / **0.767** | 27.19 / **0.723** | **25.03** / **0.747** |

Table 5.9: Quantitative comparison against super-efficient SISR architectures, including manually designed [51, 240, 16] and NAS [137]. FLOPs are calculated as the number of multiply-adds needed to convert an image to 720p ($1280 \times 720$) resolution. DSR's FLOPs are averaged over all images in the benchmarks.

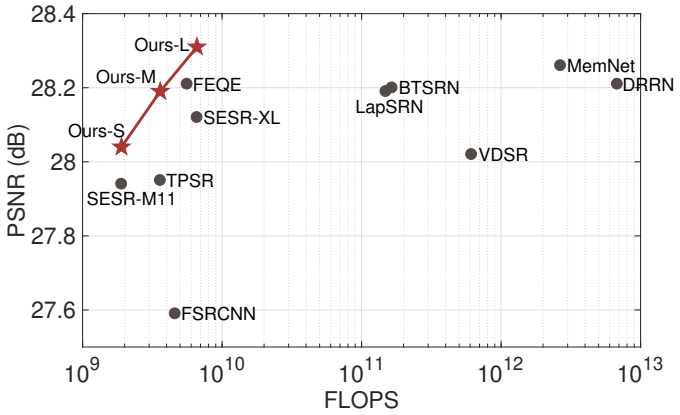



Figure 5.9: PSNR vs. FLOPs for various super-efficient SISR models. Results are obtained on Set14 dataset for $\times 4$ SISR tasks. DSR models achieve superior PSNR-FLOPs Pareto frontier compared to the state-of-the-arts.

### 5.3.6 Analyses

**Impact of joint dynamic resolution and width.** Our DSR model can adapt both the spatial resolution and architectural width for different inputs. To investigate the impact of this joint strategy, we compare with dynamic resolution alone or dynamic width alone Table 5.10. Under the same FLOPs, either dynamic resolution alone or dynamic width alone yields lower PSNR than the joint strategy. It is also interesting to note that dynamic resolution leads to higher PSNR than dynamic width, suggesting that the spatial-wise redundancy may be more phenomenon than the channel-wise redundancy in the feature-maps.

**Impact of feature-maps statistics.** In the DSR model, we compute the Spatial Information (SI) of the feature-maps to guide the routers and the channel saliency predictors. In Table 5.11, we compare SI with global average pooling and globabl maximum pooling for extracting feature-maps

| Dynamic width | Dynamic resolution | Set5 | Set14 | B100 | Urban100 |
|---|---|---|---|---|---|
| ✓ | ✗ | 32.16 | 28.56 | 27.55 | 25.98 |
| ✗ | ✓ | 32.20 | 28.61 | 27.57 | 26.00 |
| ✓ | ✓ | 32.25 | 28.63 | 27.59 | 26.04 |

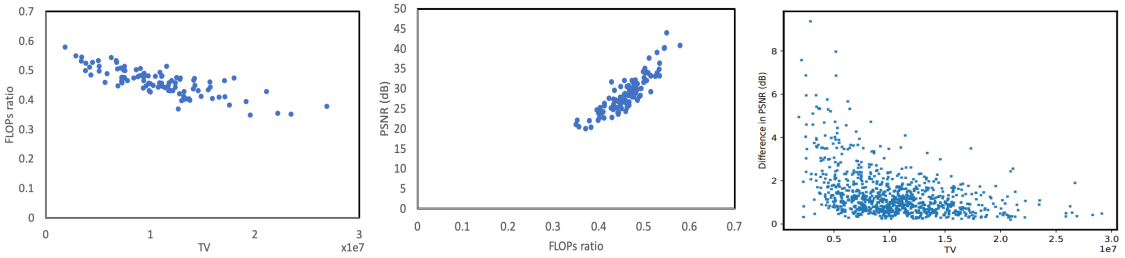
Table 5.10: Impact of the jointly adapting the spatial resolution and architectural width in the DSR model. We report PSNR results for different strategies on $\times 4$ SISR tasks.

| Statistics | Set5 | Set14 | B100 | Urban100 |
|---|---|---|---|---|
| Global max pooling | 32.05 | 28.50 | 27.51 | 25.83 |
| Global average pooling | 32.19 | 28.55 | 27.53 | 25.93 |
| Spatial Information | 32.25 | 28.63 | 27.59 | 26.04 |

Table 5.11: Comparison of different feature-maps statistics for guiding the routers modules and the channel saliency predictors in the DSR model.

statistics. SI achieves the highest PSNR on four benchmarks, due to its power to capture the high-frequency spatial details. Max pooling operation causes the greatest information loss by discarding most of the spatial pixel value, thus yields the lowest PSNR.

**Understanding the input-adaptive decisions.** We study the correlation between the decisions made by the DSR model and the complexity of input images. In Figure 5.10(a), we plot the FLOPs used to process each image in the DIV2K validation set versus the complexity of each image measured by total variation (TV) [208]. A larger value of TV implies a more complex image. In Figure 5.10(b), we plot the FLOPs versus the PSNR for each image in DIV2K validation set. Interestingly, DSR model (using EDSR backbone) assigns more FLOPs to process relatively simpler images and achieve better PSNR on these images, and assign less FLOPs to process more complex images. This leads to an average of 28.96 dB PSNR on the DIV2K validation set and $2\times$ FLOPs saving compared to the baseline EDSR model. To explain this seemingly counter-intuitive policy, we conduct the following experiment. We compare two models with substantial capacity difference, one is the EDSR model and the other one is obtained by shrinking the width of EDSR by $10\times$. Figure 5.10(c) shows the PSNR difference between these two models versus the TV values on all DIV2K images. As observed, complex images are almost equally hard for both models. On simpler images, the larger model achieves significantly higher PSNR. Our DSR model automatically learn this property, by using less FLOPs to process complex images since the PSNR would not get improved much even with more FLOPs, while using more FLOPs to process simpler images to achieve high PSNR.




(a) (b) (c)

Figure 5.10: (a) FLOPs for processing each image in the DIV2K validation set versus the total variation (TV) of these images. (b) PSNR on each image of the DIV2K validation set versus FLOPs for processing these images. (c) PSNR difference between two models with substantially different capacity versus the TV values on all DIV2K images.

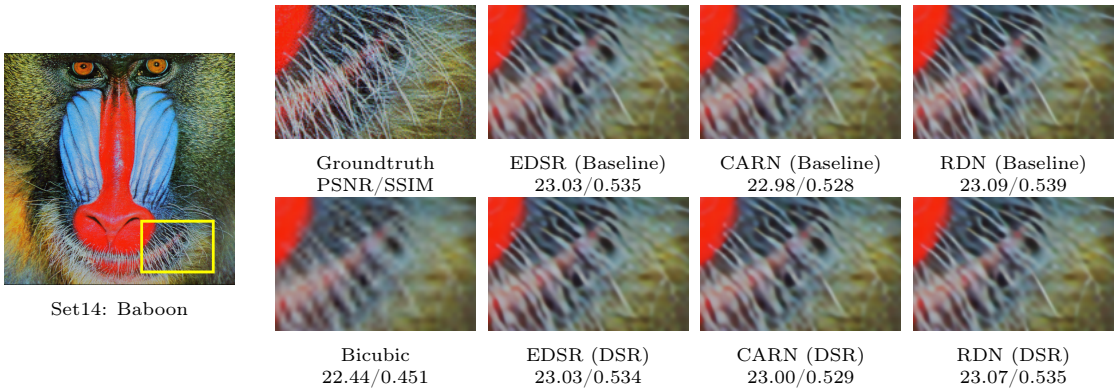



Figure 5.11: Visual comparisons of the baseline models and our DSR models on $\times 4$ SISR tasks.

## 5.4 Conclusion

In this chapter, we introduce two dynamic neural network approaches: (1) the parameter efficient dynamic convolution PEDConv use input-adaptive parameters to improve the accuracy with negligible computation overhead; (2) the dynamic architecture with input-adaptive spatial resolution and width can reduce the computation demand with little performance loss.

In PEDConv, we propose a tensor decomposition based weight reparameterization to achieve input-dependent parameters. To enforce the input dependency, we propose a mutual information maximization objective and learn the conditional prior distribution for generating the decomposition components. PEDConv can serve as an effective substitute of static convolution in existing manually designed and NAS models. Experiments on multiple deep learning tasks demonstrate the efficacy and generality of PEDConv, which significantly outperforms SOTA dynamic convolutions in terms of accuracy while requiring fewer parameters and FLOPs.

Our design of dynamic architecture are motivated by our findings that the channel activation exhibit high input-dependency and the spatial redundancy can be reduced by downsampling with

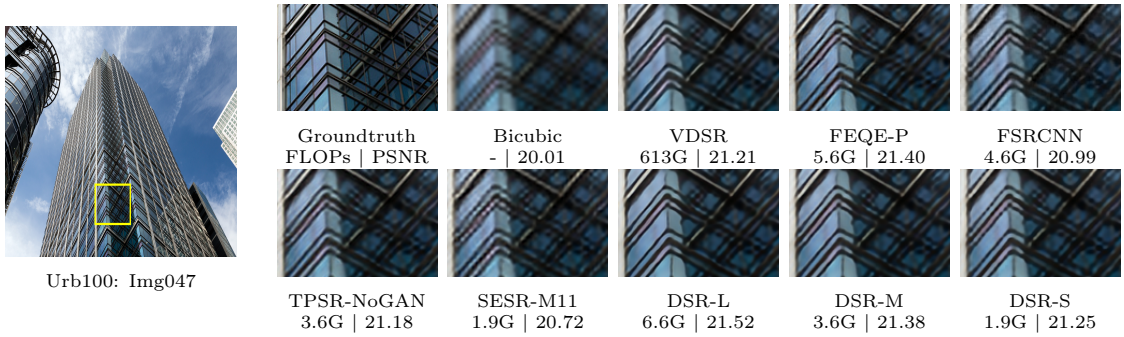



Figure 5.12: Visual comparison with SOTA super-efficient SISR architectures for $\times 4$ SISR. Our DSR models demonstrate better image quality while requiring similar or fewer FLOPs than other manually designed or NAS-based models.

minimal information loss, espectially on the low-level tasks such as super-resolution. We propose to jointly reduce spatial and channel wise redundancy and vary the computation cost for different inputs by dynamic spatial resolution and architectural width designs. Our method can be readily applied to a variety of backbones and trained end-to-end with standard training objective, in combination with a sparsity loss. Experiments on mutiple benchmarks show that our dynamic architecture achieves noticeable FLOPs savings with negligible quantitative and visual quality loss. Moreover, our models achieve superior PSNR-FLOPs trade-off compared with SOTA super-efficient super-resolution methods including NAS designed super-resolution architectures.

Overall, we have demonstrated superior accuracy-efficiency trade-off on a variety of tasks by our dynamic neural network designs.

# Chapter 6

# Data Efficient Deep Learning via Few-Shot Learning

## 6.1 Prologue

The remarkable success of current deep learning models mostly hinges on substantial supervised training examples. In contrast, humans are capable to learn new tasks rapidly given a few examples, by utilizing the previously learned prior knowledge. Few-shot learning (FSL) [58] is a machine learning paradigm that empowers deep learning models to rapidly generalize from a limited number of examples with supervised information.

Similar to human intelligence, FSL also utilizes prior knowledge to learn with few labeled examples on unseen tasks, and is closed related many other machine learning problems depending on the types of prior knowledge. When the model learns from a small number of labeled examples and utilize additional unlabeled examples from the task as the prior knowledge, FSL becomes a type of semi-supervised learning. When the model is pretrained on source tasks having abundant data and transfer to target tasks having limited training examples, FSL becomes a type of transfer learning. When the model has seen a lot of similar few-shot tasks during training and is able to generalize to new few-shot tasks, FSL becomes a type of meta-learning.

To improve the FSL performance, we perform an error decomposition analysis [252] which can provide some insights on our proposed methods in Section 6.2 and Section 6.3. Given a hypothesis $h$, we minimize the expected risk $\mathcal{R}$ with respect to the jointly distribution $p(x, y)$: $\mathcal{R}(h) = \int \ell(h(x), y)\, dp(x, y)$. Since the joint distribution is unknown, the empirical risk over the

training set of $N$ data is defined as: $\mathcal{R}_N(h) = \frac{1}{N}\sum_{i=1}^{N} \ell(h(x_i), y_i)$. We also define the following terms: $\hat{h} = \arg\min_h \mathcal{R}(h)$, $h^* = \arg\min_{h\in\mathcal{H}} \mathcal{R}(h)$, $h_N = \arg\min_{h\in\mathcal{H}} \mathcal{R}_N(h)$. The total error can be decomposed as [17]:

$$\mathbb{E}[\mathcal{R}(h_N) - \mathcal{R}(\hat{h})] = \underbrace{\mathbb{E}[\mathcal{R}(h^*) - \mathcal{R}(\hat{h})]}_{\text{approximation error}} + \underbrace{\mathbb{E}[\mathcal{R}(h_N) - \mathcal{R}(h^*)]}_{\text{estimation error}}, \tag{6.1}$$

where the expectation is with respect to random choice of the training set. The approximation error term measures the distance between the best function in the hypothesis space and the optimal hypothesis $\hat{h}$. The estimation error measures the distance between the empirical risk minimizer on a training set and the expected risk minimizer in the hypothesis space.

To reduce the total error, we can construct better hypothesis space to reduce the approximation error and/or to exploit extra data to learn more realizable empirical risk minimizer to suppress the estimation error. From the first perspective, we propose meta-learning a task-aware architecture controller to customize task-specific model space for different few-shot tasks. Moreover, we propose a parameter-efficient adaptation method to constrain the complexity of the hypothesis space and reduce the risk of overfitting. From the second perspective, we propose a semi-supervised few-shot learning framework to obtain more reliable empirical risk minimizer. An unsupervised dependency maximization loss is proposed to exploit unlabeled data in conjunction with supervised training on the few-shot labeled data. More details are introduced next.

## 6.2 Few-shot structural learning

### 6.2.1 Background

The goal of few-shot learning is to explicitly optimize a model for efficient adaptation to any low-resource task $\mathcal{T}_i$ (with very limited labeled training data) sampled from task distribution $p(\mathcal{T})$. Each task consists of a training set $\mathcal{D}_i^{\text{train}}$, a test set $\mathcal{D}_i^{\text{test}}$, and a loss function $\mathcal{L}_i$.

The prevailing approach for efficiently optimizing a base model $f_\Theta$ on a (relatively) small task-specific dataset is to use model-agnostic meta-learning (MAML [59]). The difference between meta-learning and traditional supervised learning is illustrated in Figure 6.1. Meta-learning involves a bi-level optimization process that uses a stochastic gradient-based strategy to sample a batch of tasks $\{\mathcal{T}_i\}_{i=1}^{B}$ from the task distribution $p(\mathcal{T})$ in each meta-iteration. In the inner loop, each task finetunes a copy of the model's weights $\Theta$ for a small number of steps $T_{\text{in}}$, producing task-specific weights $\Theta_i$. In the outer loop, each task model $f_{\Theta_i}$ is evaluated on its corresponding task's test set $\mathcal{D}_i^{\text{test}}$ and

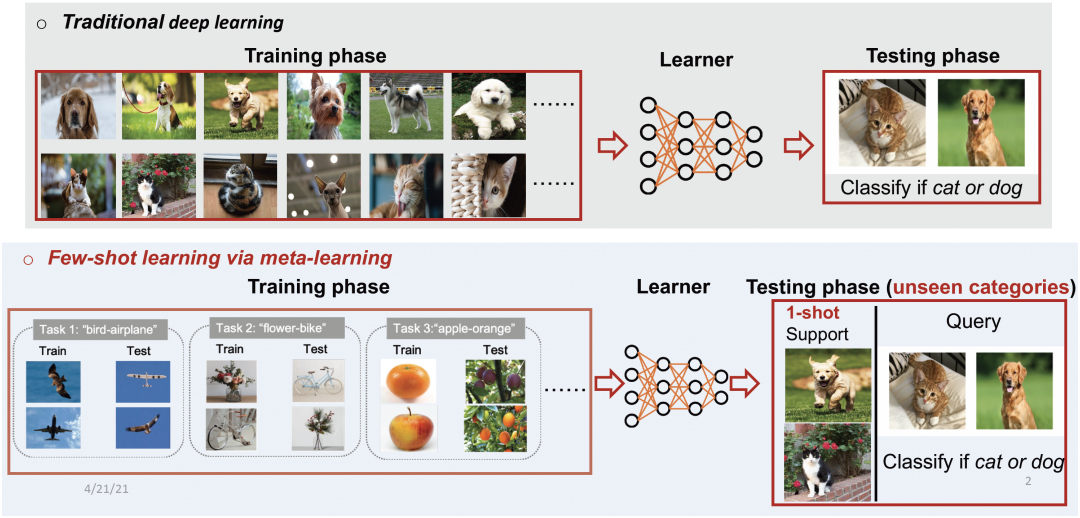



Figure 6.1: Compare meta-learning with traditional supervised learning.

these losses are summed to produce the overall meta-loss. The meta-loss $\sum_{\mathcal{T}_i \sim p(\mathcal{T})} \mathcal{L}_i^{\text{test}}(f_{\Theta_i})$ is then used to optimize and update $\Theta$. In summary, the MAML objective is given by:

$$\Theta^* = \underbrace{\underset{\Theta}{\operatorname{argmin}} \sum_{\mathcal{T}_i \sim p(\mathcal{T})} \mathcal{L}_i^{\text{test}}(\underbrace{f_{\Theta - \alpha \nabla_\Theta \mathcal{L}_i^{\text{train}}(f_\Theta)}}_{\text{inner loop optimization}})}_{\text{outer loop optimization}}. \tag{6.2}$$

MAML, however, is not generally a competitive alternative to pretrain-finetune methods for low-resource and few-shot settings, especially in natural language process (NLP) tasks To rectify MAML's limitations, we propose a meta-learning the difference (MLtD) framework with applications to NLP tasks, in order to optimize pretrained language models (e.g., GPT-2) for fast and data-efficient adaptations with the following contributions:

**MAML after pretraining.** Earlier works run MAML on random initialization, or at best with pretrained token embeddings [176], which was shown to underperform the pretrain-finetune approaches. With the increased prevalence of large-scale pretraining in NLP, we propose to initialize meta-learning from pretrained weights. Moreover, we show that pretraining, even when performed only on the meta-training data (i.e., no external text), improves performance and mitigates overfitting versus pretraining alone or MAML alone approaches, suggesting that pretraining produces a better initialization that promotes generalization in later meta-learning.

**Parameter-efficient transfer.** Adaptation data is typically limited in NLP tasks, making it easy for large language models to overfit. Previous works use very shallow CNNs [59], only adapt scale-and-shift parameters atop the original model [224], or apply various general regularization techniques such as weight decay, label smoothing, dropout, early stopping, and $\ell_1$ regularization [176, 221]. In contrast, we propose learning dynamic low-rank reparameterizations $g_{\Phi_i}$ of the base model such that $\Theta_i(x) = g_{\Phi_i}(\Theta_{\text{LM}}, x)$ for task $\mathcal{T}_i$. Here, $\Phi$ is a small set of new parameters that are adapted into task-specific $\Phi_i$ when finetuning. Notably, we modify MAML to incorporate these parameter-efficient modules, so that during task adaptation in both meta-training (the inner loop) and meta-testing (novel tasks) $\mathcal{T}_i$, we only adapt $\Phi \rightarrow \Phi_i$ instead of $\Theta \rightarrow \Theta_i$, speeding up both phases and improving overall performance.

**Architecture adaptation.** While the Transformer has proven to be a general-purpose architecture in NLP, recent work has shown that the optimal attention-then-FFN structure can vary across tasks (e.g., Sandwich Transformers [199]). However, previous NLP architecture search approaches are often task-agnostic (e.g., Primer [217]), where the sublayer search is implemented before pretraining. Meta-learning enables learning data-driven sublayers *after* pretraining, in a differentiable, task-adaptive manner. Instead of a separate search per task as in previous methods (e.g., DARTS [161]), we propose meta-learning a task-aware architecture controller to help it generalize to new tasks when searching neural architectures, by learning to directly generate task-specific sublayer structures from the dataset. By exploiting architectural knowledge learned over the task distribution, our task-adaptive model structure approach improves test-time performance.

In summary, we employ meta-learning to improve upon initializing from a pretrained $\Theta_{\text{LM}}$, allowing better downstream finetuning on tasks. By learning only the transformation weights $\Phi_i$ and the task-specific architecture $\alpha_i$ for new tasks $\mathcal{T}_i$, our method "learns to learn the difference" between a PLM and a task-specific LM in a training-efficient way. Figure 6.2 provides an overview of the proposed framework.

### 6.2.2 Efficient parameter adaptation.

Inspired by the scale-and-shift parameter learning in [224], we propose learning *affine* transformations to reparameterize the pretrained model weights towards a task. For a pretrained weight matrix $W_0^l \in \mathbb{R}^{C_{\text{in}} \times C_{\text{out}}}$ (can be any dense layer in the self-attention module or the FFN module in a

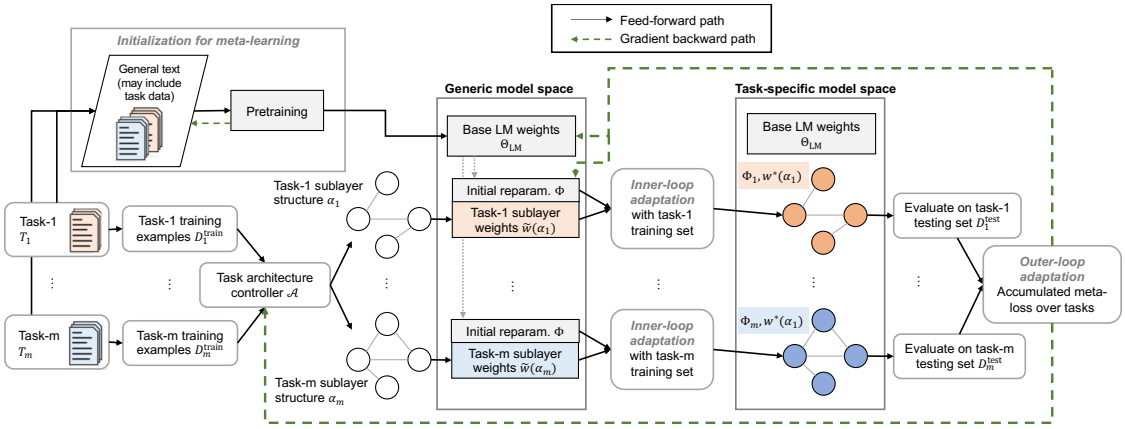



Figure 6.2: Overview of our proposed method, which learns to transform a small set of weights $\Phi_i$ (TARP learning) and modify sublayer modules $\alpha_i$ (TAMS learning) in a task-specific, data-efficient, and parameter-efficient manner. First, we initialize with a base PLM (**top left**). In each meta-iteration, we sample a batch of tasks from a task distribution (**left**). In the inner loop (**middle**), independent sets of dynamic low-rank reparameterizations are initialized, and an architecture controller generates independent task-specific sublayer modules, all of whose weights are adapted to the task's training set. Each task model is evaluated on the corresponding task's test set. In the outer loop (**right**), these task losses are summed up to produce the overall meta-loss, and the backward path optimizes the base model, the initial reparameterization, and the architecture controller.

transformer based architecture), we first reparameterize the task-specific weights as:

$$W^l = \Phi_1^l \odot W_0^l + \Phi_2^l \tag{6.3}$$

where $\Phi_1^l, \Phi_2^l \in \mathbb{R}^{C_{\text{in}} \times C_{\text{out}}}$ and $\odot$ denotes the elementwise (Hadamard) product.

At adaptation time, we apply low-rank constraints while optimizing the reparameterization weights only, giving the training objective:

$$\min_{\{\Phi_1^l, \Phi_2^l\}_{l=1}^L} \sum_{t=1}^{T} \log p(y_t | x, y_{<t}; \{W^l\}_{l=1}^L), \quad \text{s.t.} \quad \text{rank}(\Phi_i^l) < r, \ i \in \{1,2\}, \ l \in [L]. \tag{6.4}$$

The classic way to solve the rank-constrained problem is approximate the constraint by nuclear norm and adopt Proximal gradient method, where each step contains an expensive SVD computation and matrix soft-thresholding. To accelerate the training process, we apply a low-rank decomposition to the transformation weights $\Phi_i^l$. We term this approach of learning parameter-efficient affine transformations **task-adaptive reparameterization (TARP)**. We consider two existing static decomposition methods:

- **Bilinear**, which takes $\Phi_j^l = U_j^l V_j^{l^T}$ where $U_j^l \in \mathbb{R}^{C_{\text{in}} \times r}$ and $V_j^l \in \mathbb{R}^{C_{\text{out}} \times r}$, as done in the additive-only setting ($\Phi_1^l = I$) by LoRA.

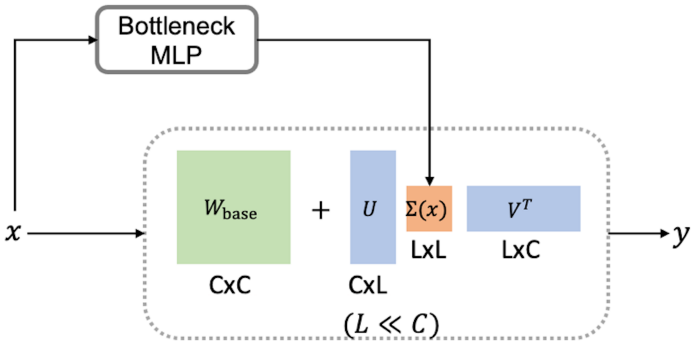



Figure 6.3: TARP with dynamic decomposition (only the additive $\Phi_2^l$ is depicted for simplicity).

- **Kronecker product**, which takes $\Phi_j^l = \sum_{k=1}^{n} H_k^l \otimes (U_k^l V_k^{l^T})$ where $H_k \in \mathbb{R}^{n \times n}$, $U_k^l \in \mathbb{R}^{(C_{\text{in}}/n) \times r}$, $V_k^l \in \mathbb{R}^{(C_{\text{out}}/n) \times r}$, and $n$ is a hyperparameter, as used in the "added-layer" Compacter approach.

In addition, we propose a novel decomposition inspired by the self-attention mechanism, which aggregate features using input-dependent attention weights. This can be regarded as a function with input-dependent parameters $y = f_{\theta(x)}(x)$. Similarly, the optimal reparameterization may vary with different input values. The computation of a reparameterized layer in the PLM becomes $y = f_{\theta(x)}(x) = [\Phi_1^l(x) \odot W_0^l + \Phi_2^l(x)]x$ where TARP parameters $\Phi_j^l(x)$ are modeled by a **dynamic low-rank decomposition** (Figure 6.3):

$$\Phi_j^l(x) = U_j^l \Sigma_j^l(x) V_j^{l^T}, \tag{6.5}$$

The square matrices $\Sigma_j^l(x) \in \mathbb{R}^{r \times r}$ are generated by a lightweight multi-layer perceptron (MLP) for different input vectors and $U_j^l, V_j^l$ are the learnable weight matrices with $r \ll \min(C_{\text{in}}, C_{\text{out}})$.

### 6.2.3 Efficient architecture adaptation.

We also propose adapting the model structure for each task in a data-driven manner, as shown in Figure 6.4 The weights of the *task-specific architectures* are learned in the inner loop, while the *task-aware architecture generator* which produces architecture candidates is learned in the outer loop. We term this approach **task-adaptive model structure (TAMS)** learning.

We first represent each task $\mathcal{T}_i$ with an embedding vector $z_i$ based on the task training set $D_i^{\text{train}}$. An embedding module $\mathcal{E}$ computes the task representation by aggregating features of all training

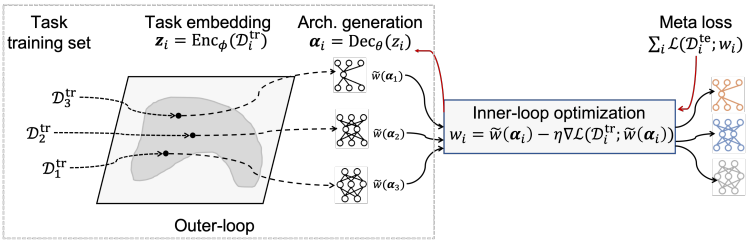



Figure 6.4: Illustration of the task-aware architecture generation from the task training data. For a specific task, the task training set is encoded as a task embedding vector by an encoder. A decoder maps the embedding vector to the architecture parameters which formulates a task-specific architecture. This architecture is initialized and tuned in the inner-loop optimization using the task training data. The encoder, decoder and initialization parameters will be updated by back-propagating the meta-gradient.

data:

$$z_i = \mathcal{E}(D_i^{\text{train}}) = \frac{\sum_{(x,y)\in D_i^{\text{train}}} \text{Embed}(x)}{|D_i^{\text{train}}|}, \tag{6.6}$$

where $\text{Embed}(x)$ are intermediate representations produced by the PLM. For encoder-decoder models (Transformer), we take Embed to be the encoder; for encoder-only or decoder-only PLMs (BERT, GPT-2), we use the token embedding layer.

Inspired by DARTS [161], we define the possible sublayer structures by a search space expressed as a directed acyclic graph (DAG), where each directed edge corresponds to a set of candidate operations $\mathcal{O}$. The task architecture is represented by a set of parameters $\alpha_i$ that encode the structure, where $\alpha_i \in \mathbb{R}^{E\times|\mathcal{O}|}$ ($E$ is the number of edges, and $|\mathcal{O}|$ is the number of operations). In our proposed TAMS approach we also introduce a controller $\mathcal{A}$ to generate these task-specific architecture parameters, as a function of the task embedding vector $\alpha_i = \mathcal{A}(z_i)$. The probability of choosing operation $m$ in edge $n$ is given by $P_n(m) = \text{softmax}_{m\in\mathcal{O}}(\alpha_i[n,m])$. In meta-testing, the discrete architecture is obtained by taking the argmax. Since argmax is non-differentiable, we use the straight-through Gumbel-Softmax estimator to backpropagate gradients for optimizing the architecture controller during meta-training.

The pseudo-code of our framework is provided in Algorithm 6. In TAMS, all possible architectures are initialized as part of the meta-parameters $\tilde{w}$ based on weight-sharing [198], i.e., architecture $\alpha_i$'s weights $\tilde{w}(\alpha_i)$ are selected from the meta-parameters. After the reparameterization steps in TARP and the architecture generation steps in TAMS, our inner loop optimization takes parameters $(\Phi, \tilde{w}(\alpha_i))$ and performs a small number of gradient steps $T_{\text{in}}$ on the task training set to give $(\Phi_i, \tilde{w}_i)$. In the outer loop optimization, we thus have to simultaneously optimize the architecture controller to perform architecture search, as well as the parameter initialization. This is in contrast to MAML which just optimizes the parameter initialization in the outer loop. The meta-loss

**Algorithm 6:** MLtD: Meta-Learning the Difference for Parameter and Data Efficient Language Model Adaptation.

**Require**: pretraining data $D^{\text{pre}}$; meta-training tasks $D^{\text{meta}}$; base model (LM) $f_\Theta$; TARP weights $\Phi$; embedding module $\mathcal{E}$ in TAMS; architecture controller $\mathcal{A}$ in TAMS; meta-parameters $\tilde{w}$ in TAMS; inner-/outer-loop learning rate $\eta_{\text{in}}/\eta_{\text{out}}$; meta-training iterations $T_{\text{meta}}$; inner-loop iterations $T_{\text{in}}$; meta-batch size $B$;

`/* Pretraining phase in MLtD */`

Pretrain the base LM's weights $\Theta \rightarrow \Theta_{\text{LM}}$ on $D^{\text{pre}}$;

`/* Meta-training phase in MLtD */`

**for** *each meta-iteration* $t \in [T_{meta}]$ **do**

  Sample a batch of tasks $\{\mathcal{T}_i\}_{i=1}^{B}$ from $D^{\text{meta}}$;

  $\mathcal{L}_{\text{meta\_loss}} = 0$;

  **for** $\mathcal{T}_i \in \{\mathcal{T}_i\}_{i=1}^{B}$ **do**

    Initialize $\Phi_i = \Phi$;

    Reparameterize $\Theta_{\text{LM}}$ with $\Phi_i$ using Eq.(6.3);

    Expand task-specific sublayers by TAMS-generated architecture $\alpha_i = \mathcal{A}(\mathcal{E}(D_i^{\text{train}}))$;

    Initialize sublayer's weights: $\tilde{w}_i = \tilde{w}(\alpha_i)$;

    **for** $T_{in}$ *inner-loop iterations* **do**

      $\left(\Phi_i, \tilde{w}_i\right) -= \eta_{\text{in}} \nabla_{(\Phi_i, \tilde{w}_i)} \mathcal{L}_{D_i^{\text{train}}}\left(f_{\Theta_{\text{LM}} \cup \Phi_i \cup \tilde{w}_i}\right)$;

    Evaluate on $D_i^{\text{test}}$: $\mathcal{L}_{\text{meta\_loss}} += \mathcal{L}_{D_i^{\text{test}}}\left(f_{\Theta_{\text{LM}} \cup \Phi_i \cup \tilde{w}_i}\right)$;

  Perform outer-loop optimization: $\left(\Theta_{\text{LM}}, \Phi, \mathcal{A}, \tilde{w}\right) -= \eta_{out} \nabla_{(\Theta_{\text{LM}}, \Phi, \mathcal{A}, \tilde{w})} \mathcal{L}_{\text{meta\_loss}}$;

**Return**: meta-trained PLM with learned $(\Theta_{\text{LM}}, \Phi, \mathcal{A}, \tilde{w})$;

becomes: $\min_{\mathcal{W}} \sum_{\mathcal{T}_i \sim p(\mathcal{T})} \mathcal{L}_{D_i^{\text{test}}}(f_{\Theta_{\text{LM}} \cup \Phi_i \cup \tilde{w}_i})$, where the tuple $\mathcal{W}$ contains the base PLM's weights $\Theta_{\text{LM}}$, the low-rank reparameterization weights $\Phi$, the architecture controller $\mathcal{A}$, and the weight-sharing meta-parameters $\tilde{w}$.

In summary, our contributions with the TAMS framework are that (1) it meta-learns a task-aware controller by training on the task distribution and then generalizes to new tasks by automatically generating an optimized architecture $\alpha_i$ from the task training data, and (2) it optimizes the controller and parameter initialization (shared by all tasks) simultaneously under a unified meta-learning objective. This is in contrast to DARTS, which performs a separate search and architecture parameter optimization for each task independently.

### 6.2.4 Few-shot dialogue personalization

Persona-Chat [291] is a dialogue generation benchmark with 1137/99/100 personas for training, validation, and testing respectively. We follow recent work [176, 221] and regard learning a dialogue model for each persona as a few-shot meta-learning task. On average, each persona has 8.3 unique dialogues, 6-8 turns per dialogue, and 15 words per turn. Following these works, we use a standard Transformer model with pretrained GLoVe embeddings and separate the dialogues by their persona description into meta-training/-validation/-testing using [176]'s splits. An illustration of the few-shot dialogue generation task on Persona-Chat is given in Figure 6.5.

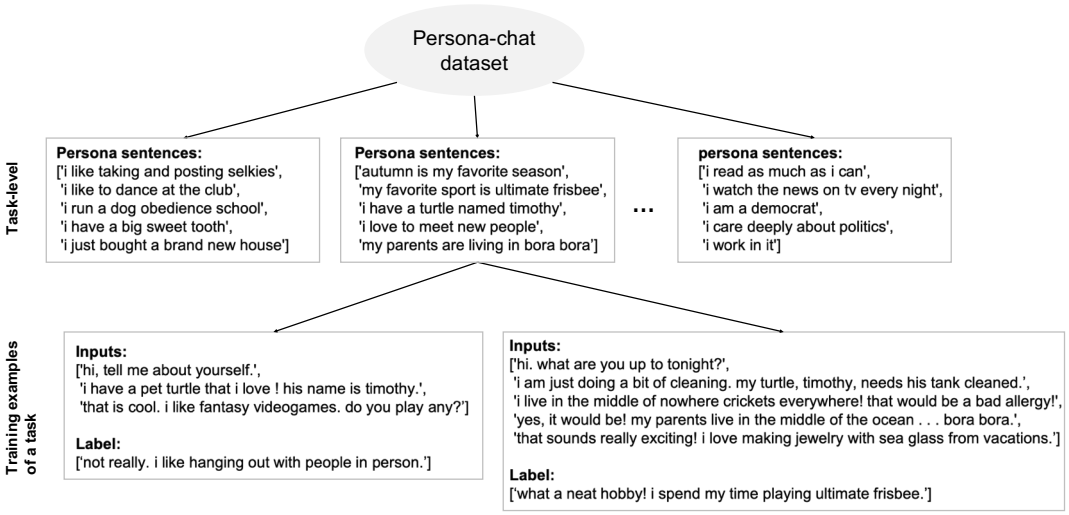



Figure 6.5: Illustration of the few-shot dialogue generation task on Persona-Chat dataset.

| **Method** | **PPL** | **BLEU** |
|---|---|---|
| Pretrain (multitask)* | 36.75 | 0.64 |
| Pretrain+Finetune* | 33.14 | 0.90 |
| MAML+Finetune* | 40.34 | 0.74 |
| CMAML+Finetune* | 36.30 | 0.89 |
| *Pretrain+Persona** | *30.42* | *1.00* |
| Pretrain+MAML+Finetune | 32.54 | 0.97 |
| MLtD (TARP only) | 32.15 | 0.99 |
| MLtD | **28.14** | **1.20** |

Table 6.1: Comparison of test perplexity (PPL; lower is better) and BLEU (higher is better) for few-shot dialogue generation on Persona-Chat dataset. *: published results from [176, 221]; the rest are ours.

**Baselines.** The following methods are from previous works: **Pretrain** denotes a multitask dialogue model trained on labeled data from all meta-training tasks. **MAML** meta-trains the Transformer model from scratch [176], and **CMAML** [221] additionally applies a pruning algorithm to customize the model structures for different tasks. +**Finetune** corresponds to finetuning on each testing task. Finally, **Pretrain+Persona** is a partial oracle for reference only, where the persona description is available to the model when generating the response.

**Results.** We include the same evaluation metrics from previous works (perplexity, BLEU score). Training MAML from scratch yields worse results than the Pretrain model. However, when MAML is initialized from the multitask model (**Pretrain+MAML+Finetune**), the result already outperforms previous work. Note that the *same labeled data* is used for both Pretrain and MAML, suggesting that meta-learning benefits from the more robust initialization that pretraining provides to improve task-specific few-shot adaptation.

Moreover, we see further improvements by "meta-learning the difference" (MLtD). By using

| Domain | # of tokens | | | |
|---|---|---|---|---|
| | Text only | Task corpus | | |
| | | train | val | test |
| Dialog | 44.96M | 27K | 74K | 75K |
| Email | 117.54M | 37K | 243K | 237K |
| Movie review | 11.36M | 633K | 1056K | 6193K |
| Debate | 122.99M | 59K | 188K | 197K |
| Social media | 153.30M | 68K | 229K | 229K |
| Science | 41.73M | 63K | 221K | 314K |

Table 6.2: Data sizes for AdaptSum [280] across the six domains, for both the text-only domain-related corpus and the low-resource task corpus.

| Method | Dialog | Email | Movie | Debate | Social | Science | Avg. |
|---|---|---|---|---|---|---|---|
| Baseline (full finetuning)* | 39.95 | 24.71 | 25.13 | 24.48 | 21.76 | 72.76 | 34.80 |
| DAPT (Domain-Adaptive Pre-Training)* | 41.22 | 26.50 | 24.25 | 26.71 | 22.95 | 71.88 | 35.59 |
| TAPT (Task-Adaptive Pre-Training)* | 40.15 | 25.30 | 25.27 | 24.59 | 22.81 | 73.08 | 35.20 |
| SDPT (Supervised Domain Pre-Training)* | 42.84 | 25.16 | 25.45 | 25.61 | 22.43 | 73.09 | 35.76 |
| MLtD | **44.81** | 25.30 | **26.83** | **26.88** | **24.40** | **74.03** | **37.04** |
| (TARP only) | 42.88 | **26.92** | 25.98 | 25.95 | 23.34 | 73.69 | 36.46 |
| (TARP only, no meta-learning) | 40.39 | 23.20 | 25.81 | 26.67 | 21.46 | 73.20 | 35.12 |
| (TARP only, multitask pretraining instead) | 41.82 | 25.41 | 26.17 | 25.70 | 22.54 | 73.50 | 35.85 |

Table 6.3: ROUGE F1s from multi-domain adaptation for abstractive summarization on AdaptSum (higher is better). All methods are initialized with pretrained BART and finetuned on the labeled task training set of each domain at the end. *: published results from [280], using DAPT and TAPT methods from [75]; the rest are ours.

TARP for MAML's inner loop adaptation (**MLtD, TARP only**), we attain equivalent or better results and faster training time while only updating a small amount of task-specific parameters. This indicates that our method helps mitigate overfitting to low-resource tasks. Finally, by further incorporating TAMS (**MLtD**), we use the full framework and achieve the best performance, suggesting the task-adapted model structure search gives better architectures for different personas.

### 6.2.5 Low-resource abstractive summarization

AdaptSum [280] is a new multi-domain dataset used to evaluate domain adaptation schemes for abstractive summarization. It consists of six diverse target domains ranging from movie reviews to scientific abstracts. Each domain has a low-resource task corpus and a larger unlabeled text corpus as well (list and statistics in Table 6.2) that is used to evaluate domain- and task-adaptive pretraining (DAPT/TAPT [75]). We use pretrained BART [138] and finetune on each low-resource task corpus.

**Baselines.** **DAPT** continues pretraining with the self-supervised objective [138] using the unlabeled domain corpus. **TAPT** continues pretraining with the set of unlabeled documents found in the target summarization task. **SDPT** uses the XSum dataset in the News domain to further pretrain BART with a supervised training objective using document-summary pairs before finetuning.

| Method | Dialog | Email | Movie | Debate | Social | Science | Avg. |
|---|---|---|---|---|---|---|---|
| Baseline (full finetuning) | 31.95 | 31.57 | 42.25 | 34.38 | 33.02 | 28.82 | 33.67 |
| Zero-shot (no finetuning) | 37.26 | 38.45 | 49.46 | 41.38 | 37.13 | 34.20 | 39.65 |
| DAPT† | 35.15 | **16.04** | 43.12 | 33.83 | 27.15 | 18.96 | 29.04 |
| MLtD | 29.66 | 16.93 | **35.38** | **30.61** | **19.78** | **17.06** | **24.90** |
| (TARP only) | **28.63** | 18.67 | 39.73 | 32.70 | 26.93 | 20.39 | 27.84 |
| (TARP only, no meta-learning) | 31.66 | 31.59 | 41.78 | 33.18 | 32.78 | 28.20 | 33.19 |

Table 6.4: Test perplexities from multi-domain language modeling adaptation on AdaptSum (lower is better). All methods are initialized with pretrained GPT-2 medium and finetuned on the labeled domain set at the end. †: our re-implementation of [75].

**Results.** We find that **MLtD**, even without architecture search (**TARP only**), outperforms DAPT, TAPT, and SDPT. These methods use in-domain or task knowledge and the standard pretraining objective to improve adaptation to the target task, while our method considers cross-domain knowledge via the meta-learning objective, sampling meta-training tasks from multiple domain corpora to train the model. Moreover, the use of meta-learning as preparation outperforms multitask pretraining (**TARP only, multitask pretraining instead**), signifying that mere exposure to the cross-domain data may not be enough and using a meta-learning objective to explicitly optimize the model weights for the lightweight adaptation is beneficial. Finally, we see that without meta-learning or multitasking (**TARP only, no meta-learning**) our performance is also better than the baseline. This demonstrates the effectiveness of the lightweight TARP adaptation, which matches the performance of full finetuning while only updating less than 5% of parameters.

### 6.2.6 Multi-domain language modeling

Though the text corpora in AdaptSum were originally included to perform domain-adaptive pretraining, we also use them to evaluate our methods on multi-domain language modeling. As this is a novel benchmark, to demonstrate fast adaptation we take $T_{\text{in}} = 1$, i.e., we only finetune the model for 1 epoch on unseen testing tasks.

**Baselines.** We start with pretrained GPT-2 medium (345M parameters) [202] with input sequence length 512 using the Transformers library [256]. Finetuning is performed on the training documents of the task corpus, and we evaluate perplexity on the test documents of the task corpus. The only exception to finetuning is **Zero-shot** which evaluates the pretrained GPT-2 model directly. **DAPT** continues pretraining of GPT-2 with the language modeling objective on the unlabeled domain corpus before finetuning.

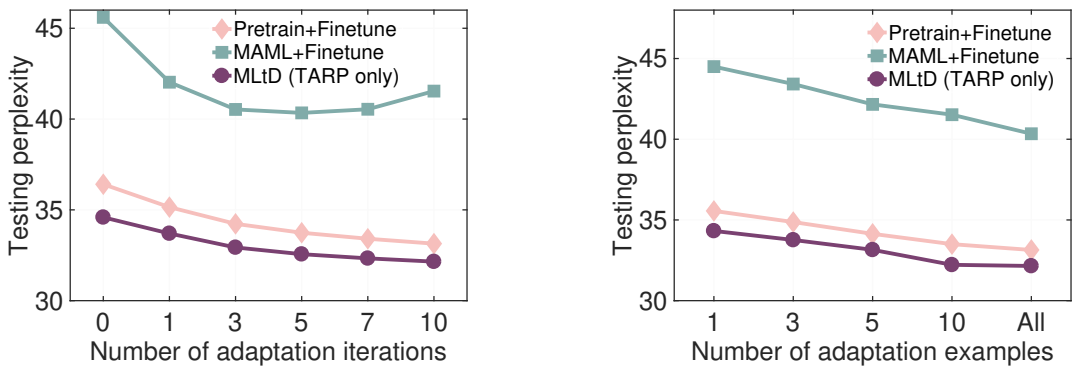



Figure 6.6: Perplexities on Persona-Chat testing tasks with MLtD (TARP only) versus Pretrain+Finetune and MAML+Finetune. **Left**: Influence of number of adaptation iterations. **Right**: Influence of the number of adaptation dialogues.

**Results.** Our findings on the low-resource abstractive summarization tasks also hold for the unconditional causal language modeling tasks. Namely, we see equal or better performance of TARP vs. full finetuning and that meta-learning plays a significant role in the task adaptation quality. In contrast to the summarization tasks, we see that TAMS leads to noticeable improvements. We will explain why this may be the case and present the searched sublayer modules by TAMS in the analyses section.

### 6.2.7 Analyses

**Pretraining improves meta-learning.** We analyze the performance of MLtD on Persona-Chat at meta-testing time (i.e., finetuning then running inference on unseen personas) with respect to the number of inner loop steps and training dialogues. In Figure 6.6(left), we see that original MAML (no pretraining) overfits, while finetuning the multitask-pretrained model keeps improving. Moreover, MLtD atop the multitask-pretrained model followed by finetuning continues to improve test perplexity. In Figure 6.6(right), we fix the finetuning steps and vary the number of training dialogues used in finetuning. Using more dialogues improves perplexity for all three methods, with MLtD still leading over full MAML and direct finetuning after pretraining. The takeaway from these results is that applying MAML on a pretrained model prevents overfitting and promotes better generalizability from meta-learning.

**Dynamic TARP vs. other parameter-efficient transfer methods.** We benchmark our dynamic low-rank reparameterization on a variety of NLP models and tasks. To show that dynamic TARP individually improves on full finetuning and other parameter-efficient adaptation methods, we

| Method | Params. per task | E2E BLEU | E2E NIST | E2E METEOR | E2E ROUGE-L | E2E CIDEr | DART BLEU | WebNLG BLEU |
|---|---|---|---|---|---|---|---|---|
| Finetuning (full)* | 100% | 68.2 | 8.62 | 46.2 | 71.0 | 2.47 | 46.0 | 47.6 |
| FT-Top2* | 7.1% | 68.1 | 8.59 | 46.0 | 70.8 | 2.41 | 38.1 | 33.5 |
| BitFit* | 0.1% | 67.2 | 8.63 | 45.1 | 69.3 | 2.32 | 43.3 | 50.5 |
| Adapter* | 3.2% | 68.9 | 8.71 | 46.1 | 71.3 | 2.47 | 45.4 | 54.0 |
| Bilinear TARP | 2.4% | 68.8 | 8.75 | 46.1 | 70.8 | 2.43 | 46.7 | 54.0 |
| Kronecker TARP | 2.4% | 68.2 | 8.73 | 45.2 | 69.4 | 2.36 | 45.6 | 53.1 |
| Dynamic TARP | 1% | **69.7**$_{\pm 0.1}$ | **8.78**$_{\pm 0.02}$ | **46.9**$_{\pm 0.2}$ | **72.1**$_{\pm 0.1}$ | **2.51**$_{\pm 0.01}$ | **47.9**$_{\pm 0.2}$ | **55.3**$_{\pm 0.1}$ |

Table 6.5: Comparison with other parameter-efficient transfer methods for natural language generation (E2E, DART, WebNLG) on GPT-2 medium. *: published results from [112, 285]; the rest are ours. For our dynamic TARP, we provide the 95% confidence interval over 5 runs.

report single-task results here. For generative tasks, we use pretrained GPT-2 medium on natural language generation datasets: we specifically evaluate on E2E [190], WebNLG [65] and DART [185]. For classification, we use pretrained RoBERTa [165] on low-resource GLUE [243] tasks, which were evaluated on by many recent parameter efficient adaptation methods. We chose the rank $r$ to give similar parameter counts to other approaches: $r = 4$ in Table 6.5 and $r = 8$ in Table 6.6.

For generative tasks, we compare with **finetuning** all layers; **FT-Top2**, which only finetunes the last two layers of the model; **BitFit** [285], which only finetunes the biases; and **Adapter tuning** [112], which only finetunes the adapter layers inserted after each feed-forward and self-attention sublayer. As shown in Table 6.5, TARP methods match or outperform other parameter-efficient methods, while learning task-specific parameters that are <3% of the number of base parameters and keep the base model unchanged. Among the three TARP variants, we find that **Dynamic** > **Bilinear** > **Kronecker** in terms of performance across generative metrics. This suggests that the optimal adaptation to the underlying model weights may vary per token, which dynamic low-rank accounts for.

For classification tasks, we primarily compare with [79], which proposed a unified framework connecting several state-of-the-art adaptation methods [112, 115, 147] and devised an improved method (**MAM Adapter**). Our dynamic TARP can only partly be viewed in this unified framework as we explore a novel design dimension, i.e., making the modification to the base model dynamic w.r.t. input tokens. Moreover, in contrast to additive-only modifications in [79], our dynamic TARP applies both multiplicative and additive modifications. In Table 6.6, dynamic TARP introduces and trains only 1% versus the number of original parameters, while achieving comparable results to full finetuning (+0.3 abs.) and outperforming the previous best, MAM adapters (+1.0 abs.).

Tables 6.5 and 6.6 also show that our lightweight TARP method outperforms full finetuning on corresponding evaluating metrics, while being faster as it only adapts a tiny amount of weights. The

| Method | Params. per task | CoLA Matt. corr | MRPC Acc. | STS-B Pear. corr | RTE Acc. | SST-2 Acc. | MNLI Acc. | QNLI Acc. | QQP Acc. | Avg. |
|---|---|---|---|---|---|---|---|---|---|---|
| Finetuning (full)* | 100% | 63.6 | 90.2 | 91.2 | 78.7 | 94.8 | **87.6** | 92.8 | 91.9 | 86.4 |
| Masking* | 3.1% | 60.3 | 88.5 | - | 69.2 | 94.5 | - | 92.4 | - | - |
| BitFit* | 0.1% | 61.8 | **92.0** | 90.8 | 77.8 | 93.7 | 85.2 | 91.3 | 84.5 | 84.6 |
| LoRA† | 1% | 62.2 | 89.6 | 90.6 | 72.9 | 94.8 | 87.4 | 92.4 | 92.3 | 85.3 |
| AdapterFusion* | 1% | - | 89.7 | - | 78.8 | 93.7 | 86.2 | - | 90.3 | - |
| MAM Adapter* | 0.5% | 59.2 | 88.5 | 90.6 | 74.3 | 94.2 | 87.4 | 92.6 | 90.2 | 84.6 |
| MAM Adapter† | 1% | 63.7 | 89.4 | 90.6 | 76.6 | 94.4 | **87.6** | 92.8 | 90.9 | 85.7 |
| Dynamic TARP | 1% | **64.5**$_{\pm.7}$ | 89.9$_{\pm.3}$ | **91.3**$_{\pm.2}$ | **79.1**$_{\pm.6}$ | **94.9**$_{\pm.1}$ | **87.6**$_{\pm.2}$ | **93.3**$_{\pm.1}$ | **93.1**$_{\pm.1}$ | **86.7** |

Table 6.6: Comparison with other adaptation methods on low-resource GLUE tasks. We report single-task results (including our dynamic TARP) of adapting RoBERTa base on the training set of each task only. *: published results from [165, 296, 285, 197, 79]; †: recreated using [79]'s implementation; the rest are ours. For our dynamic TARP, we provide the 95% confidence interval over 5 runs.

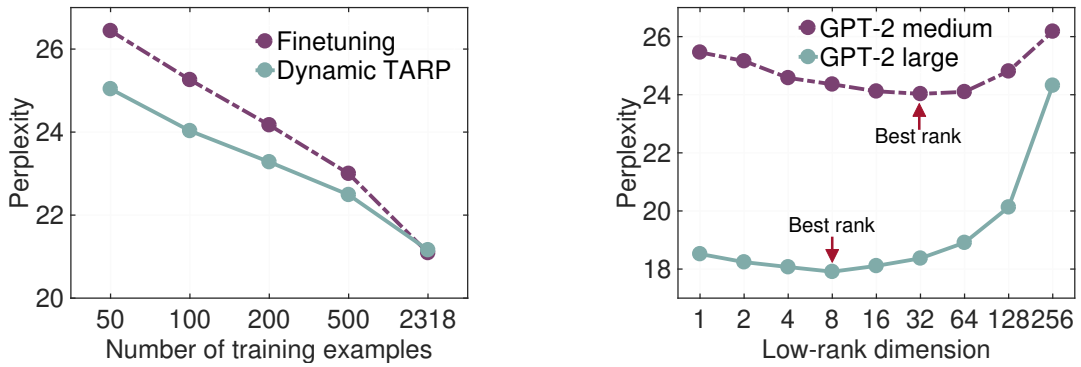
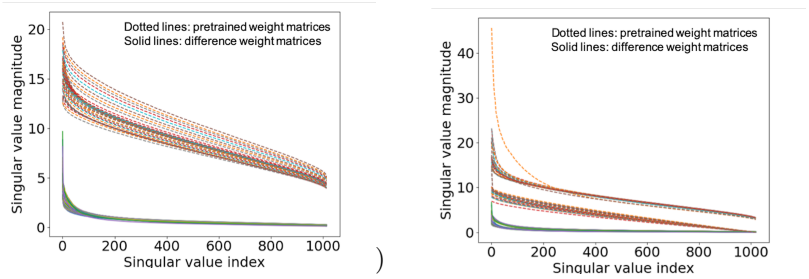



Figure 6.7: Testing perplexities on WikiText-2 with full finetuning and/or low-rank adaptation with dynamic TARP. **Left**: Low-rank adaptation is extremely helpful on low-resource tasks. **Right**: Holding the number of training examples fixed, the model adaptation space is optimized by fewer dimensions.

training time further improves through utilizing the training data more efficiently. In Figure 6.7(left) we compare perplexities of our method against finetuning on subsets of WikiText-2 and see that finetuning increasingly underperforms as the number of examples decrease. To explain this behavior, in Figure 6.7(right) we fix the number of training examples to 100 and ablate the rank. Our method performs best with a very small rank value, suggesting that the difference between the pretrained and finetuned weight matrices lies in a lower-dimensional subspace. This complements [2]'s observation that direct adaptation in lower-dimensional spaces can be equally as effective as in the original space. This is also illustrated by Figure 6.8, showing that the different weight matrices (post-adaptation weight minus pre-adaptation weight) have only a few major singular values. Moreover, we find that the larger the model (GPT-2 medium vs. the GPT-2 small), the lower the rank value required for the best adaptation performance.

**TAMS discovers better architectures.** Recent studies have shown that simple modifications to the transformer architecture, such as re-organizing the MHSA and FFN modules [297] or adding




(a) Singular values of FFN weights in GPT-2 model.

(b) Singular values of MHSA weights in GPT-2 model.

Figure 6.8: Singular values of the pretrained GPT-2 weights (dotted lines) and the difference weights (solid lines) after adaptation on WikiText-2 dataset. The difference weights have only a few major singular values.

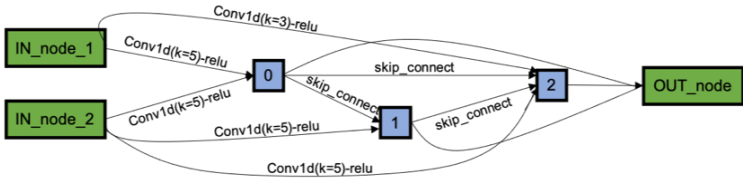



Figure 6.9: Dominant structure of the TAMS-learned sublayers (augmented to the original FFN modules in GPT-2) for AdaptSum language modeling.

1D convolutions to self-attention [217], improve the task performance. Similarly, from our results in Table 6.4, adapting the model structure through sub-layer modifications in our meta-learning framework further reduces the testing perplexity compared to MLtD with task-agnostic fixed model structure. Applying task-aware architecture search (TAMS) on the FFN module incurs less than 5% additional model parameters compared to the original GPT-2 model, but reduces the perplexity by 3 points on average. A limitation we observe is that the TAMS method tends to produce a dominant architecture (Figure 6.9) as opposed to one different architecture for each task. We conjecture this may be because our initial task representation strategy has low variance due to averaging across the entire task training data. This may explain why TAMS did not uniformly improve MLtD in all settings. Nevertheless, the perplexity reduction implies that there is still room to optimize the architecture of current LMs without significantly increasing total model size. Thus, we believe task-aware architecture search is a promising direction to continue to invest in the future.

**Training efficiency of MLtD.** Since meta-learning explicitly optimizes the model for fast adaptation, and because our lightweight TARP method only updates a small set of weights, our method requires fewer epochs to reach convergence, and thus takes the least of time to adapt to a task compared to alternative methods, as shown in Figure 6.16.

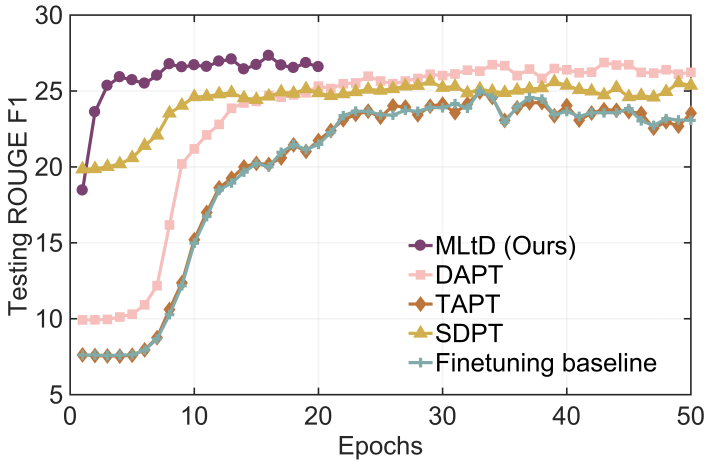



Figure 6.10: Convergence analysis for finetuning BART models obtained by different methods on AdaptSum, using the Debate domain as an example. Our MLtD demonstrates the best training efficiency.

## 6.3 Semi-supervised few-shot learning

### 6.3.1 Background

The fundamental challenge for few-shot learning is the difficulty to estimate the task data distribution of unseen categories with just a handful of labeled training examples. Although meta-learning methods explicitly optimize the model for few-shot adaptation by constructing many few-shot tasks during training to accumulate the task-level inductive bias, these methods have limited flexibility to handle testing tasks that are very different from the training tasks. On the other hand, semi-supervised learning also aims to reduce the reliance on labeled examples and exploits additional unlabeled data to improve the performance. Recent few-shot learning approaches have begun exploring transductive [164, 207, 117, 45, 18] and semi-supervised learning [145, 206, 251] to solve few-shot tasks, by using auxiliary information from unlabeled examples (Figure 6.11). Self-training [203] is one of the most straight-forward approaches to utilize the unlabeled data: we can first train the model with the few-shot labeled data, then use the model to infer the pseudo-labels (class that has the maximum predicted probability) of the unlabeled data, and finally augment the pseudo-labeled data to the few-shot labeled set to retrain the model. Self-training, however, suffers from two limitations: (1) since the model is trained with very limited labeled examples, the model may not capture the data distribution and the generated. pseudo-labels will have low quality with significant label noise; (2) there is no sample selection strategy to filter out the label noise, and including the wrongly labeled examples into training will jeopardize the final model performance, counteracting our goal of using unlabeled data to improve the accuracy.

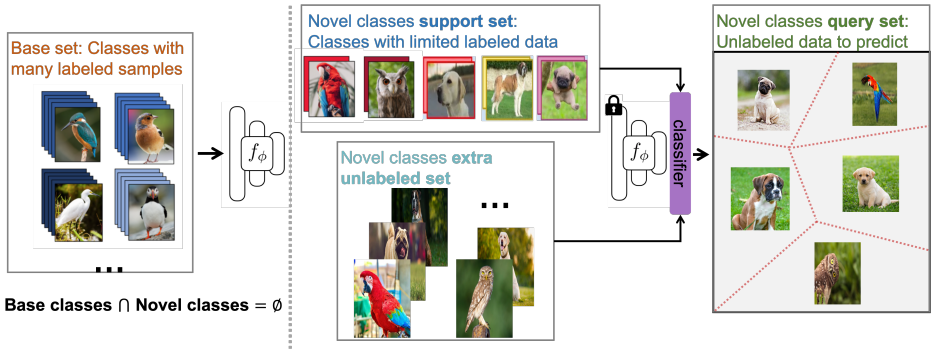



Figure 6.11: Illustration of the semi-supervised few-shot learning.

To rectify the limitations in both the meta-learning and the self-training approaches, we propose a novel semi-supervised few-shot learning framework. Firstly, we propose an unsupervised *Dependency Maximization* loss to better use the unlabeled data to enhance the model training and generate higher quality pseudo-labels. The proposed loss maximizes the statistical dependence between the embedded features of the unlabeled data and their softmax predictions. We further propose an *Instance Discriminant Analysis* to evaluate the pseudo-labeled data from the perspective of feature discriminant power and select the most faithful pseudo-labels to augment the few-shot labeled training set, in order to alleviate the accumulation of label noise. To speed up our sample selection process, we also derive an efficient approximation for our discriminant criterion. Experiments demonstrate that the proposed framework outperforms state-of-the-art meta-learning, transductive learning, and semi-supervised learning methods on a variety of few-shot benchmarks, including mini-ImageNet and tiered-ImageNet. Moreover, our framework also achieves the best accuracy on the challenging cross-domain few-shot learning scenario.

### 6.3.2 Overview

We first pretrain a CNN feature extractor $f_\theta$ on a labeled base dataset $\mathcal{X}_{base} = \{(\mathbf{x}_i, \mathbf{y}_i), \mathbf{y}_i \in \mathcal{Y}_{base}\}$, where $\mathcal{Y}_{base}$ denotes the set of classes in the base dataset. The testing few-shot tasks are sampled from a novel dataset $\mathcal{X}_{novel} = \{(\mathbf{x}_i, \mathbf{y}_i), \mathbf{y}_i \in \mathcal{Y}_{novel}\}$ with completely unseen categories $\mathcal{Y}_{novel}$, i.e., the base and novel datasets have mutually disjoint classes $\mathcal{Y}_{base} \cap \mathcal{Y}_{novel} = \emptyset$. We follow the standard $N$-way $K$-shot formulation [164, 207, 117, 45, 18, 145, 206, 251]. Each few-shot task $\mathcal{T}_i$ has $N$ classes that are randomly sampled from $\mathcal{Y}_{novel}$ without replacement. Each class has $K$ labeled examples, which are referred as the *support set* $\mathcal{D}^S_{\mathcal{T}_i}$ ($|\mathcal{D}^S_{\mathcal{T}_i}| = N \times K$). Each task also has a *query set* $\mathcal{D}^Q_{\mathcal{T}_i}$, which consists of $Q$ unlabeled and unseen examples from the same $N$ classes ( $|\mathcal{D}^Q_{\mathcal{T}_i}| = Q \times K$). The unlabeled query set served as the testing set in the few-shot task. In semi-supervised learning, there

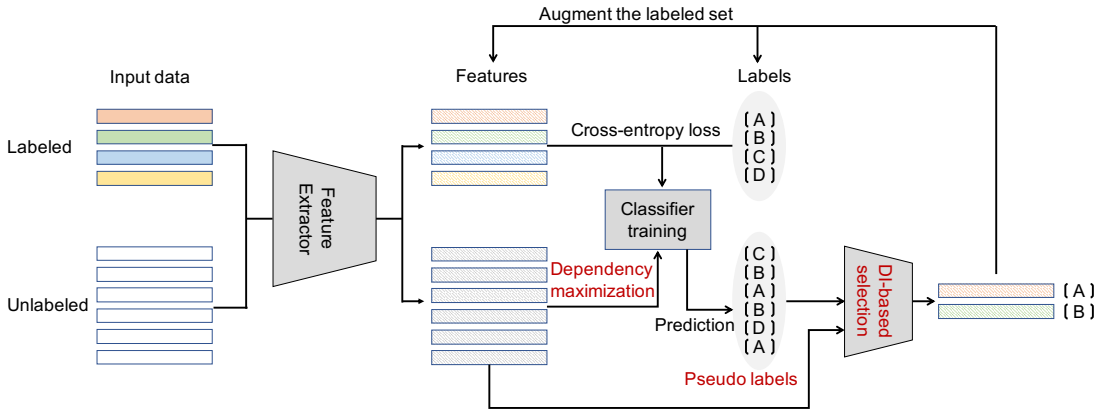



Figure 6.12: Illustration of the proposed semi-supervised few-shot learning framework with the unsupervised dependency maximization loss for classifier training and the instance discriminant analysis for pseudo-label selection.

is also an additional unlabeled set $D^U_{\mathcal{T}_i}$. An illustration of our framework is presented in Figure 6.12. When solving a specific few-shot task with support set $D^S_{\mathcal{T}_i}$ and unlabeled set $D^U_{\mathcal{T}_i}$, we perform the following steps:

1. Append a randomly initialized linear classifier to the end of the pretrained feature extractor.
2. Train the classifier using the supervised loss and the dependency maximization loss jointly.
3. Generate the pseudo-labels of the unlabeled set.
4. Evaluate the credibility of the pseudo-labeled examples based on the instance discriminant analysis and select the most trustworthy ones to augment the support set.
5. Repeat step 2 - 4 until the generated pseudo-labels becomes stable.
6. Use the final classifier to perform inference on the query set.

### 6.3.3 Unsupervised dependency maximization loss.

The dependency maximization (DM) loss is differentiable and is optimized with standard gradient descent to enhance the classifier training. DM loss maximizes the statistical dependence between the features of the unlabeled set and their label predictions, in conjunction with minimizing the cross-entropy loss, as shown in Figure 6.13. DM loss can be viewed as a surrogate of the empirical risk on the unlabeled data, which helps to restrict the hypothesis space and regularizes the prediction on the unlabeled data.

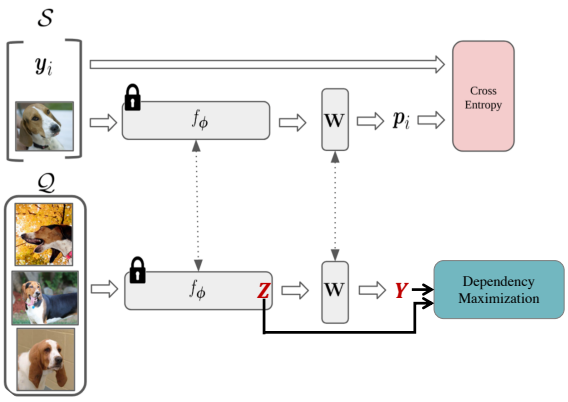



Figure 6.13: The semi-supervised classifier training involves minimizing the cross-entropy loss on the few-shot labeled examples ($\mathcal{S}$) and maximizes the dependency loss on the unlabeled examples ($\mathcal{Q}$).

We begin by listing some notations before introducing how to measure the dependence between features and label predictions. Let $Z$ be a random vector (of size $\mathbb{R}^d$ where $d$ is the feature dimension) associated with the embedded features of the unlabeled set, $Y$ denotes the random vector (of size $\mathbb{R}^N$ where $N$ is number of classes) associated with their softmax predictions. $P_{Z,Y}$ is the joint distribution between these two random variables. To measure the dependence between $Z$ and $Y$, we define the cross-covariance operator based on [9]:

$$C_{zy} := \mathbb{E}_{zy}[(\Phi(z) - \mu_z) \otimes (\Psi(y) - \mu_y)], \tag{6.7}$$

where $\Phi : \mathcal{Z} \to \mathcal{F}$ (or $\Psi : \mathcal{Y} \to \mathcal{G}$) defines a kernel mapping from the latent feature space (or prediction space) to a reproducing kernel Hilbert space (RKHS) $F$ (or $G$), with mean vectors defined as $\mu_z$ (or $\mu_y$). To summarize the degree of dependence between $Z$ and $Y$, we use the Hilbert-Schmidt norm of $C_{zy}$, which is given by the trace of $C_{zy}C_{zy}^T$. In this paper, we use the square of the Hilbert-Schmidt norm of the cross-covariance operator, $\|C_{zy}\|_{HS}^2$, because it can detect nonlinear dependence, as shown in the following theorem:

**Theorem 3** ($C_{zy}$ and Independence [67])**.** *Assume $F$ and $G$ are RKHSs with characteristic kernels. Then, we have $\|C_{zy}\|_{HS}^2 = 0$ if and only if $Z$ and $Y$ are independent.*

Characteristic kernels such as Gaussian kernel, i.e. $k(x, x') = \exp\big(-\|x - x'\|_2^2/(2\sigma^2)\big)$, allows us to measure any dependence between $Z$ and $Y$. In our case, $\|C_{zy}\|_{HS}^2$ is zero only if the features and the label predictions of the unlabeled set are independent. Clearly, we aim to achieve the opposite. We aim to maximize the dependence between the features and predictions by maximizing $\|C_{zy}\|_{HS}^2$.

To utilize the dependence measure as a loss function, we need an empirical estimate from finite

number of samples. Denote the kernel functions associated with the RKHS $F$ and $G$ by $k(z, z')$ and $l(y, y')$; let $\mathbf{K}, \mathbf{L} \in \mathbb{R}^{U \times U}$ be the Gram matrices defined on the features and softmax predictions associated with the unlabeled set $\mathcal{D}^U_{\mathcal{T}_i}$, containing entries $\mathbf{K}_{i,j} = k(z_i, z_j)$ and $\mathbf{L}_{i,j} = l(y_i, y_j)$. The empirical estimator of $\|C_{zy}\|^2_{HS}$ is given by:

$$\widehat{\|C_{zy}\|^2_{HS}} := (U-1)^{-2} \operatorname{tr}(\mathbf{KHLH}), \tag{6.8}$$

where $\mathbf{H} = \mathbf{I}_U - (1/U)\mathbf{1}_U\mathbf{1}_U^T$ is the centering matrix, $\mathbf{I}_U$ is an identity matrix, $\mathbf{1}_U$ is a vector with all ones, and $\operatorname{tr}(\cdot)$ is the matrix trace operation. We show by the following theorem [67] that this empirical estimator converges sufficiently:

**Theorem 4** (Bound on empirical DM loss.)**.** *Assume $k$ and $l$ are bounded almost everywhere by 1, and are non-negative. Then, with constants $\alpha^2 > 0.24$ and $C$, for $U > 1$ and all $\delta > 0$, with probability at least $1 - \delta$ for all $P_{ZY}$, we have*

$$\left|\widehat{\|C_{zy}\|^2_{HS}} - \|C_{zy}\|^2_{HS}\right| \leq \sqrt{\frac{log(6/\delta)}{\alpha^2 U}} + \frac{C}{U} \tag{6.9}$$

The overall training objective is obtained by adding the empirical dependence measure on the unlabeled set to the supervised cross-entropy loss on the support set:

$$\min_{\mathbf{W},\mathbf{b}} \underbrace{-\frac{1}{NK} \sum_{(x,y)\in\mathcal{D}^S_{\mathcal{T}_i}} \log \frac{\exp(\mathbf{W}_y^T f_\theta(\mathbf{x}) + \mathbf{b}_y)}{\exp(\sum_{c=1}^N \mathbf{W}_c^T f_\theta(\mathbf{x}) + \mathbf{b}_c)}}_{\text{Cross-entropy minimization on support set}} \underbrace{-\lambda \cdot (U-1)^{-2} \operatorname{tr}(\mathbf{KHLH})}_{\text{Dependency maximization on unlabeled set}}, \tag{6.10}$$

where $\mathbf{W}, \mathbf{b}$ denote the weight and bias of the softmax linear classifier $h_\phi$, and the label prediction is given by $\hat{\mathbf{y}} = h_\phi(\mathbf{z}) = \text{softmax}(\mathbf{W}^T\mathbf{z} + \mathbf{b}) = \text{softmax}(\mathbf{W}^T f_\theta(\mathbf{x}) + \mathbf{b})$.

Objective (6.10) is optimized for each test task using gradient descent w.r.t. $\mathbf{W}$ and $\mathbf{b}$. The pretrained feature extractor $f_\theta$ is frozen. $\mathbf{W}$ and $\mathbf{b}$ are initialized based on the class prototypes computed over the support set: $\mathbf{W}^0 = [2\boldsymbol{\mu}_1, ..., 2\boldsymbol{\mu}_N] \in \mathbb{R}^{d\times N}$ and $\mathbf{b}^0 = [-\|\boldsymbol{\mu}_1\|_2^2, ..., -\|\boldsymbol{\mu}_N\|_2^2] \in \mathbb{R}^{N\times 1}$. Then, the weight and bias parameters are updated using both the support and unlabeled samples in the few-shot task without mini-batch sampling.

### 6.3.4 Instance discriminant analysis for pseudo-label selection.

After training the classifier with our proposed DM loss in Eq.(6.10), we can predict the labels $\hat{y}_u$ for the unlabeled examples in $\mathcal{D}^U_{\mathcal{T}_i}$ as their pseudo-labels. In this section, we present an instance discriminant analysis (IDA) algorithm to evaluate the quality of these pseudo-labeled examples and select the most trustworthy ones to augment few-shot training set.

IDA is a sample selection or outlier removal algorithm that aims to remove a subset of training

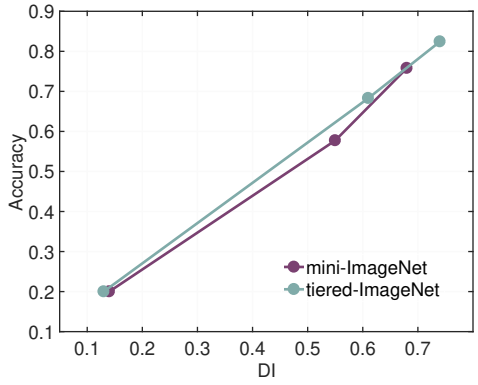



Figure 6.14: The feature discriminant power metric (measured by the normalized $\psi$ in Eq.(6.11)) can serve as a surrogate for the pseudo-labels' actual accuracy on the unlabeled set. Thus, we use $\psi$ in Eq.(6.11) as a metric for pseudo-label selection.

sample a priori, and train the classifier with the remaining subset. We find that the feature discriminant power computed using the latent features and pseudo-labels can be used as a surrogate for the pseudo-labels' quality, as shown in Figure 6.14. The results suggest that a wrongly labeled example would be detrimental to the overall data separability of the unlabeled set, causing low feature discriminant power; while a correctly labeled example would facilitate the data separability, thus improving the feature discriminant power.

Inspired by Fisher's discriminant analysis, we evaluate the quality of each pseudo-labeled example by computing its contribution to the overall data separability using Fishers Criterion. Denote the pseudo-labeled examples by $\{(\mathbf{x}_u, \hat{y}_u)|\mathbf{x}_u \in \mathcal{D}^U_{\mathcal{T}_i}\}$. Note that the true labels are not known, but we use the pseudo labels for the following computation. Let $f(\mathbf{x}_u)$ denote the embedded feature of instance $u$ (for notation simplicity, we omit $\theta$ for the feature extractor). We define the scatter matrix and the between-class scatter matrix as $\bar{\mathbf{S}} = \sum_{u=1}^{U}(f(\mathbf{x}_u) - \boldsymbol{\mu})(f(\mathbf{x}_u) - \boldsymbol{\mu})^T$ and $\mathbf{S}_B = \sum_{c\in\{1,\dots,N\}} M_c(\boldsymbol{\mu}_c - \boldsymbol{\mu})(\boldsymbol{\mu}_c - \boldsymbol{\mu})^T$, where $\boldsymbol{\mu}$ is the mean of all embedded features associated with the pseudo-labeled set, $M_c$ is the number of instances belonging to class $c$, $\boldsymbol{\mu}_c$ is the mean vector of the embedded features belonging to class $c$, and $N$ is the number classes in the given few-shot task. Then, the Fishers Criterion ($\psi$) is defined as the ratio of the between-class scatter matrix to the scatter matrix:

$$\psi := \mathrm{tr}\{\bar{\mathbf{S}}^{-1}\mathbf{S}_B\}, \tag{6.11}$$

where $\mathrm{tr}(\cdot)$ denotes the matrix trace operation. To explain more, the eigen-vectors of matrix $\bar{\mathbf{S}}^{-1}\mathbf{S}_B$ compose the optimal space that maximises the between-class separability while minimising the within-class variability. The Fishers Criterion, calculated as the summation of the corresponding

eigen-values, is regarded as a measure of the overall data separability.

Next, we evaluate the credibility of each pseudo-labeled instance $(\mathbf{x}_u, \hat{y}_u)$ by measuring its contribution to the overall discriminant power, i.e. to measure the difference of Fishers Criterion value when the instance is present and the instance is removed. The impact of removing a specific example on the discriminant power $\psi$ is given by:

$$d\psi_u := \text{tr}\{\bar{\mathbf{S}}^{-1}\mathbf{S}_B\} - \text{tr}\{[\bar{\mathbf{S}}\neg u]^{-1}\mathbf{S}_B\neg u\}, \tag{6.12}$$

where $\bar{\mathbf{S}}\neg u$ and $\mathbf{S}_B\neg u$ are derived from the remaining data after removing instance $u$. $d\psi_u$ captures the reduction in the feature discriminant power caused by removing instance $u$, and it can be used as a metric for our sample selection process. Larger $d\psi_u$ indicates that the instance has greater (positive) impact to the data separability, thus its pseudo-label is more trustworthy and the instance should be selected to the augmented support set. We sort the pseudo-labeled examples in the descending order of their $d\psi_u$ value, and only select the top-ranking examples.

The exact computation of $d\psi_u$ can be expensive, since it requires multiple matrix inverse. In order to perform our IDA-based sample evaluation and selection more efficiently, we provide the following approximation of the $d\psi_u$, which can be computed without any matrix operations, with only inner-product and scaler operations:

**Proposition 2** (Bound on $d\psi_u$)**.** *Instance Discriminant Analysis (IDA) $d\psi_u$ of a sample $u$ is upper-bounded by:*

$$d\psi_u \leq \frac{\delta f(\mathbf{x}_u)^T f(\mathbf{x}_u)}{\rho(f(\mathbf{x}_u)^T f(\mathbf{x}_u) - \rho)} + \frac{H_{4,1/2}(\nu_u + f(\mathbf{x}_u)^T f(\mathbf{x}_u))}{\rho(M_u - 1)} + \frac{f(\mathbf{x}_u)^T f(\mathbf{x}_u)(\nu_u + f(\mathbf{x}_u)^T f(\mathbf{x}_u))}{\rho(f(\mathbf{x}_u)^T f(\mathbf{x}_u) - \rho)(M_u - 1)} \tag{6.13}$$

*where $\delta = \sum_{c\in\{1,\ldots,N\}} M_c \boldsymbol{\mu}_c^T \boldsymbol{\mu}_c$; $\rho > 0$ is the ridge parameter; $M_u$ is the number of examples sharing the same pseudo label as $\mathbf{x}_u$ in the dataset (including $\mathbf{x}_u$); $H_{4,1/2} = \sum_{k=1}^{4} k^{-1/2}$ is the generalized harmonic number; $M_c$ is number of examples that have pseudo label equal to class $c$; $\boldsymbol{\mu}_c$ is the mean of class $c$; $\boldsymbol{\mu}_u$ is the mean of class that example $\mathbf{x}_u$ belongs to; and $\nu_u = M_u[(\boldsymbol{\mu}_u^T\boldsymbol{\mu}_u)^2 - 4(\boldsymbol{\mu}_u^T\boldsymbol{\mu}_u)(\boldsymbol{\mu}_u^T f(\boldsymbol{x}_u)) + 2(f(\boldsymbol{x}_u)^T f(\boldsymbol{x}_u))(\boldsymbol{\mu}_u^T\boldsymbol{\mu}_u) + 2(\boldsymbol{\mu}_u^T f(\boldsymbol{x}_u))^2]^{1/2}$.*

In practice, we iteratively select the pseudo-labeled examples to augment the support set as shown in Algorithm 7. The classifier is firstly trained with the few-shot labeled data in the support set and the unlabeled using the joint loss in Eq.(6.10). After the pseudo-labels are generated by the classifier, We select a subset of faithful pseudo-labels to expand the support set. The expanded support set and the unlabeled set are then used to update the classifier based on Eq.(6.10) again. We iterate the above process to progressively enhance the classifier until the generated pseudo-labels becomes stable.

**Algorithm 7:** Semi-Supervised Few-Shot Learning.

**Require** Support data $D^S_{\mathcal{T}_i} = \{\mathbf{x}_n, y_n\}_{n=1}^{NK}$; Qeury data $D^Q_{\mathcal{T}_i}$; unlabeled data $D^U_{\mathcal{T}_i} = \{\mathbf{x}_u\}_{u=1}^{U}$; pretrained feature extractor $f_\theta(\cdot)$ ;
Initialize the augmented support set $(X_s, y_s) = \{\mathbf{x}_n, y_n\}_{n=1}^{NK}$ ;
Append a randomly initialized classifier $h_\phi(\cdot)$ to the feature extractor ;
**while** *pseudo-labels are not stabilized* **do**
    Train the classifier $h_\phi(\cdot)$ on $(X_s, y_s)$ and $D^U_{\mathcal{T}_i}$ using the joint loss in Eq.(6.10) ;
    Generate pseudo-labels for $\{\mathbf{x}_u\}_{u=1}^{U}$ and obtain $\{\mathbf{x}_u, \hat{y}_u\}_{u=1}^{U}$ ;
    Compute $d\psi$ for each pseudo-labeled example by Eq.(6.12) ;
    Select the most faithful subset $(X_{sub}, y_{sub})$ from $\{\mathbf{x}_u, \hat{y}_u\}_{u=1}^{U}$, and merge them into $(X_s, y_s)$. ;
Use the final augmented support set $(X_s, y_s)$ to retrain the classifier ;
Run inference on the query examples $D^Q_{\mathcal{T}_i}$ (i.e., the test set) ;
**Return** The trained classifier.

### 6.3.5 Experiments

We evaluate on four widely used few-shot benchmarks: *mini*-ImageNet [238], *tiered*-ImageNet [206], CUB [241], and CIFARFS [251]. Throughout the experiments, the hyper-parameters in the DM loss and IDA algorithm are kept fixed. We use Guassian kernel with bandwith $\sigma = 0.5$ for computing the empirical dependency metric in the DM loss. The weighting factor $\lambda$ in Eq.(6.10) is 0.01. When using IDA to select pseudo-labels, we select at most 5 samples per class at each iteration. When training the linear classifier, we use ADAM optimizer with $10^{-4}$ learning rate and run for 1000 iteration. We use WRN28-10 [284] as the CNN feature extractor, as it has been widely used by previous works. We train the feature extractor using the same procedure as [18]. The model is trained for 90 epochs with a batch-size of 128, using an initial learning rate of 0.1 which is divided by 10 at epoch 45 and 60. Standard data augmentations are employed, including random resized cropping, color jittering, and random horizontal flipping. All input images have a resolution of $84 \times 84$. During testing, we randomly sample 600 few-shot tasks from the unseen classes. The model needs to solve each testing task and we report the averaged testing accuracy, following the same setup as [164, 207, 117, 45, 18, 145, 206, 251].

**Semi-Supervised few-shot learning (SSFSL) results.** We follow the SSFSL setup in [251, 207, 151], where each testing task has an additional set of unlabeled examples from the same classes as the support set. The results are shown in Table 6.7. On *mini*-ImageNet and *tiered*-ImageNet datasets, we use 50 unlabeled examples per class in both 1-shot and 5-shot scenarios. On CIFARFS and CUB datasets, we use 80 and 30 unlabeled examples per class, respectively. Compared with previous methods that use more than 100 unlabeled examples per class, our method achieves noticeable higher accuracy across all datasets with accuracy improvements ranging 5%$\sim$13%. For example,

| Method | Type | *mini*-ImageNet | | CUB | | CIFARFS | | *tiered*-ImageNet | |
|---|---|---|---|---|---|---|---|---|---|
| | | 1-shot | 5-shot | 1-shot | 5-shot | 1-shot | 5-shot | 1-shot | 5-shot |
| DSN [214] | Inductive | 62.64 | 78.83 | - | - | 72.30 | 85.10 | 66.22 | 82.79 |
| FEAT [272] | | 66.78 | 82.05 | - | - | - | - | 70.80 | 84.79 |
| DeepEMD [288] | | 65.91 | 82.41 | 75.65 | 88.69 | - | - | 71.16 | 86.03 |
| TransFinetune† [45] | Transductive | 65.73 | 78.40 | - | - | 76.58 | 85.79 | 73.34 | 85.50 |
| LaplacianShot† [304] | | 74.86 | 84.13 | 80.96 | 88.68 | - | - | 80.18 | 87.56 |
| TIM† [18] | | 77.80 | 87.40 | 82.20 | 90.80 | - | - | 82.10 | 89.80 |
| SIB† [117] | | 70.00 | 79.20 | - | - | 80.00 | 85.30 | - | - |
| EPNet† [207] | | 70.74 | 84.34 | 87.75 | 94.03 | - | - | 78.50 | 88.36 |
| ICI+LR [251] | | 66.80 | 79.26 | 88.06 | 92.53 | 73.97 | 84.13 | 80.79 | 87.92 |
| BD-CSPN† [162] | | 70.31 | 81.89 | 87.45 | 91.74 | | | 78.74 | 86.92 |
| LST [145] | Semi-supervised | 70.10 | 78.70 | - | - | - | - | 77.70 | 85.20 |
| ICA+MSP† [151] | | 80.11 | 85.78 | - | - | - | - | 86.00 | 89.39 |
| EPNet† [207] | | 79.22 | 88.05 | - | - | - | - | 83.69 | 89.34 |
| ICI+LR [251] | | 71.41 | 81.12 | 91.11 | 92.98 | 78.07 | 84.76 | 85.44 | 89.12 |
| MixMatch [14] | | 63.02 | 82.24 | - | - | - | - | - | - |
| FixMatch [218] | | 53.71 | 76.02 | - | - | - | - | 69.00 | 85.93 |
| **Ours** | Transductive | 80.60 | 87.02 | 92.43 | 94.77 | 79.52 | 86.16 | 85.87 | 89.61 |
| | Semi-supervised | **83.34** | **88.17** | **93.51** | **95.44** | **82.16** | **87.26** | **87.12** | **90.54** |

Table 6.7: Comparison with previous state-of-the-art methods on four few-shot classification datasets. Our method consistently outperforms the competitors across four benchmarks.

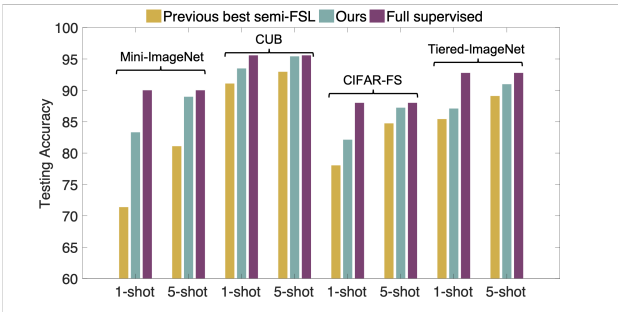



Figure 6.15: Compare our semi-supervised few-shot learning with full-shot supervised baseline. The previous results shown in the figure are from ICI [251].

compared to previous state-of-the-art [151], our method achieves 3.2% higher accuracy on 1-shot *mini*-ImageNet. Moreover, our method using only 5-shot labeled data can compete with the full-shot supervised learning on these benchmarks, as shown in Figure 6.15.

**Transductive few-shot learning (TFSL) results.** TFSL assumes that the query examples comes in an a bulk at inference stage, so that the model can utilize the query examples to augment the few-shot support set. We take the query set as the unlabeled set and applye our DM and IDA components. This transductive setup is used by all the other compared methods as well. As shown in Table 6.7, our method outperforms previous state-of-the-art TFSL approaches across, especially on 1-shot tasks where the labeled support data is extremely limited. For example, on 1-shot *mini*-ImageNet, our method outperforms [18] by 2.8% accuracy. Comparing our SSFSL results

| Method | Type | *mini*-ImageNet→ CUB | |
|---|---|---|---|
| | | 1-shot | 5-shot |
| MAML [59] | Inductive | - | 51.34 |
| ProtoNet [216] | Inductive | - | 62.02 |
| RelationNet [225] | Inductive | - | 57.71 |
| Finetuning [33] | Inductive | 48.56 | 65.57 |
| LaplacianShot § [117] | Transductive | 55.46 | 66.33 |
| **Ours** (Transductive) | Transductive | **55.79** | **71.01** |

Table 6.8: Results on the cross-domain few-shot learning problem.

| Method | Type | 10-way | | 20-way | |
|---|---|---|---|---|---|
| | | 1-shot | 5-shot | 1-shot | 5-shot |
| Baseline++ [33] | Inductive | 40.43 | 56.89 | 26.92 | 42.80 |
| LEO [209] | Inductive | 45.26 | 64.36 | 31.42 | 50.48 |
| MetaOpt [135] | Inductive | 44.83 | 64.49 | 31.50 | 51.25 |
| $S2M2_R$ [178] | Inductive | 50.40 | 70.93 | 36.50 | 58.36 |
| EPNet [207] | Transductive | 53.70 | 72.17 | 38.55 | 59.01 |
| TIM [18] | Transductive | 56.10 | 72.80 | 39.30 | 59.50 |
| BD-CSPN [162] | Transductive | 51.58 | 69.35 | 36.00 | 55.23 |
| **Ours** | Transductive | **60.05** | **75.93** | **41.47** | **61.91** |

Table 6.9: Results on 10-/20-way few-shot tasks from the *mini*-ImageNet.

versus the TFSL results, we see that the additional unlabeled examples in the semi-supervised setup can largely improve the final accuracy.

**Cross-domain few-shot learning results.** In real applications, the testing few-shot tasks may come from domains that are different from the training data. Previous work [33] showed that many meta-learning algorithms perform no better than the simplest transfer learning when there exists a domain-shift training and testing data. Therefore, we also evaluate the cross-domain few-shot learning scenario, where the feature extractor is pretrained on the *mini*-ImageNet dataset and we apply our method to solve testing few-shot tasks sampled from the CUB dataset. As shown in Table 6.8, our method outperforms previous meta-learning and transductive learning methods, suggesting that our method is applicable to more realistic few-shot learning problems.

### 6.3.6 Analyses

**Impact of number of classes.** We evaluate on more challenging 10-way and 20-way few-shot tasks, where each task contains data from 10 or 20 unseen classes and each class only has 1 or 5 labeled examples. As shown in Table 6.9, compared with previous meta-learning or transductive learning methods, our method achieves the highest accuracy. For example, our method outperforms previous best method [162] by 3% on 1-shot 20-way tasks.

| Loss | *mini*-ImageNet | | *tiered*-ImageNet | |
|---|---|---|---|---|
| | 1-shot | 5-shot | 1-shot | 5-shot |
| Cross-entropy (CE) only | 57.73 | 78.17 | 68.29 | 85.31 |
| CE + Entropy [45] | 65.73 | 78.40 | 73.34 | 85.50 |
| CE + Mutual Information [18] | 71.54 | 83.92 | 77.02 | 87.57 |
| **CE + Dependency (Ours)** | **75.80** | **85.26** | **82.42** | **89.12** |

Table 6.10: Ablation study on the effectiveness of proposed dependency maximization loss. Results are testing accuracy. All methods use WRN28-10 as the backbone.

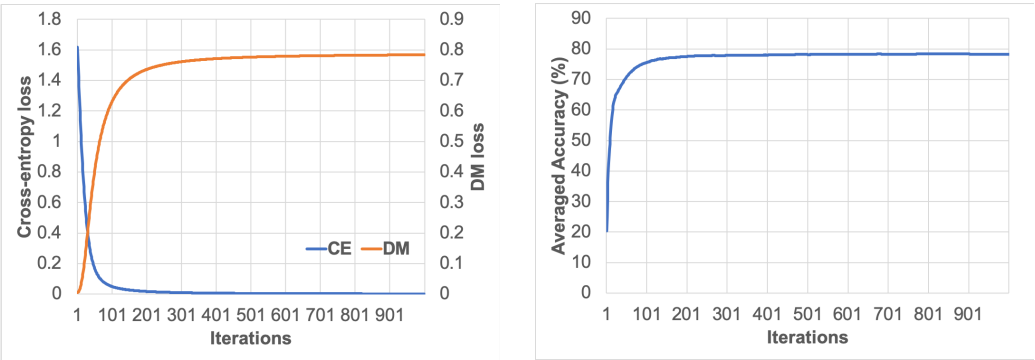



Figure 6.16: Convergence plot while we use Eq.(6.10) to train the classifer on 1-shot *mini*-ImageNet tasks. Left: cross-entropy loss and DM loss versus iterations. Right: accuracy of the classifier trained by the joint loss versus iterations.

**Impact of dependency maximization.** To validate the effectiveness of our proposed DM loss, we compare it with other unsupervised losses in Table 6.10. [45] proposes to minimize the conditional entropy of the label predictions on unlabeled data; [18] proposes to maximize the weighted mutual information on unlabeled data. We find that incorporating the DM loss consistently outperform other types of transductive learning on both *mini*-ImageNet and *tiered*-ImageNet. This suggests that maximizing the dependency between the latent features and the label predictions can effectively improve the generalization performance. Furthermore, Figure 6.16 shows the convergence plot when we use the joint loss in Eq.(6.10) to train the classifier on 1-shot *mini*-ImageNet tasks. We see that the dependency value increases monotonically and converge quickly. Meanwhile, the cross-entropy loss gets decreased monotonically and the accuracy consistently improves until stabilized.

**Compare IDA with other sample selection strategy.** To validate the effectiveness of IDA, we compare it with other pseudo-label selection strategy under the transductive and semi-supervised settings in Table 6.11. A naive strategy is to randomly select some pseudo-labeled examples to augment the support set. Another strategy is to select high-confidence examples, i.e. selecting the pseudo-labels whose largest class probability is above a certain threshold [145]. One can also leverage the nearest-neighbour strategy to select examples based on their distance to the centroid of each class

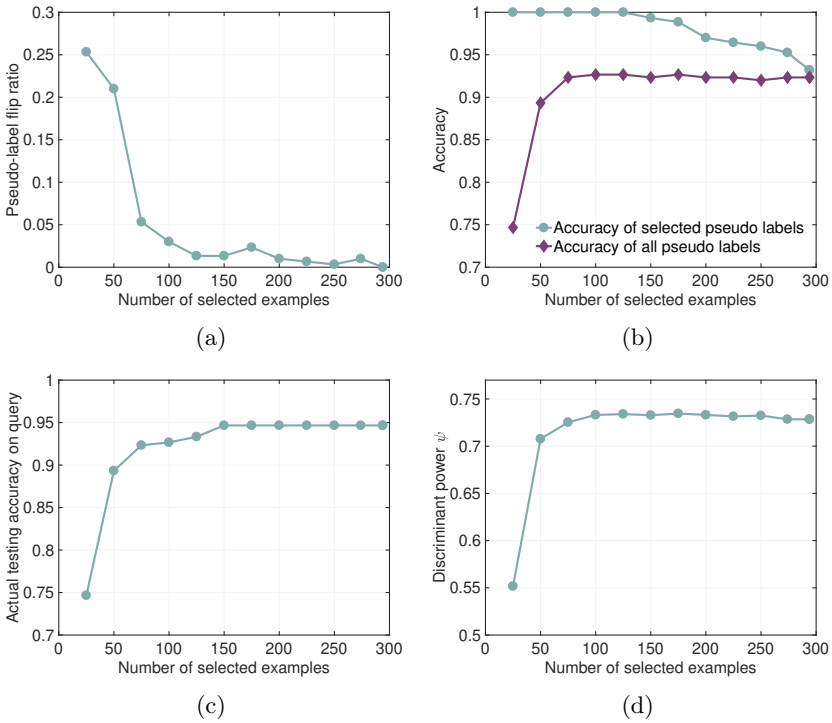



Figure 6.17: (a) The flip ratio of the pseudo-labels between two consecutive iterations. (b) The accuracy of the pseudo-labels w.r.t. the ground-truth labels. (c) The actual testing accuracy on the query set versus number of selected pseudo-labeled examples. (d) The discriminant power of the pseudo-labeled set over iterations.

in the feature space. The last strategy we compare to is ICI [251], which selects pseudo-labels based on a linear regression hypothesis. We set 15 unlabeled examples for each class and iteratively select 5 examples per class by different strategies to retrain the classifier on *mini*-ImageNet. As shown, IDA outperforms all other strategies. Specifically, IDA outperforms the ICI selection method, suggesting that our discriminant hypothesis can select more faithful pseudo-labeled examples than the linear regression hypothesis in [251]. We also provide visualizations for our iterative IDA algorithm in Figure 6.17. We have the following observations: (1) the pseudo labels gradually stabilize without label flipping at the end; (2) IDA can select more trustworthy pseudo labels as the accuracy of the selected pseudo labels is always higher than the accuracy of the overall pseudo labels; (3) the accuracy of the overall pseudo-labels gradually improves, indicating that some wrongly labeled examples in previous iterations can be corrected and re-considered to select in later iterations; (4) the actual testing accuracy on the query set consistently improves as we select more pseudo-labeled examples.

| Selection strategy | Transductive | | Semi-supervised | |
|---|---|---|---|---|
| | 1-shot | 5-shot | 1-shot | 5-shot |
| No selection | 56.06 | 75.43 | 56.06 | 75.43 |
| Random | 59.01 | 76.38 | 59.46 | 76.58 |
| Nearest Neighbour | 63.24 | 77.63 | 63.10 | 77.75 |
| Confidence | 63.29 | 77.92 | 63.57 | 77.71 |
| ICI [251] | 65.32 | 78.30 | 64.60 | 77.96 |
| **IDA (Ours)** | **67.17** | **80.00** | **67.36** | **80.18** |

Table 6.11: Comparing IDA to other metrics for selecting the pseudo-labeled examples for self-training. Results are testing accuracy on *mini*-ImageNet. All methods use ResNet-12 as the backbone.

## 6.4 Conclusion

Few-shot learning (very limited labels) has become a fundamental problem in modern AI research. In this chapter, we propose various few-shot learning approaches to address problems such as how to quickly adapt to model structures to different few-shot tasks and how to improve the model performance when we have very limited labeled data but also have some unlabeled data.

We have shown that explicit meta-learning is a useful preparation step on top of pretrained models to improve later finetuning. For fast architecture adaptation on few-shot tasks, we propose to meta-learn a task-aware architecture controller on the task distribution so that it generalize to new tasks when searching neural architectures, by learning to directly generate task-specific model structures from the task training data. The proposed lightweight controller for generating task-aware architectures set a new state-of-the-art on various few-shot language tasks such as Persona-Chat dialog personalization.

We have also shown that exploiting unlabeled data can greatly improve the few-shot performance. We introduce a simple semi-supervised learning approach. Specifically, we propose an unsupervised dependency maximization loss based on the Hilbert-Schmidt norm of the cross-covariance operator, which. maximizes the statistical dependency between the features of unlabeled data and their label predictions. This extra loss term help use the unlabeled data for training the model and shows better performance than previous unsupervised losses. We also employ the pseudo-labeling strategy, further propose an instance discriminant analysis to evaluate the quality of each pseudo-labeled example and only select the most faithful ones to training set augmentation. Our experiments demonstrate that the dependency loss and the instance discriminant analysis together advance the semi-supervised performance on multiple few-shot classification benchmarks.

# Chapter 7

# Self-Supervised Deep Learning

## 7.1 Prologue

Modern deep learning models have strong scaling property: increasing the model sizes has significant gains. However, the superb model qualities heavily depend on the excessively large labeled datasets for supervised learning. In order to reduce the reliance on large-scale labeled datasets and improve the transferablility of the learned representations, self-supervised learning methods are gaining popularity, which learn high quality transferable representations from unlabeled data. Unsupervised pretraining followed by in-domain or in-task supervised finetuning has been widely adopted to advance the performance in different modalities and have shown greater performance than large-scale supervised pretraining.

After the advent of the self-attention based transformer models, reconstruction based denoising or masked auto-encoders are revitalized for the self-supervised pretraining and already achieve great success on language tasks. However, different from the discrete tokens in text data containing highly semantic meanings, the raw images are redundant and only contain low-level information. Current practices on pretraining the models with a pixel-level recovery task tends to push the model to learn short-range dependencies and local image statistics, since the missing pixels can be recovered from neighborhood with little high-level understanding of the image contents. On the other hand, recent weakly supervised image pretraining methods also aim to learn representations without human labels by exploring the natural language supervision from text cations accompanying the images. Benefiting from the affluent semantic signals in the caption text, the model can have holistic understanding of the image contents beyond low-level statistics.

In this section, we propose masked image pretraining on language assisted representation, dubbed as MILAN. Instead of predicting raw pixels or low level features, our pretraining objective is to reconstruct the image features with substantial semantic signals that are obtained using the caption text supervision. Interestingly, the learned representations from our MILAN method even outperform the langauge-supervised targets. Moreover, to accommodate our reconstruction target, we propose a more efficient prompting decoder architecture and a semantic aware mask sampling mechanism, which further advance the transfer performance of the pretrained model. We evaluate our pretrained models on many different tasks and MILAN delivers higher accuracy than the previous works. On standard classification tasks on ImageNet, we use a vanilla ViT-L model with 307M parameters to achieve the best accuracy of 87.3%. Moreover, our pretrained models show promising transfer performance in downstream detection and segmentation tasks. Finally, our models demonstrate strong label efficiency in the semi-supervised learning task and robustness to adversarial/out-of-distribution examples.

## 7.2 Multi-modal self-supervised pretraining

### 7.2.1 Self-supervision

In recent years, we have seen a wide adoption of applying natural language processing (NLP) techniques in computer vision (CV) tasks. The vision transformer (ViT) model [54] applies the self-attention based transformer architecture to vision tasks and have achieved remarkable performance. However, training ViT models requires much larger labeled datasets to avoid overfitting, such as using ImageNet-22K [41] and JFT-300M [223] datasets. Explicitly labeling large image datasets is hardly affordable.

Self-supervised learning aims to learn transferable representations from unlabeled data by pretext tasks [21, 49, 235, 57, 142, 66, 286], and have become a popular paradigm to reduce the reliance on very large labeled dataset in both NLP and CV. Before the advent of vision transformer architectures, CNN-based self-supervised learning mainly adopt constrastive methods [23, 36, 31, 81, 231, 35, 68, 22, 259, 250, 261, 6] to learn augmentation invariance by enforcing similarity between different views from the same image while avoiding model collapse. The learned representations can achieve high linear separability and are commonly evaluated by linear probing. However, contrastive learning heavily depends on strong data augmentation and effective negative sampling. With the popularity of transformer architectures, recent self-supervised learning methods revitalize the use of denoising autoencoders [237, 195, 29, 290, 189] to pretrain the model with reconstruction based masked

prediction objectives, which have been shown to yield higher quality representations [11, 80, 262, 255] than the contrastive objective.

In reconstruction based self-supervised pretraining, the model receives incomplete input data with a large portion of patches or words removed and learns to reconstruct the missing contents based on the encoding results. It is first exemplified by BERT [44] in NLP. Acting like a masked autoencoder [237], BERT randomly masks some percentage of the input word tokens and learns to reconstruct the vocabularies of those masked tokens. Works in [11, 80, 262, 255] adopt similar techniques in CV to address the data-hungry issue of ViT models. A large percentage of the input image patches are randomly masked with the goal of reconstructing raw pixels [80], discrete visual vocabularies [11], or local image features [255].

### 7.2.2 Language-image pretraining

Learning visual representations from language supervision has also received substantial research insterests. Early work [60] embeds images and texts into a shared semantic space so that the model is able to recognize classes even without explicit labels. Other methods leverage the caption supervision to train the vision model by completing the image captioning task [43] or the masked language modeling task [211]. Recently, benefiting from contrastive training [282] and the scalability of modern backbones, CLIP [201] and ALIGN [123] learn strong visual representations on large-scale image-text datasets, advancing the transfer performance on the downstream vision tasks. Later works improve CLIP by introducing more auxiliary loss functions to assist the image-text contrastive loss, such as image self-supervision loss [183, 150], self-distillation loss [38], and token-wise max similarity [271].

### 7.2.3 Masked image pretraining on language assisted representation

The reconstruction based self-supervised learning can be extended in several directions. Unlike the masked word tokens in NLP which contain rich semantic information, the masked image patches only contain low-level pixel data. Several works [11, 52, 34] explore higher level visual concepts, e.g., discrete visual vocabularies, as the **reconstruction targets**. However, those methods still only retrieve semantic signals from raw image pixels, which by itself is a difficult task. In addition, the selection of the reconstruction targets heavily influences the **decoder design** in autoencoders, as the decoder serves to reconstruct the masked features with the guidance from the encoder's output representations. Full fledged transformer blocks are used in the decoder of MAE [80] to reconstruct masked input patches pixel by pixel, whereas light weight linear layer is adopted in the

decoder of MaskFeat [255] to reconstruct local features of the image. Therefore, if we were using reconstruction targets that preserves more semantics, a task tailored decoder architecture would be required. Furthermore, different **sampling strategies** (*e.g.,* grid, block, random) of the input image patches affect the final performance of masked image pretraining [80, 262]. Majority of prior arts [11, 52, 80, 262, 255, 8, 34] sample the masked patches uniformly at random since it is unbiased and can guarantee coverage. However, it is indifferent to more discriminative image patches and unimportant ones, thus may suffer from slow training convergence [126]. In this work, we analyze three highly correlated aspects in masked autoencoders: the reconstruction target, the decoder design, and the mask sampling strategy. We propose a new approach called **MILAN**, which performs the masked image pretraining on language assisted representation. In specific:

(1) We recognize the limitation of extracting semantic signals form raw image pixels alone. But such signals are readily available in the captions accompanying the images. Recent works (e.g., CLIP [201]) explore the use of caption supervision to learn image representations using the abundant image-text pairs on the Internet. The output image features from those models implicitly contain semantic information that facilitates the interpretation of the image contents. In this work, we take the image features coming out of the CLIP image encoder as the reconstruction targets for the masked image pretraining, which benefits from natural language supervision and encourages the model to learn high level visual concepts. More interestingly, we will show that the quality of the representation improves on the targets after masked image pretraining.

(2) We realize the tight coupling between the decoder architecture and the reconstruction targets. We design an efficient prompting decoder suitable for reconstruction targets that are latent representations containing affluent semantic signals. It freezes the encoder's output representations of the unmasked patches and uses them as "fixed prompts" to reconstruct the features of the masked patches. Prompting decoder achieves higher accuracy and reduces computational cost simultaneously.

(3) Different image patch sampling strategies impact the pretraining efficiency. Since our reconstruction targets provide global structure information of the images, we propose a semantic aware mask sampling mechanism to discriminate semantically important image patches from the insignificant background patches, which improves representation quality and pretraining efficiency.

(4) Combining the three aspects leads to our MILAN framework, as shown in Figure 7.1. Our ViT-B/16 (86M parameters) and ViT-L/16 (307M parameters) models pretrained and finetuned on ImageNet-1K dataset achieve 86.4% and 87.3% top-1 accuracy, respectively. Moreover, MILAN significantly boosts the linear probing (only finetune the linear classifier layer) accuracy compared to reconstruction based and language-image based pretraining methods, and achieves state-of-the-art

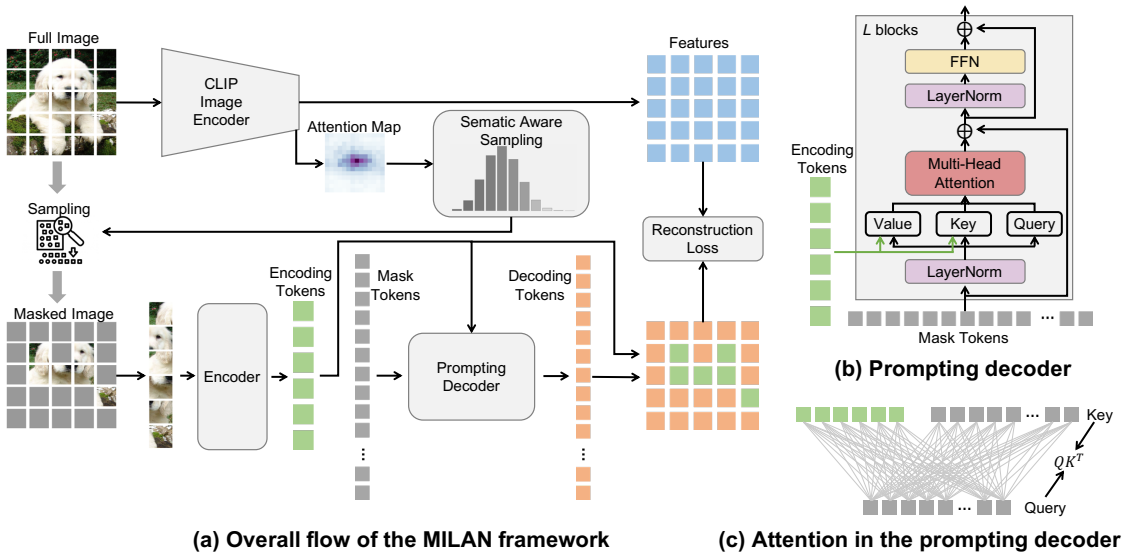



Figure 7.1: (a) The overall flow of MILAN. The masked autoencoder uses the outputs of the CLIP image encoder as the reconstruction target. An efficient prompting decoder freezes the features of the encoding tokens and only updates the mask tokens. A semantic aware sampling is used to guide the selection of the unmasked image patches. The reconstruction loss is computed on the representation features of both masked and unmasked patches. (b) A detailed diagram of the prompting decoder. (c) The attention computation in the prompting decoder.

performance on the downstream object detection, instance segmentation, and semantic segmentation tasks.

## 7.3 Method

### 7.3.1 Overview

The overall flow of MILAN is illustrated in Figure 7.1(a). We use a masked autoencoder architecture similar to MAE [80]. The encoder transforms the unmasked patches into latent representations. The decoder reconstructs the representations of the masked patches assisted by the features of the unmasked patches. We use the latent features that the CLIP image encoder produces from the full image as the reconstruction targets, which are contextualized representations based on the global structure of the image and the associated caption information. The attention map is extracted from the last self-attention layer of the CLIP image encoder and is used to construct a semantic aware sampling distribution to select the unmasked patches. The sampled patches are sent into the encoder and mapped to the latent feature space. We design a prompting decoder that freezes the encoder's output when hallucinating the features of the masked patches from mask tokens. As shown in Figure 7.1(b), the query of the attention block in the prompting decoder only contains the features

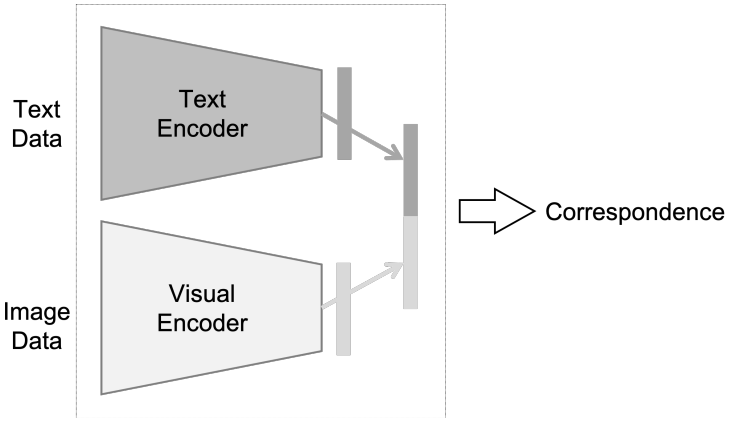



Figure 7.2: Illustration of the cross-modal training on image-text data by CLIP's contrastive objective.

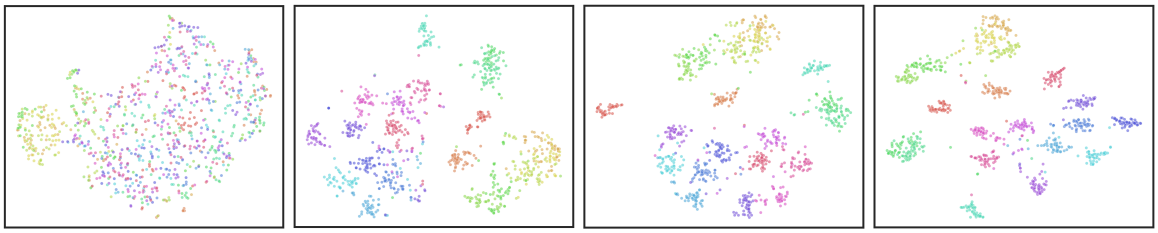
of mask tokens. The key and value matrices comprise both the encoder's outputs and the features of mask tokens. The masked patches' features from the prompting decoder and the unmasked patches' features from the encoder are combined at the end. We re-order the combined full set of features to align with the targets, and compute the reconstruction loss. After pretraining, the encoder model is what we want at the end, which learns transferable representations for later tasks.

Although MILAN uses a masked auto-encoder architecture, it differs from MAE [80] in: 1) the targets we predict are latent representations obtained with language guidance, whereas MAE reconstructs raw pixels; 2) mask sampling in MILAN is more adapted to patches' discriminativeness in contrast to MAE's uniform sampling; 3) our prompting decoder does not update the encoder's output and thus is more efficient.

### 7.3.2 Reconstruction target: language assisted representation

The reconstruction target is a crucial component in masked image pretraining. It influences the semantics of the learned latent representations. Language naturally contains rich semantics, while such information is more difficult to extract from image pixels directly. Thus, an image-text pair provides more meaningful learning signals than an image alone. In practice, texts accompanying images can be easily obtained at scale in the form of image captions. Such large unlabeled image-text datasets foster a series of weakly-supervised image pretraining methods [201, 183, 150, 215] with caption supervision. It is expected that the visual representations learned with language guidance can provide affluent semantic information. Therefore, we take the language assisted representations as the reconstruction target in our masked image pretraining framework.

We primarily use the pretrained CLIP [201] model to generate the reconstruction targets. CLIP is trained on a dataset containing images and free-form text captions using the InfoNCE loss [235].



(a) MAE pretrained (b) CLIP image encoder (c) MILAN pretrained (d) MILAN finetuned

Figure 7.3: t-SNE visualization of the learned features from ViT-B/16 obtained by different pretraining methods. We plot the features before the final linear head. We use images of randomly sampled 20 classes in ImageNet-1K validation split.

The model has an image encoder and a text encoder, both using the transformer architecture. The encoded image and text features are projected to the same dimension and normalized. Given a batch of image-text pairs, CLIP trains the image encoder and the text encoder by maximizing the feature cosine similarity for the matching image-text pairs in the batch while minimizing the cosine similarity for all other non-matching pairs, as shown in Figure 7.2. Though without labeled data, the image features are trained to match the paired text features and distill the rich semantics embedded in the text features. This is illustrated in Figure 7.3, where the last layer features produced by MAE's pretrained model, CLIP's image encoder, and our pretrained model are visualized by t-SNE plots. As shown, the representations from CLIP's image encoder for each category tend to be grouped together while the pretrained model from MAE cannot distinguish the visual concepts in different categories. Since MILAN uses the image features from CLIP as the target and trains the model with a more challenging masked prediction objective. The learned representations are better clustered for different categories than both MAE and CLIP approaches.

The pretraining objective of MILAN is formally described as follows. Let $f_\theta$ denote the CLIP image encoder, whose weights $\theta$ are frozen. The masked autoencoder under training comprises encoder $g_\xi$ with weights $\xi$ and decoder $h_\nu$ with weights $\nu$. From a given full image $\boldsymbol{x}$, the CLIP image encoder outputs the target features $\{\boldsymbol{t}_j\}_{j=1}^N = f_\theta(\boldsymbol{x})$ where $N$ is the number of image patches. We mask a high portion of patches in $\boldsymbol{x}$, and obtain a masked image $\tilde{\boldsymbol{x}}$. The masked autoencoder outputs the reconstructed representations $\{\boldsymbol{p}_j\}_{j=1}^N = (h_\nu \circ g_\xi)(\tilde{\boldsymbol{x}})$. We apply $\ell_2$-normalization to both the targets and reconstructions: $\bar{\boldsymbol{t}}_j = \boldsymbol{t}_j / \|\boldsymbol{t}_j\|_2$, $\bar{\boldsymbol{p}}_j = \boldsymbol{p}_j / \|\boldsymbol{p}_j\|_2$ for $j \in [N]$. Finally, we define the following mean squared error between the normalized target features and reconstructed representations:

$$\mathcal{L}_{\xi,\nu} = (1/N) \cdot \sum_{j=1}^{N} \|\bar{\boldsymbol{p}}_j - \bar{\boldsymbol{t}}_j\|_2^2. \tag{7.1}$$

MILAN learns the weights $\xi, \nu$ of the masked autoencoder by minimizing objective (7.1). Note that the reconstruction loss is computed on the features of both masked and unmasked patches.

### 7.3.3 Decoder design: prompting decoder

Since the decoder is discarded after pretraining, the encoder needs to learn rich semantic information in the latent representations of the unmasked patches from the reconstruction target. To achieve so, the functional roles of the encoder and decoder need to be clearly segregated. All representation learning for the unmasked patches is completed in the encoder, while the decoder is only for predicting the target features of the masked patches. However, in some previous works [262, 11], the decoder is as simple as one linear layer, which may be insufficient to reconstruct the masked representations, and portions of the encoder may serve as the decoder. MAE [80] uses a deep decoder that not only updates the mask tokens but also enhances the features of the unmasked patches. Because our method reconstructs language guided latent representations instead of raw pixels, the encoder's outputs should only provide clues to the decoder to complete the missing patches' features without being updated in the decoder. Otherwise, the representation quality from the encoder becomes sub-optimal.

In MILAN, we propose a prompting decoder shown in Figure 7.1(b), where the representations of the unmasked patches from the encoder are frozen, serving as "fixed prompts". They are appended to the keys and values in each attention module of the prompting decoder's transformer block while the queries only contain the features of mask tokens. In specific, the mutli-head attention (MHA) module in Figure 7.1(b) performs the following operations:

$$\begin{aligned}\mathrm{MHA}(X, Z) &= \mathrm{Attn}\big(XW_q,\ \mathrm{concat}(Z, XW_k),\ \mathrm{concat}(Z, XW_v)\big), \\ &= \sum_{h=1}^{H} \mathrm{softmax}\Big(\frac{XW_q^h \mathrm{concat}(Z, XW_k^h)^T}{\sqrt{d^h}}\Big)\mathrm{concat}(Z, XW_v^h)W_o^h. \end{aligned} \tag{7.2}$$

In Eq.(7.2), $X$ and $Z$ are features of mask tokens and encoder's outputs, respectively. "Attn" is short for the softmax attention operation. "concat" means concatenating along the sequence dimension. $H$ is the number of heads. $W_q^h, W_k^h, W_v^h, W_o^h$ represent the query, key, value and output projection weight matrices in each head. $d^h$ is the embedding dimension in each head. As shown in Figure 7.1(c), our prompting decoder only computes the self-attention among the features of mask tokens and the cross-attention between the encoder's output and the features of mask tokens. Moreover, the FFN modules in prompting decoder only compute on the features of mask tokens.

Using the default 75% masking ratio, our prompting decoder reduces the computation cost by 20% compared to MAE [80]. More importantly, prompting decoder improves the finetuning accuracy significantly.

### 7.3.4 Masking strategy: semantic aware sampling

To make the masked image pretraining a meaningful pretext task for representation learning, prior works mask a very high portion of input image patches uniformly at random. With this aggressive masking strategy, it is possible that the remaining few visible patches only contain background information, which may not provide the important clues needed to reconstruct the foreground objects buried in the piles of masked patches, obstructing the model to learn transferable representations. This becomes a more severe problem in our framework, because the latent representations from the encoder are frozen in the decoding process. To ensure the representation quality of the pretrained model, previous methods [80, 8, 255] usually require very long pretraining epochs.

To improve the pretraining efficiency, we propose a semantic aware mask sampling strategy that can make more rational decisions on which patches to mask when using a very high masking ratio. The idea is that the few visible patches fed into the encoder cover important image regions with high probabilities, so that the latent representations from the encoder provide sufficient clues to the decoder to predict the representations of the masked patches.

To discriminate the semantically important patches from the unimportant ones, we use the attention weights from the last self-attention layer in the CLIP image encoder, which takes the patches of the entire image and an extra class token as input. Denote the input features to the last self-attention layer of the CLIP image encoder by $[\mathbf{z}_{\text{class}}; \mathbf{z}_1; ...; \mathbf{z}_N] \in \mathbb{R}^{(N+1)\times d}$, where $N$ is the sequence length and $d$ is the embedding dimension. The interaction between the class token and other features is given by the following attention mechanism:

$$\mathbf{s}_{\text{class}} = \text{softmax}(\mathbf{q}_{\text{class}}\mathbf{K}^T/\sqrt{d}), \tag{7.3}$$

where $\mathbf{s}_{\text{class}} \in \mathbb{R}^{1\times(1+N)}$ is the attention vector of the class token. $\mathbf{q}_{\text{class}} = \mathbf{z}_{\text{class}}W_q$ is the query vector of the class token, and $\mathbf{K} = [\mathbf{z}_{\text{class}}; \mathbf{z}_1; ...; \mathbf{z}_N]W_k$ is the key matrix, where the query and key projection matrices have dimensions $W_q, W_k \in \mathbb{R}^{d\times d}$. For simplicity, we show a single-head attention in Eq.(7.3). When multiple attention heads are present, $\mathbf{s}_{\text{class}}$ is obtained by averaging over all the heads. Because the class token from the last layer of the CLIP image encoder is used to align with the text embedding from the text encoder, $\mathbf{s}_{\text{class}}$ reflects how much information one image patch contributes to the output features of the CLIP image encoder. The magnitude of the $i$-th element in $\mathbf{s}_{\text{class}}$, denoted by $\mathbf{s}_{\text{class}}(i)$, indicates whether the $i$-th patch is semantically important or not. The attention vector $\mathbf{s}_{\text{class}}$ provides us the premise to design a non-uniform sampling distribution. Due to the softmax operation, we can regard $\mathbf{s}_{\text{class}}(i)$ as the probability of leaving the $i$-th patch unmasked in the input image. Let $r$ represent the masking ratio. The indices of the unmasked patches are obtained

| Tasks | Models | Datasets | Eval metrics |
|---|---|---|---|
| Image classification | ViT-B/-L | ImageNet (1.28M train data, 1K categories) | Top-1 accuracy |
| Object detection | Mask R-CNN | COCO (118K train data, 80 categories) | Mean average precision |
| Semantic segmentation | UperNet | ADE20K (25K train data, 3K categories, 193K parts) | Mean intersection over union |
| Adversarial examples | ViT-B | ImageNet-Adversarial (8K natural adversarial images in 200 classes) | Top-1 accuracy |
| Distribution shift | ViT-B | ImageNet-Rendition/-Sketch (30K rendition images from 200 classes/ 50K sketch images from 1K classes) | Top-1 accuracy |

Table 7.1: Summary of the tasks, models, and datasets on which we have evaluated MILAN.

by sampling a Multinomial distribution according to the probabilities $\{\mathbf{s}_{\text{class}}(0), ..., \mathbf{s}_{\text{class}}(N)\}$ for $\lceil (1-r)N \rceil$ trials without replacement.

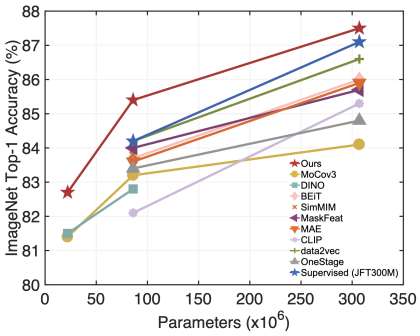
## 7.4 Experiments

We pretrain the ViT-B/16 and ViT-L/16 models using MILAN method on ImageNet-1K dataset for 400 epochs using PyTorch framework on A100 machines. We report the finetuning, linear probing, and semi-supervised learning accuracy for the image classification tasks on ImageNet-1K dataset in Section 7.4.1,Section 7.4.2,Section 7.4.3. We also evaluate the transferability of the pretrained models on object detection, instance segmentaton, and semantic segmentation tasks in Section 7.4.4. Apart from these standard tasks, we further demonstrate the adversarial/domain robustness of the MILAN models in Section 7.4.5. A summary of the tasks, models, and datasets are provided in Table 7.1.

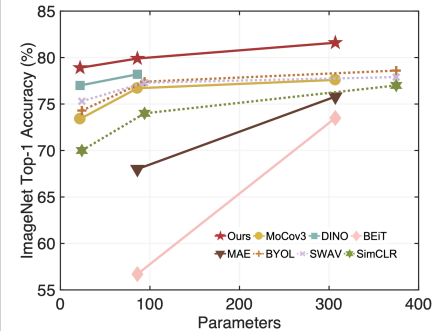
### 7.4.1 Finetuning results on ImageNet-1K.

Table 7.2 compares the finetuning accuracy on ImageNet-1K dataset using MILAN and previous works on ViT models. We pretrain and finetune the ViT models using ImageNet-1K dataset only. Since the CLIP model is pretrained on YFCC15M, we also list it in the training data entry for MILAN. However, we only use CLIP image encoder's output features as the reconstruction target for our masked autoencoder. Even though the supervised ViT models are pretrained on the large JFT300M dataset with explicit human labels, MILAN still outperforms them by a clear margin, improving the accuracy on ViT-B/16 by +2.2% at input resolution $384 \times 384$.

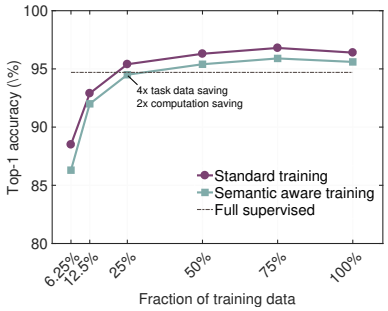
The self-supervised pretraining methods are divided by using contrastive or reconstruction based objectives. We also compare with large-scale (3.6B training samples) weakly-supervised pretraining



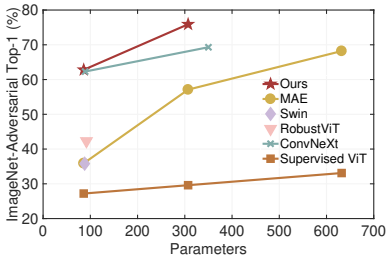

(a) Finetuning top-1 on ImageNet.




(b) Linear probing top-1 on ImageNet




(c) Few-shot learning of MILAN models. models.




(d) Adversarial robustness of MILAN models.

Figure 7.4: Results of cross-modal reconstruction based learning (MILAN). We compare MILAN with previous SOTA self-supervised learning methods in terms of finetuning accuracy, linear probing accuracy, and adversarial robustness on ImageNet. Moreover, MILAN models can improve data efficiency when transferring to downstream fine-grained classification dataset.

using hashtag supervision [215]. MILAN produces higher accuracy than all listed prior arts. Compared with MAE [80], MILAN improves the accuracy by +1.8% for ViT-B/16 and +0.8% for ViT-L/16.

The CLIP [201] and SLIP [183] models learn visual representations with language supervision on a large image-text dataset. Finetuning their pretrained image encoder does not lead to competitive accuracy. However, when using the image features from the pretrained CLIP model as the reconstruction target to train a mask autoencoder, MILAN improves the accuracy by 2%∼5%.

### 7.4.2 Linear probing performance.

Instead of finetuning the entire model, we can perform linear probing by appending a linear classifier after the final layer of the pretrained model and only finetune the linear classifier, which greatly save the training cost and memory for transfering the pretrained model. Table 7.3 compares the top-1 accuracy on ImageNet-1K dataset using various pretraining methods. As the results show, MILAN beats the accuracy of reconstruction based and language-image pretraining based approaches by a large margin. It matches (on ViT-B/16) and even outperforms (on ViT-L/16) the contrastive

| Method | Training data | Resolution | ViT-B/16 | | ViT-L/16 | |
|---|---|---|---|---|---|---|
| | | | Epochs | Top-1 (%) | Epochs | Top-1 (%) |
| Supervised [234] | IN1K | 224 | - | 83.8 (+1.6) | - | 84.9 (+1.8) |
| ***contrastive or clustering based*** | | | | | | |
| MoCov3 [36] | IN1K | 224 | 300 | 83.2 (+2.2) | 300 | 84.1 (+2.6) |
| DINO [23] | IN1K | 224 | 400 | 82.8 (+2.6) | - | - |
| iBOT [300] | IN22K+IN1K | 224 | 320 | 84.4 (+1.0) | 200 | 86.3 (+0.4) |
| ***reconstruction based*** | | | | | | |
| BEiT [11] | DALLE250M+IN22K+IN1K | 224 | 150 | 83.7 (+1.7) | 150 | 86.0 (+0.7) |
| mc-BEiT [144] | OpenImages9M+IN1K | 224 | 800 | 84.1 (+1.3) | 800 | 85.6 (+1.1) |
| PeCo [52] | IN1K | 224 | 800 | 84.5 (+0.9) | 800 | 86.5 (+0.2) |
| SimMIM [262] | IN1K | 224 | 800 | 83.8 (+1.6) | - | - |
| MaskFeat [255] | IN1K | 224 | 1600 | 84.0 (+1.4) | 1600 | 85.7 (+1.0) |
| data2vec [8] | IN1K | 224 | 800 | 84.2 (+1.2) | 1600 | 86.6 (+0.1) |
| CAE [34] | IN1K | 224 | 800 | 83.6 (+1.8) | - | - |
| MAE [80] | IN1K | 224 | 1600 | 83.6 (+1.8) | 1600 | 85.9 (+0.8) |
| ***language-image pretraining based*** | | | | | | |
| CLIP [201] | OpenAI400M+IN1K | 224 | - | 82.1 (+3.3) | - | 85.3 (+1.4) |
| SLIP [183] | YFCC15M+IN1K | 224 | 100 | 83.4 (+2.0) | 100 | 84.8 (+1.9) |
| **MILAN** | OpenAI400M+IN1K | 224 | 400 | **85.4** | 400 | **86.7** |
| Supervised [54] | JFT300M+IN1K | 384 | 90 | 84.2 (+2.2) | 90 | 87.1 (+0.2) |
| BEiT [11] | DALLE250M+IN1K | 384 | 800 | 84.6 (+1.8) | 800 | 86.3 (+1.0) |
| SWAG [215] | IG3.6B+IN1K | 384 | 2 | 85.3 (+1.1) | - | - |
| **MILAN** | OpenAI400M+IN1K | 384 | 400 | **86.4** | 400 | **87.3** |

Table 7.2: Comparison of the **finetuning** top-1 accuracy on ImageNet-1K dataset. All models are pretrained with 224×224 input resolution. We compare finetuning with both 224×224 and 384×384 resolutions. "Epochs" refer to the pretraining epochs. "-": not reported by the original paper.

based methods, which learn more linearly separable representations by instance discrimination [36] or clustering [23], and are known to be more effective in linear probing [31, 81, 35, 68].

### 7.4.3 Semi-supervised learning on ImageNet-1K.

After the unspervised pretraining, we compare finetuning results on a fraction of the labeled examples on ImageNet. Following [32], we sample 10% of the labeled examples from the ImageNet training sett in a class-balanced way (roughly 128 images per class), and finetune the whole model on these labeled data only. Table 7.4 shows the comparisons of our results against previous state-of-the-art semi-supervised learning methods. Again, our MILAN models significantly improves over previous results. For example, compared with SimCLR v2 [32], we advance the accuracy by 1.8% while requiring only much less model parameters, suggesting that our pretrained models have higher label efficiency.

### 7.4.4 Downstream tasks

**Object detection and instance segmentation on COCO.** To verify the transferability of MILAN, we evaluate it on COCO dataset [159] for object detection and instance segmentation. The

| Method | ViT-B/16 | | ViT-L/16 | |
|---|---|---|---|---|
| | Epochs | Top-1 (%) | Epochs | Top-1 (%) |
| ***contrastive or clustering based*** | | | | |
| MoCov3 [36] | 300 | 76.7 (+3.2) | 300 | 77.6 (+4.0) |
| DINO [23] | 400 | 78.2 (+1.7) | - | - |
| iBoT [300] | 1600 | **79.5** (+0.4) | 1000 | 81.0 (+0.6) |
| ***reconstruction based*** | | | | |
| BEiT [11] | 800 | 56.7 (+23.2) | 800 | 73.5 (+8.1) |
| SimMIM [262] | 800 | 56.7 (+23.2) | - | - |
| MaskFeat [255] | - | - | 1600 | 67.7 (+13.9) |
| CAE [34] | 800 | 68.3 (+11.6) | - | - |
| MAE [80] | 1600 | 68.0 (+11.9) | 1600 | 75.8 (+5.8) |
| ***language-image pretraining based*** | | | | |
| CLIP [201] | 25 | 66.5 (+13.4) | 25 | 70.5 (+11.1) |
| SLIP [183] | 25 | 72.1 (+7.8) | 25 | 76.0 (+5.6) |
| **MILAN** | 400 | **79.9** | 400 | **81.6** |

Table 7.3: Comparison of the **linear probing** top-1 accuracy on ImageNet-1K dataset. "Epochs" refer to the pretraining epochs of various methods. All methods adopt 224×224 input resolution in pretraining and linear probing.

| Method | Architecture | Parameters | Top-1 (%) |
|---|---|---|---|
| CPC v2 [92] | ResNet-161 | 305M | 73.1 |
| SimCLR [31] | ResNet-50 (4×) | 375M | 74.4 |
| SimCLR v2 [32] | SKNet-152 (3×) | 795M | 80.1 |
| BYOL [68] | ResNet-50 (4×) | 375M | 75.7 |
| MILAN | ViT-L/16 | 307M | **81.9** |

Table 7.4: Accuracy on ImageNet by finetuning the pretrained models with few labels, using only **10%** of the labeled examples.

pretrained ViT backbone is adapted to FPN [156] in the Mask R-CNN framework [82], which is finetuned end-to-end on COCO training set to produce the bounding boxes (evaluated by box AP) and the instance masks (evaluated by mask AP) simultaneously. The results are shown in Table 7.5. Compared to supervised pretraining, our MILAN performs better in both tasks, achieving 4.7/2.6 and 5.7/3.6 points improvements by $AP^{box}/AP^{mask}$ for ViT-B and ViT-L, respectively. Compared with the previous best result from MAE, which is obtained by 1600-epoch pretraining, MILAN advances $AP^{box}/AP^{mask}$ by 2.3/0.6 and 1.7/0.3 points for ViT-B and ViT-L but only pretrains for 400 epochs.

**Semantic segmentation on ADE20K.** We also transfer our pretrained model to semantic segmentation task on the ADE20K dataset [299]. The encoder ViT model pretrained on ImageNet-1K dataset serves as the backbone of UperNet [260], and is finetuned together with the segmentation layers. In Table 7.5, we report the mean intersection over union (mIoU) averaged over all semantic categories. Our method significantly improves the transferring results of ViT-B to 52.7, surpassing superviseed baseline and MAE by 5.3 and 4.6 points, respectively.

| Method | Epochs | ViT-B | | | ViT-L | | |
|---|---|---|---|---|---|---|---|
| | | Object detection | Instance segmentation | Semantic segmentation | Object detection | Instance segmentation | Semantic segmentation |
| Supervised [82, 260] | - | 47.9 (+4.7) | 42.9 (+2.6) | 47.4 (+5.3) | 49.3 (+5.7) | 43.9 (+3.6) | 49.9 (+5.4) |
| MoCov3 [36] | 300 | 47.9 (+4.7) | 42.7 (+2.8) | 47.3 (+5.4) | 49.3 (+5.7) | 44.0 (+3.5) | 49.1 (+6.2) |
| DINO [23] | 300 | 46.8 (+5.8) | 41.5 (+4.0) | 47.2 (+5.5) | - | - | - |
| BEiT [11] | 300 | 42.6 (+10.) | 38.8 (+6.7) | 45.7 (+7.0) | 53.3 (+1.7) | 47.1 (+0.4) | 53.3 (+2.0) |
| PeCo [52] | 300 | 43.9 (+8.7) | 39.8 (+5.7) | 46.7 (+6.0) | - | - | - |
| SplitMask [56] | 300 | 46.8 (+5.8) | 42.1 (+3.4) | 45.7 (+7.0) | - | - | - |
| CAE [34] | 800 | 49.2 (+3.4) | 43.3 (+2.2) | 48.8 (+3.9) | - | - | - |
| MAE [80] | 1600 | 50.3 (+2.3) | 44.9 (+0.6) | 48.1 (+4.6) | 53.3 (+1.7) | 47.2 (+0.3) | 53.6 (+1.7) |
| **MILAN** | 400 | **52.6** | **45.5** | **52.7** | **55.0** | **47.5** | **55.3** |

Table 7.5: Results of object detection ($AP_{box}$) and instance segmentation ($AP_{mask}$) are obtained by using Mask R-CNN on COCO dataset with an input resolution of 1024×1024. Semantic segmentation (mIoU) results are obtained by using UperNet on ADE20K with an input resolution of 512×512. All methods use ViTs pretrained on ImageNet-1K dataset as the backbone. "Epochs" refer to the pretraining epochs.

| Method | Parameters | ImageNet-Adversarial | ImageNet-Rendition | ImageNet-Sketch |
|---|---|---|---|---|
| ViT [54] | 86M | 27.2 | 49.4 | 35.6 |
| Swin [167] | 88M | 35.8 | 46.6 | 32.4 |
| RobustViT [179] | 92M | 28.5 | 48.7 | 36.0 |
| ConvNeXt [168] | 89M | 36.7 | 51.3 | 38.2 |
| MAE [80] | 86M | 35.9 | 48.3 | 34.5 |
| MILAN | 86M | **62.2** | **64.1** | **46.3** |

Table 7.6: Comparison of robustness to adversarial examples and distribution shifts on ImageNet datasets.

### 7.4.5 Robustness evaluation

In Table 7.4.5, we evaluate the robustness of our models to adversarial examples and distribution shifts. For adversarial robustness, we test the natural adversarial examples from ImageNet-Adversarial dataset [94]. For out-of-distribution robustness, we test images from ImageNet-Rendition [93] and ImageNet-Sketch [245] datasets, which contain images with naturally occurring distribution changes. After pretraining, we only finetune the models on the original ImageNet and directly run inference on these different validation sets, without any specialized fine-tuning. Our results outperform previous best results from either large-scale pretraining or specialized/advanced architectures by clear margins. For example, on ImageNet-Adversarial, our MILAN is 25.5% better than ConvNeXt under same parameters.

## 7.5 Analyses

We investigate the effectiveness of the different components in MILAN through an ablation study in Table 7.7. The results are based on pretraining a ViT-B/16 model on ImageNet-1K dataset for 400 epochs, followed by a 100-epoch finetuning. We tune the optimal learning rate for each entry, including the MAE baseline. We make the following findings in our study.

| | Method | Epochs | Top-1 (%) |
|---|---|---|---|
| #1 | Baseline (MAE) | 400 (1600) | 83.0 (83.6) |
| #2 | + CLIP target | 400 | 83.9 |
| #3 | + Prompting decoder | 400 | 83.0 |
| #4 | + Semantic aware sampling | 400 | 83.3 |
| #5 | + Prompting decoder + Semantic aware sampling | 400 | 83.3 |
| #6 | + CLIP target + Semantic aware sampling | 400 | 84.1 |
| #7 | + CLIP target + Prompting decoder | 400 | 85.1 |
| #8 | + CLIP target + Prompting decoder + Semantic aware sampling | 400 (1600) | **85.4** (85.6) |
| #9 | + SLIP target + Prompting decoder + Semantic aware sampling | 400 | 84.4 |

Table 7.7: Ablation study of different components in MILAN. All results are obtained by pretraining and finetuning ViT-B/16 model on ImageNet-1K dataset at 224×224 resolution.

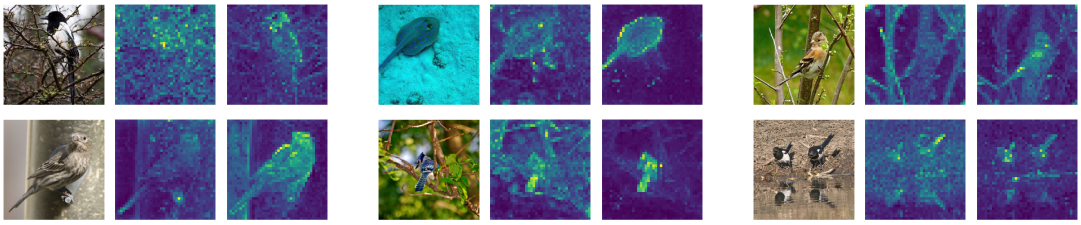


Figure 7.5: Visualization of original images (left), and the attention features extracted from the last self-attention layer of ViT-B/16 model pretrained by MAE (middle) and MILAN (right).

Figure 7.6: Visualization of the original images (left), masked images by the semantic aware sampling strategy with 75% masking ratio (middle), and the reconstruction loss patch-by-patch (right). For the plots of reconstruction loss, darker green colors indicate higher loss values. As shown, both unmasked patches and masked foreground patches have lower losses.

(1) By changing the reconstruction target from raw pixels to language guided representations provided by CLIP, the top-1 accuracy is improved by 0.9% (#2 *vs.* #1 in Table 7.7). We hypothesize that our target provides more semantic learning signals for pretraining, and encourages the model to get a good grasp of the visual contents instead of the low level statistics, illustrated in Figure 7.5.

(2) With the CLIP target, replacing the original decoder in MAE by our prompting decoder in Figure 7.1(b) further improves the accuracy by 1.2% (#7 *vs.* #2 in Table 7.7). However, prompting decoder does not increase accuracy when the raw pixels are reconstructed, as the MAE model does (#3 *vs.* #1 in Table 7.7). This big difference can be explained by the different pretraining objectives. When the targets are in the latent space, the encoder's output of the unmasked patches' representations should be able to align with the targets without requiring further updates in the decoder. Therefore, in our case of using the CLIP target, the proposed efficient decoder brings in significant accuracy improvements. While in MAE, the encoder's output requires transformations in the decoder to map from the latent space back to raw pixels. In short, the decoder design is

heavily correlated with the reconstruction target. We realize the critical coupling effect of these two components and propose mutually beneficial design choices to boost the accuracy.

(3) The proposed semantic aware mask sampling strategy is generally beneficial regardless of the reconstruction target. After applying the CLIP target and the prompting decoder, semantic aware sampling further improves the uniformally random sampling by 0.3%, yielding the best 85.4% accuracy among different model variants (#8 *vs.* #7 in Table 7.7). Semantic aware sampling slightly reduces the pretraining difficulty and obtains lower training loss, because it favours more important image regions, which can be illustrated in Figure 7.6. Although the task becomes easier, the learned representation enjoys better accuracy.

(4) To demonstrate that masked image pretraining can generally benefit from the reconstruction target assisted with language supervision, we use another recently proposed language-image pretraining method, SLIP [183], to generate the image features as the reconstruction target. Reconstructing the image features from SLIP still outperforms reconstructing raw pixels, surpassing MAE by 1.4% (#9 *vs.* #1 in Table 7.7). Since SLIP already incorporates contrastive based image self-supervision into language-image pretraining, directly finetuning SLIP's image encoder yields an accuracy of 82.6%. Interestingly, one more step of reconstruction based pretraining by MILAN further improves the representation quality, boosting the accuracy by another 1.8%.

Finally, we note that a very long pretraing schedule is no longer necessary for MILAN compared to MAE. Our method enjoys much fewer epochs while achieving higher accuracy: MILAN achieves 85.4% after a 400-epoch pretraining, while MAE achieves 83.6% after a 1600-epoch pretraining.

## 7.6 Conclusion

Large-scale pretraining on unlabeled data in a masked auto-encoder framework have shown great success in learning transferable representations leading to advances many different vision and language tasks. In vision domain, however, the learned visual concepts may still lag the rich semantic information compared to self-supervised learning on language data. Understanding images assisted by language captions has also been explored in the weakly supervised pretraining setting. The learned features from language-image pretraining can be transferred to downstream tasks via zero shot learning. However, directly finetuning these models do not reveal competitive results. In this chapter, we combined these two paradigms into a MILAN method by using the outputs from the language-image pretraining as the reconstruction target for the masked image pretraining, where the model receives incomplete input images but learns to reconstruct the full latent features that embed

rich semantics from language supervision. MILAN creates a more challenging objective, where the model not only needs to distill the knowledge from target features on unmasked patches but also learns to reconstruct the representations of the masked patches through the decoder. This challenging reconstruction objective can better leverage the guidance from the target features and even improve the representation quality compared to the targets. Moreover, we proposed a more effective decoder architecture and a semantic aware sampling mechanism to improve both the training efficiency and the quality of the learned representations. Extensive experiments have demonstrated that our method leads to higher accuracy on many different tasks than previous language-image based (e.g., CLIP) and masked prediction based (e.g., MAE) approaches.

# Chapter 8

# Epilogue

We have established in chapter 1 the basic subspace analysis metrics that have been repeatedly used as the selection or the optimization criteria for structural learning to obtain less redundant, more efficient, and more accurate deep learning models. We have progressively presented the structural learning methods from the simplest iterative approach in chapter 3 to more sophisticated channel exploration and omni-dimensional structural learning approaches in chapter 4. Although many of the introduced methods are related to neural network pruning, we have gone beyond model compression and demonstrate them as general approaches to achieve better model efficiency and accuracy. For example, the channel exploration method does not require a well-trained full model as the starting point. Instead, it starts from a random sub-structure and progressively adapt to more optimal structure, by reducing the redundant parameters while growing more orthogonal weights based on our subspace analysis metrics. This method provides a platform for future works to explore more accurate models in larger model space.

Apart from learning static architectures, we have also introduced in chapter 5 the dynamic neural networks that enjoy improved representation power and computation efficiency, by adapting the both architecture and parameters in both sample-wise and spatial-wise dependent during inference. The great success of dynamic neural networks may motivate future works on developing principled approaches with theoretical guarantees for the control functions in the dynamic neural networks so that the input-dependent decisions becomes stable and interpretable. Moreover, since the current deep learning hardware and libraries are mostly optimized for static models, future works on customizing the hardware to dynamic neural networks can better harness the runtime efficiency gains.

In addition to the model efficiency, we have also introduced approaches to improve the generaliza-

tion performance of deep learning models on low-resource or few-shot datasets in chapter 6. Few-shot learning attempts to build data-efficient models that can rapidly generalize from few labeled examples as human intelligence. Based on the error decomposition analysis, we have proposed few-shot architecture adaptation to customize the best hypothesis space for few-shot tasks to suppress the approximation error and have proposed semi-supervised few-shot learning to suppress the estimation error so that the empirical risk minimizer becomes more reliable.

In a similar vein, self-supervised learning also aims to learn models that generalize well and can produce highly transferable representations without large-scale supervised data. We have introduce in chapter 7 a cross-modality self-supervised learning framework that harness the best from two paradigms: the masked autoencoder based pretraining and the language supervised visual representation learning. The demonstrated scaling property, feature clustering property, transferability to different tasks and domains, label efficiency and adversarial robustness of the resulting models may motivate future works to build more powerful vision foundation models from a cross-modality perspective. We have only scratched the surface of future directions and challenges towards building more efficient, modular, generalizable, and easily adaptable deep learning models, and we believe it to be a bright area of research with many breakthroughs to come.

# Bibliography


[1] David Barber Felix Agakov. The im algorithm: a variational approach to information maximization. *Advances in neural information processing systems*, 16:201, 2004.

[2] Armen Aghajanyan, Sonal Gupta, and Luke Zettlemoyer. Intrinsic dimensionality explains the effectiveness of language model fine-tuning. In *Proceedings of the 59th Annual Meeting of the Association for Computational Linguistics and the 11th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*, pages 7319–7328, 2021.

[3] Eirikur Agustsson and Radu Timofte. Ntire 2017 challenge on single image super-resolution: Dataset and study. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition Workshops*, pages 126–135, 2017.

[4] Namhyuk Ahn, Byungkon Kang, and Kyung-Ah Sohn. Fast, accurate, and lightweight super-resolution with cascading residual network. In *Proceedings of the European Conference on Computer Vision (ECCV)*, pages 252–268, 2018.

[5] Guillaume Alain and Yoshua Bengio. Understanding intermediate layers using linear classifier probes. *arXiv preprint arXiv:1610.01644*, 2016.

[6] Dosovitskiy Alexey, Philipp Fischer, Jost Tobias, Martin Riedmiller Springenberg, and Thomas Brox. Discriminative, unsupervised feature learning with exemplar convolutional, neural networks. In *Transactions on Pattern Analysis and Machine Intelligence*, volume 38, pages 1734–1747, 2016.

[7] Antreas Antoniou, Harrison Edwards, and Amos Storkey. How to train your maml. *arXiv preprint arXiv:1810.09502*, 2018.

[8] Alexei Baevski, Wei-Ning Hsu, Qiantong Xu, Arun Babu, Jiatao Gu, and Michael Auli. Data2vec: A general framework for self-supervised learning in speech, vision and language. *arXiv preprint arXiv:2202.03555*, 2022.

[9] Charles R Baker. Joint measures and cross-covariance operators. *Transactions of the American Mathematical Society*, 1973.

[10] Nitin Bansal, Xiaohan Chen, and Zhangyang Wang. Can we gain more from orthogonality regularizations in training deep cnns? *Proceedings of Advances in Neural Information Processing Systems*, 2018.

[11] Hangbo Bao, Li Dong, and Furu Wei. Beit: Bert pre-training of image transformers. *arXiv preprint arXiv:2106.08254*, 2021.

[12] Peyman Bateni, Raghav Goyal, Vaden Masrani, Frank Wood, and Leonid Sigal. Improved few-shot visual classification. *CVPR*, 2020.

[13] Yoshua Bengio, Nicholas Léonard, and Aaron Courville. Estimating or propagating gradients through stochastic neurons for conditional computation. *arXiv preprint arXiv:1308.3432*, 2013.

[14] David Berthelot, Nicholas Carlini, Ian Goodfellow, Nicolas Papernot, Avital Oliver, and Colin A Raffel. Mixmatch: A holistic approach to semi-supervised learning. *Advances in neural information processing systems*, 32, 2019.

[15] Marco Bevilacqua, Aline Roumy, Christine Guillemot, and Marie Line Alberi-Morel. Low-complexity single-image super-resolution based on nonnegative neighbor embedding. 2012.

[16] Kartikeya Bhardwaj, Milos Milosavljevic, Alex Chalfin, Naveen Suda, Liam O'Neil, Dibakar Gope, Lingchuan Meng, Ramon Matas, and Danny Loh. Collapsible linear blocks for super-efficient super resolution. *arXiv preprint arXiv:2103.09404*, 2021.

[17] Léon Bottou and Olivier Bousquet. The tradeoffs of large scale learning. *Advances in neural information processing systems*, 20, 2007.

[18] Malik Boudiaf, Ziko Imtiaz Masud, Jérôme Rony, José Dolz, Pablo Piantanida, and Ismail Ben Ayed. Transductive information maximization for few-shot learning. *NeurIPS*, 2020.

[19] Han Cai, Chuang Gan, Tianzhe Wang, Zhekai Zhang, and Song Han. OFA-ResNet-50. `https://github.com/mit-han-lab/once-for-all`, 2020.

[20] Han Cai, Chuang Gan, Tianzhe Wang, Zhekai Zhang, and Song Han. Once-for-all: Train one network and specialize it for efficient deployment. *Proceedings of International Conference on Learning Representations*, 2020.

[21] Mathilde Caron, Piotr Bojanowski, Armand Joulin, and Matthijs Douze. Deep clustering for unsupervised learning of visual features. In *European Conference on Computer Vision*, pages 132–149, 2018.

[22] Mathilde Caron, Ishan Misra, Julien Mairal, Priya Goyal, Piotr Bojanowski, and Armand Joulin. Unsupervised learning of visual features by contrasting cluster assignments. In *Advances in Neural Information Processing Systems*, volume 33, pages 9912–9924, 2020.

[23] Mathilde Caron, Hugo Touvron, Ishan Misra, Hervé Jégou, Julien Mairal, Piotr Bojanowski, and Armand Joulin. Emerging properties in self-supervised vision transformers. In *International Conference on Computer Vision*, pages 9650–9660, 2021.

[24] Xiangyu Chang, Yingcong Li, Samet Oymak, and Christos Thrampoulidis. Provable benefits of overparameterization in model compression: From double descent to pruning neural networks. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 35, pages 6974–6983, 2021.

[25] Boyu Chen, Peixia Li, Chuming Li, Baopu Li, Lei Bai, Chen Lin, Ming Sun, Junjie Yan, and Wanli Ouyang. Glit: Neural architecture search for global and local image transformer. In *ICCV*, 2021.

[26] Jianbo Chen, Mitchell Stern, Martin J Wainwright, and Michael I Jordan. Kernel feature selection via conditional covariance minimization. In *Advances in Neural Information Processing Systems*, pages 6946–6955, 2017.

[27] Jin Chen, Xijun Wang, Zichao Guo, Xiangyu Zhang, and Jian Sun. Dynamic region-aware convolution. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 8064–8073, 2021.

[28] Minghao Chen, Houwen Peng, Jianlong Fu, and Haibin Ling. Autoformer: Searching transformers for visual recognition. In *ICCV*, 2021.

[29] Mark Chen, Alec Radford, Rewon Child, Jeffrey Wu, Heewoo Jun, David Luan, and Ilya Sutskever. Generative pretraining from pixels. In *International Conference on Machine Learning*, pages 1691–1703, 2020.

[30] Tianlong Chen, Yu Cheng, Zhe Gan, Lu Yuan, Lei Zhang, and Zhangyang Wang. Chasing sparsity in vision transformers: An end-to-end exploration. In *NeurIPS*, 2021.

[31] Ting Chen, Simon Kornblith, Mohammad Norouzi, and Geoffrey Hinton. A simple framework for contrastive learning of visual representations. In *International Conference on Machine Learning*, pages 1597–1607. PMLR, 2020.

[32] Ting Chen, Simon Kornblith, Kevin Swersky, Mohammad Norouzi, and Geoffrey E Hinton. Big self-supervised models are strong semi-supervised learners. *Advances in neural information processing systems*, 33:22243–22255, 2020.

[33] Wei-Yu Chen, Yen-Cheng Liu, Zsolt Kira, Yu-Chiang Frank Wang, and Jia-Bin Huang. A closer look at few-shot classification. *ICLR*, 2019.

[34] Xiaokang Chen, Mingyu Ding, Xiaodi Wang, Ying Xin, Shentong Mo, Yunhao Wang, Shumin Han, Ping Luo, Gang Zeng, and Jingdong Wang. Context autoencoder for self-supervised representation learning. *arXiv preprint arXiv:2202.03026*, 2022.

[35] Xinlei Chen and Kaiming He. Exploring simple siamese representation learning. In *Conference on Computer Vision and Pattern Recognition*, pages 15750–15758, 2021.

[36] Xinlei Chen, Saining Xie, and Kaiming He. An empirical study of training self-supervised vision transformers. In *International Conference on Computer Vision*, pages 9640–9649, 2021.

[37] Yinpeng Chen, Xiyang Dai, Mengchen Liu, Dongdong Chen, Lu Yuan, and Zicheng Liu. Dynamic convolution: Attention over convolution kernels. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 11030–11039, 2020.

[38] Ruizhe Cheng, Bichen Wu, Peizhao Zhang, Peter Vajda, and Joseph E Gonzalez. Data-efficient language-supervised zero-shot learning with self-distillation. In *Conference on Computer Vision and*

*Pattern Recognition*, pages 3119–3124, 2021.

[39] Rewon Child, Scott Gray, Alec Radford, and Ilya Sutskever. Generating long sequences with sparse transformers. *arXiv preprint arXiv:1904.10509*, 2019.

[40] Ting-Wu Chin, Ruizhou Ding, Cha Zhang, and Diana Marculescu. Towards efficient model compression via learned global ranking. *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2020.

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

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

[43] Karan Desai and Justin Johnson. Virtex: Learning visual representations from textual annotations. In *Conference on Computer Vision and Pattern Recognition*, pages 11162–11173, 2021.

[44] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*, 2018.

[45] Guneet S Dhillon, Pratik Chaudhari, Avinash Ravichandran, and Stefano Soatto. A baseline for few-shot image classification. *ICLR*, 2020.

[46] Xiaohan Ding, Guiguang Ding, Yuchen Guo, and Jungong Han. Centripetal sgd for pruning very deep convolutional networks with complicated structure. *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 4943–4953, 2019.

[47] Xiaohan Ding, Guiguang Ding, Yuchen Guo, Jungong Han, and Chenggang Yan. Approximated oracle filter pruning for destructive cnn width optimization. *Proceedings of International Conference on Machine Learning*, 2019.

[48] Xiaohan Ding, Tianxiang Hao, Jianchao Tan, Ji Liu, Jungong Han, Yuchen Guo, and Guiguang Ding. Lossless cnn channel pruning via decoupling remembering and forgetting. *Proceedings of the IEEE/CVF International Conference on Com-puter Vision*, 2021.

[49] Carl Doersch, Abhinav Gupta, and Alexei A Efros. Unsupervised visual representation learning by context prediction. In *International Conference on Computer Vision*, pages 1422–1430, 2015.

[50] Chao Dong, Chen Change Loy, Kaiming He, and Xiaoou Tang. Image super-resolution using deep convolutional networks. *IEEE transactions on pattern analysis and machine intelligence*, 38(2):295–307, 2015.

[51] Chao Dong, Chen Change Loy, and Xiaoou Tang. Accelerating the super-resolution convolutional neural network. In *European conference on computer vision*, pages 391–407. Springer, 2016.

[52] Xiaoyi Dong, Jianmin Bao, Ting Zhang, Dongdong Chen, Weiming Zhang, Lu Yuan, Dong Chen, Fang Wen, and Nenghai Yu. Peco: Perceptual codebook for bert pre-training of vision transformers. *arXiv preprint arXiv:2111.12710*, 2021.

[53] Xuanyi Dong and Yi Yang. Network pruning via transformable architecture search. *Proceedings of Advances in Neural Information Processing Systems*, pages 759–770, 2019.

[54] Alexey Dosovitskiy, Lucas Beyer, Alexander Kolesnikov, Dirk Weissenborn, Xiaohua Zhai, Thomas Unterthiner, Mostafa Dehghani, Matthias Minderer, Georg Heigold, Sylvain Gelly, et al. An image is worth 16x16 words: Transformers for image recognition at scale. *arXiv preprint arXiv:2010.11929*, 2020.

[55] Alexey Dosovitskiy, Lucas Beyer, Alexander Kolesnikov, Dirk Weissenborn, Xiaohua Zhai, Thomas Unterthiner, Mostafa Dehghani, Matthias Minderer, Georg Heigold, Sylvain Gelly, et al. An image is worth 16x16 words: Transformers for image recognition at scale. In *ICLR*, 2021.

[56] Alaaeldin El-Nouby, Gautier Izacard, Hugo Touvron, Ivan Laptev, Hervé Jegou, and Edouard Grave. Are large-scale datasets necessary for self-supervised pre-training? *arXiv preprint arXiv:2112.10740*, 2021.

[57] Aleksandr Ermolov, Aliaksandr Siarohin, Enver Sangineto, and Nicu Sebe. Whitening for self-supervised representation learning. In *International Conference on Machine Learning*, pages 3015–3024, 2021.

[58] Li Fei-Fei, Robert Fergus, and Pietro Perona. One-shot learning of object categories. *IEEE transactions on pattern analysis and machine intelligence*, 28(4):594–611, 2006.

[59] Chelsea Finn, Pieter Abbeel, and Sergey Levine. Model-agnostic meta-learning for fast adaptation of deep networks. *ICML*, 2017.

[60] Andrea Frome, Greg S Corrado, Jon Shlens, Samy Bengio, Jeff Dean, Marc'Aurelio Ranzato, and Tomas Mikolov. Devise: A deep visual-semantic embedding model. In *Advances in Neural Information Processing Systems*, volume 26, 2013.

[61] Hang Gao, Xizhou Zhu, Steve Lin, and Jifeng Dai. Deformable kernels: Adapting effective receptive fields for object deformation. *arXiv preprint arXiv:1910.02940*, 2019.

[62] Shangqian Gao, Feihu Huang, Weidong Cai, and Heng Huang. Network pruning via performance maximization. *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 9270–9280, 2021.

[63] Shangqian Gao, Feihu Huang, Jian Pei, and Heng Huang. Discrete model compression with resource constraint for deep neural networks. *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 1899–1908, 2020.

[64] Xitong Gao, Yiren Zhao, Łukasz Dudziak, Robert Mullins, and Cheng-zhong Xu. Dynamic channel pruning: Feature boosting and suppression. *arXiv preprint arXiv:1810.05331*, 2018.

[65] Claire Gardent, Anastasia Shimorina, Shashi Narayan, and Laura Perez-Beltrachini. Creating training corpora for nlg micro-planning. In *55th annual meeting of the Association for Computational Linguistics (ACL)*, 2017.

[66] Priya Goyal, Mathilde Caron, Benjamin Lefaudeux, Min Xu, Pengchao Wang, Vivek Pai, Mannat Singh, Vitaliy Liptchinsky, Ishan Misra, Armand Joulin, et al. Self-supervised pretraining of visual features in the wild. *arXiv preprint arXiv:2103.01988*, 2021.

[67] Arthur Gretton, Olivier Bousquet, Alex Smola, and Bernhard Schölkopf. Measuring statistical dependence with hilbert-schmidt norms. *ALT*, 2005.

[68] Jean-Bastien Grill, Florian Strub, Florent Altché, Corentin Tallec, Pierre Richemond, Elena Buchatskaya, Carl Doersch, Bernardo Avila Pires, Zhaohan Guo, Mohammad Gheshlaghi Azar, et al. Bootstrap your own latent-a new approach to self-supervised learning. In *Advances in Neural Information Processing Systems*, volume 33, pages 21271–21284, 2020.

[69] Ming Gu and Stanley C Eisenstat. Efficient algorithms for computing a strong rank-revealing qr factorization. *SIAM Journal on Scientific Computing*, 17(4):848–869, 1996.

[70] Jinyang Guo, Wanli Ouyang, and Dong Xu. Channel pruning guided by classification loss and feature importance. *Proceedings of AAAI*, 2020.

[71] Jinyang Guo, Wanli Ouyang, and Dong Xu. Multi-dimensional pruning: A unified framework for model compression. *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 1508–1517, 2020.

[72] Shaopeng Guo, Yujie Wang, Quanquan Li, and Junjie Yan. Dmcp: Differentiable markov channel pruning for neural networks. *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 1539–1547, 2020.

[73] Yi Guo, Huan Yuan, Jianchao Tan, Zhangyang Wang, Sen Yang, and Ji Liu. Gdp: Stabilized neural network pruning via gates with differentiable polarization. *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 5239–5250, 2021.

[74] Zichao Guo, Xiangyu Zhang, Haoyuan Mu, Wen Heng, Zechun Liu, Yichen Wei, and Jian Sun. Single path one-shot neural architecture search with uniform sampling. In *ECCV*, 2020.

[75] Suchin Gururangan, Ana Marasović, Swabha Swayamdipta, Kyle Lo, Iz Beltagy, Doug Downey, and Noah A Smith. Don't stop pretraining: Adapt language models to domains and tasks. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pages 8342–8360, 2020.

[76] Isabelle Guyon, Jason Weston, Stephen Barnhill, and Vladimir Vapnik. Gene selection for cancer classification using support vector machines. *Machine learning*, 46(1-3):389–422, 2002.

[77] Kai Han, Yunhe Wang, Qi Tian, Jianyuan Guo, Chunjing Xu, and Chang Xu. Ghostnet: More features from cheap operations. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 1580–1589, 2020.

[78] Haoyu He, Jing Liu, Zizheng Pan, Jianfei Cai, Jing Zhang, Dacheng Tao, and Bohan Zhuang. Pruning self-attentions into convolutional layers in single path. *arXiv preprint arXiv:2111.11802*, 2021.

[79] Junxian He, Chunting Zhou, Xuezhe Ma, Taylor Berg-Kirkpatrick, and Graham Neubig. Towards a unified view of parameter-efficient transfer learning. *arXiv preprint arXiv:2110.04366*, 2021.

[80] Kaiming He, Xinlei Chen, Saining Xie, Yanghao Li, Piotr Dollár, and Ross Girshick. Masked autoencoders are scalable vision learners. *arXiv preprint arXiv:2111.06377*, 2021.

[81] Kaiming He, Haoqi Fan, Yuxin Wu, Saining Xie, and Ross Girshick. Momentum contrast for unsupervised visual representation learning. In *Conference on Computer Vision and Pattern Recognition*, pages 9729–9738, 2020.

[82] Kaiming He, Georgia Gkioxari, Piotr Dollár, and Ross Girshick. Mask r-cnn. In *International Conference on Computer Vision*, pages 2961–2969, 2017.

[83] Kaiming He, Georgia Gkioxari, Piotr Dollár, and Ross Girshick. Mask r-cnn. *Proceedings of the IEEE international conference on computer vision*, pages 2961–2969, 2017.

[84] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. *Proceedings of the IEEE international conference on computer vision*, pages 1026–1034, 2015.

[85] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016.

[86] Pengcheng He, Xiaodong Liu, Jianfeng Gao, and Weizhu Chen. Deberta: Decoding-enhanced bert with disentangled attention. *arXiv preprint arXiv:2006.03654*, 2020.

[87] Yang He, Yuhang Ding, Ping Liu, Linchao Zhu, Hanwang Zhang, and Yi Yang. Learning filter pruning criteria for deep convolutional neural networks acceleration. *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 2009–2018, 2020.

[88] Yang He, Guoliang Kang, Xuanyi Dong, Yanwei Fu, and Yi Yang. Soft filter pruning for accelerating deep convolutional neural networks. *Proceedings of International Joint Conference on Artificial Intelligence*, 2018.

[89] Yihui He, Ji Lin, Zhijian Liu, Hanrui Wang, Li-Jia Li, and Song Han. Amc: Automl for model compression and acceleration on mobile devices. *Proceedings of the European Conference on Computer Vision*, pages 784–800, 2018.

[90] Yang He, Ping Liu, Ziwei Wang, Zhilan Hu, and Yi Yang. Filter pruning via geometric median for deep convolutional neural networks acceleration. *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 4340–4349, 2019.

[91] Yihui He, Xiangyu Zhang, and Jian Sun. Channel pruning for accelerating very deep neural networks. *Proceedings of the IEEE International Conference on Computer Vision*, pages 1389–1397, 2017.

[92] Olivier Henaff. Data-efficient image recognition with contrastive predictive coding. In *International conference on machine learning*, pages 4182–4192. PMLR, 2020.

[93] Dan Hendrycks, Steven Basart, Norman Mu, Saurav Kadavath, Frank Wang, Evan Dorundo, Rahul Desai, Tyler Zhu, Samyak Parajuli, Mike Guo, et al. The many faces of robustness: A critical analysis of out-of-distribution generalization. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 8340–8349, 2021.

[94] Dan Hendrycks, Kevin Zhao, Steven Basart, Jacob Steinhardt, and Dawn Song. Natural adversarial examples. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 15262–15271, 2021.

[95] Roger A Horn and Charles R Johnson. Matrix analysis. 2012.

[96] Qibin Hou, Daquan Zhou, and Jiashi Feng. Coordinate attention for efficient mobile network design. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 13713–13722, 2021.

[97] Ruibing Hou, Hong Chang, MA Bingpeng, Shiguang Shan, and Xilin Chen. Cross attention network for few-shot classification. *NeurIPS*, 2019.

[98] Yuenan Hou, Zheng Ma, Chunxiao Liu, Zhe Wang, and Chen Change Loy. Network pruning via resource reallocation. *arXiv preprint arXiv:2103.01847*, 2021.

[99] Zejiang Hou and Sun-Yuan Kung. A kernel discriminant information approach to non-linear feature selection. *Proceedings of the IEEE International Joint Conference on Neural Networks*, 2019.

[100] Zejiang Hou and Sun-Yuan Kung. A discriminant information approach to deep neural network pruning. *Proceedings of the IEEE International Conference on Pattern Recognition*, pages 9553–9560, 2020.

[101] Zejiang Hou and Sun-Yuan Kung. Efficient image super resolution via channel discriminative deep neural network pruning. *Proceedings of IEEE International Conference on Acoustics, Speech and Signal Processing*, pages 3647–3651, 2020.

[102] Zejiang Hou and Sun-Yuan Kung. Efficient image super resolution via channel discriminative deep neural network pruning. *Proceedings of the IEEE International Conference on Acoustics, Speech and Signal Processing*, pages 3647–3651, 2020.

[103] Zejiang Hou and Sun-Yuan Kung. A feature-map discriminant perspective for pruning deep neural networks. *arXiv preprint arXiv:2005.13796*, 2020.

[104] Zejiang Hou and Sun-Yuan Kung. Parameter efficient dynamic convolution via tensor decomposition. *Proceedings of the British Machine Vision Conference*, 2021.

[105] Zejiang Hou and Sun-Yuan Kung. Multi-dimensional dynamic model compression for efficient image super-resolution. *Proceedings of the IEEE Winter Conference on Applications of Computer Vision*, pages 633–643, 2022.

[106] Zejiang Hou and Sun-Yuan Kung. Multi-dimensional dynamic model compression for efficient image super-resolution. *Proceedings of the IEEE/CVF Winter Conference on Applications of Computer Vision*, pages 633–643, 2022.

[107] Zejiang Hou and Sun-Yuan Kung. Multi-dimensional vision transformer compression via dependency guided gaussian process search. *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 3669–3678, 2022.

[108] Zejiang Hou and Sun-Yuan Kung. Semi-supervised few-shot learning via dependency maximization and instance discriminant analysis. *Journal of Signal Processing Systems*, 2022.

[109] Zejiang Hou, Minghai Qin, Fei Sun, Xiaolong Ma, Kun Yuan, Yi Xu, Yen-Kuang Chen, Rong Jin, Yuan Xie, and Sun-Yuan Kung. Chex: Channel exploration for cnn model compression. *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 12287–12298, 2022.

[110] Zejiang Hou, Julian Salazar, and George Polovets. Meta-learning the difference: Preparing large language models for efficient adaptation. *Transactions of the Association for Computational Linguistics*, 2022.

[111] Zejiang Hou, Julian Salazar, and George Polovets. Meta-learning the difference: Preparing large language models for efficient adaptation. *Transactions of the Association for Computational Linguistics*, 2022.

[112] Neil Houlsby, Andrei Giurgiu, Stanislaw Jastrzebski, Bruna Morrone, Quentin De Laroussilhe, Andrea Gesmundo, Mona Attariyan, and Sylvain Gelly. Parameter-efficient transfer learning for nlp. In *International Conference on Machine Learning*, pages 2790–2799. PMLR, 2019.

[113] Andrew Howard, Mark Sandler, Grace Chu, Liang-Chieh Chen, Bo Chen, Mingxing Tan, Weijun Wang, Yukun Zhu, Ruoming Pang, Vijay Vasudevan, et al. Searching for mobilenetv3. *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 1314–1324, 2019.

[114] Andrew G Howard, Menglong Zhu, Bo Chen, Dmitry Kalenichenko, Weijun Wang, Tobias Weyand, Marco Andreetto, and Hartwig Adam. Mobilenets: Efficient convolutional neural networks for mobile vision applications. *arXiv preprint arXiv:1704.04861*, 2017.

[115] Edward J Hu, Yelong Shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Lu Wang, and Weizhu Chen. Lora: Low-rank adaptation of large language models. *arXiv preprint arXiv:2106.09685*, 2021.

[116] Jie Hu, Li Shen, and Gang Sun. Squeeze-and-excitation networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 7132–7141, 2018.

[117] Shell Xu Hu, Pablo G Moreno, Yang Xiao, Xi Shen, Guillaume Obozinski, Neil D Lawrence, and Andreas Damianou. Empirical bayes transductive meta-learning with synthetic gradients. *ICLR*, 2020.

[118] Jia-Bin Huang, Abhishek Singh, and Narendra Ahuja. Single image super-resolution from transformed self-exemplars. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*,

pages 5197–5206, 2015.

[119] Zehao Huang and Naiyan Wang. Data-driven sparse structure selection for deep neural networks. *Proceedings of the European Conference on Computer Vision*, pages 304–320, 2018.

[120] Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. *Proceedings of International Conference on Machine Learning*, pages 448–456, 2015.

[121] P ITU-T RECOMMENDATION. Subjective video quality assessment methods for multimedia applications. *International telecommunication union*, 1999.

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

[123] Chao Jia, Yinfei Yang, Ye Xia, Yi-Ting Chen, Zarana Parekh, Hieu Pham, Quoc Le, Yun-Hsuan Sung, Zhen Li, and Tom Duerig. Scaling up visual and vision-language representation learning with noisy text supervision. In *International Conference on Machine Learning*, pages 4904–4916. PMLR, 2021.

[124] Ding Jia, Kai Han, Yunhe Wang, Yehui Tang, Jianyuan Guo, Chao Zhang, and Dacheng Tao. Efficient vision transformers via fine-grained manifold distillation. *arXiv preprint arXiv:2107.01378*, 2021.

[125] Donald R Jones, Matthias Schonlau, and William J Welch. Efficient global optimization of expensive black-box functions. *Journal of Global optimization*, 13(4):455–492, 1998.

[126] Ioannis Kakogeorgiou, Spyros Gidaris, Bill Psomas, Yannis Avrithis, Andrei Bursuc, Konstantinos Karantzalos, and Nikos Komodakis. What to hide from your students: Attention-guided masked image modeling. *arXiv preprint arXiv:2203.12719*, 2022.

[127] Minsoo Kang and Bohyung Han. Operation-aware soft channel pruning using differentiable masks. *Proceedings of International Conference on Machine Learning*, pages 5122–5131, 2020.

[128] Jiwon Kim, Jung Kwon Lee, and Kyoung Mu Lee. Accurate image super-resolution using very deep convolutional networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 1646–1654, 2016.

[129] Diederik P Kingma and Max Welling. Auto-encoding variational bayes. *arXiv preprint arXiv:1312.6114*, 2013.

[130] Dieter Kraft et al. A software package for sequential quadratic programming. 1988.

[131] Alex Krizhevsky, Vinod Nair, and Geoffrey Hinton. The cifar-10 dataset. *online: http://www. cs. toronto. edu/kriz/cifar. html*, page 4, 2014.

[132] Sun Yuan Kung. *Kernel methods and machine learning*. Cambridge University Press, 2014.

[133] Sun-Yuan Kung and Zejiang Hou. Augment deep bp-parameter learning with local xai-structural learning. *Communications in Information and Systems*, 20(3):319–352, 2020.

[134] Chen-Yu Lee, Saining Xie, Patrick Gallagher, Zhengyou Zhang, and Zhuowen Tu. Deeply-supervised nets. In *Artificial intelligence and statistics*, pages 562–570, 2015.

[135] Kwonjoon Lee, Subhransu Maji, Avinash Ravichandran, and Stefano Soatto. Meta-learning with differentiable convex optimization. *CVPR*, 2019.

[136] Namhoon Lee, Thalaiyasingam Ajanthan, and Philip HS Torr. Snip: Single-shot network pruning based on connection sensitivity. *Proceedings of International Conference on Learning Rrepresentations*, 2019.

[137] Royson Lee, Łukasz Dudziak, Mohamed Abdelfattah, Stylianos I Venieris, Hyeji Kim, Hongkai Wen, and Nicholas D Lane. Journey towards tiny perceptual super-resolution. In *European Conference on Computer Vision*, pages 85–102. Springer, 2020.

[138] Mike Lewis, Yinhan Liu, Naman Goyal, Marjan Ghazvininejad, Abdelrahman Mohamed, Omer Levy, Veselin Stoyanov, and Luke Zettlemoyer. Bart: Denoising sequence-to-sequence pre-training for natural language generation, translation, and comprehension. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pages 7871–7880, 2020.

[139] Bailin Li, Bowen Wu, Jiang Su, and Guangrun Wang. Eagleeye: Fast sub-net evaluation for efficient neural network pruning. *Proceedings of European Conference on Computer Vision*, pages 639–654, 2020.

[140] Changlin Li, Guangrun Wang, Bing Wang, Xiaodan Liang, Zhihui Li, and Xiaojun Chang. Dynamic slimmable network. *Proceedings of the IEEE/CVF International Conference on Computer Vision*, 2021.

[141] Hao Li, Asim Kadav, Igor Durdanovic, Hanan Samet, and Hans Peter Graf. Pruning filters for efficient convnets. *Proceedings of International Conference on Learning Representations*, 2017.

[142] Junnan Li, Pan Zhou, Caiming Xiong, and Steven CH Hoi. Prototypical contrastive learning of unsupervised representations. *arXiv preprint arXiv:2005.04966*, 2020.

[143] Tuanhui Li, Baoyuan Wu, Yujiu Yang, Yanbo Fan, Yong Zhang, and Wei Liu. Compressing convolutional neural networks via factorized convolutional filters. *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 3977–3986, 2019.

[144] Xiaotong Li, Yixiao Ge, Kun Yi, Zixuan Hu, Ying Shan, and Ling-Yu Duan. mc-beit: Multi-choice discretization for image bert pre-training. *arXiv preprint arXiv:2203.15371*, 2022.

[145] Xinzhe Li, Qianru Sun, Yaoyao Liu, Qin Zhou, Shibao Zheng, Tat-Seng Chua, and Bernt Schiele. Learning to self-train for semi-supervised few-shot classification. *NeurIPS*, 2019.

[146] Xiang Li, Wenhai Wang, Xiaolin Hu, and Jian Yang. Selective kernel networks. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 510–519, 2019.

[147] Xiang Lisa Li and Percy Liang. Prefix-tuning: Optimizing continuous prompts for generation. *arXiv preprint arXiv:2101.00190*, 2021.

[148] Yunsheng Li, Yinpeng Chen, Xiyang Dai, Mengchen Liu, Dongdong Chen, Ye Yu, Lu Yuan, Zicheng Liu, Mei Chen, and Nuno Vasconcelos. Revisiting dynamic convolution via matrix decomposition. *arXiv preprint arXiv:2103.08756*, 2021.

[149] Yawei Li, Shuhang Gu, Christoph Mayer, Luc Van Gool, and Radu Timofte. Group sparsity: The hinge between filter pruning and decomposition for network compression. *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 8018–8027, 2020.

[150] Yangguang Li, Feng Liang, Lichen Zhao, Yufeng Cui, Wanli Ouyang, Jing Shao, Fengwei Yu, and Junjie Yan. Supervision exists everywhere: A data efficient contrastive language-image pre-training paradigm. *arXiv preprint arXiv:2110.05208*, 2021.

[151] Moshe Lichtenstein, Prasanna Sattigeri, Rogerio Feris, Raja Giryes, and Leonid Karlinsky. Tafssl: Task-adaptive feature sub-space learning for few-shot classification. *ECCV*, 2020.

[152] Lucas Liebenwein, Cenk Baykal, Harry Lang, Dan Feldman, and Daniela Rus. Provable filter pruning for efficient neural networks. *Proceedings of International Conference on Learning Representations*, 2020.

[153] Bee Lim, Sanghyun Son, Heewon Kim, Seungjun Nah, and Kyoung Mu Lee. Enhanced deep residual networks for single image super-resolution. In *Proceedings of the IEEE conference on computer vision and pattern recognition workshops*, pages 136–144, 2017.

[154] Mingbao Lin, Rongrong Ji, Yan Wang, Yichen Zhang, Baochang Zhang, Yonghong Tian, and Ling Shao. Hrank: Filter pruning using high-rank feature map. *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2020.

[155] Shaohui Lin, Rongrong Ji, Chenqian Yan, Baochang Zhang, Liujuan Cao, Qixiang Ye, Feiyue Huang, and David Doermann. Towards optimal structured cnn pruning via generative adversarial learning. *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 2790–2799, 2019.

[156] Tsung-Yi Lin, Piotr Dollár, Ross Girshick, Kaiming He, Bharath Hariharan, and Serge Belongie. Feature pyramid networks for object detection. In *Conference on Computer Vision and Pattern Recognition*, pages 2117–2125, 2017.

[157] Tsung-Yi Lin, Priya Goyal, Ross Girshick, Kaiming He, and Piotr Dollár. Focal loss for dense object detection. In *CVPR*, 2017.

[158] Tsung-Yi Lin, Michael Maire, Serge Belongie, James Hays, Pietro Perona, Deva Ramanan, Piotr Dollár, and C Lawrence Zitnick. Microsoft coco: Common objects in context. *Proceedings of European conference on computer vision*, pages 740–755, 2014.

[159] Tsung-Yi Lin, Michael Maire, Serge Belongie, James Hays, Pietro Perona, Deva Ramanan, Piotr Dollár, and C Lawrence Zitnick. Microsoft coco: Common objects in context. In *European Conference on Computer Vision*, pages 740–755. Springer, 2014.

[160] Huan Liu and Hiroshi Motoda. *Computational methods of feature selection*. CRC Press, 2007.

[161] Hanxiao Liu, Karen Simonyan, and Yiming Yang. Darts: Differentiable architecture search. *arXiv preprint arXiv:1806.09055*, 2018.

[162] Jinlu Liu, Liang Song, and Yongqiang Qin. Prototype rectification for few-shot learning. *ECCV*, 2020.

[163] Wei Liu, Dragomir Anguelov, Dumitru Erhan, Christian Szegedy, Scott Reed, Cheng-Yang Fu, and Alexander C Berg. Ssd: Single shot multibox detector. *Proceedings of European conference on computer vision*, pages 21–37, 2016.

[164] Yanbin Liu, Juho Lee, Minseop Park, Saehoon Kim, Eunho Yang, Sung Ju Hwang, and Yi Yang. Learning to propagate labels: Transductive propagation network for few-shot learning. *arXiv preprint arXiv:1805.10002*, 2018.

[165] Yinhan Liu, Myle Ott, Naman Goyal, Jingfei Du, Mandar Joshi, Danqi Chen, Omer Levy, Mike Lewis, Luke Zettlemoyer, and Veselin Stoyanov. Roberta: A robustly optimized bert pretraining approach. *arXiv preprint arXiv:1907.11692*, 2019.

[166] Zhuang Liu, Jianguo Li, Zhiqiang Shen, Gao Huang, Shoumeng Yan, and Changshui Zhang. Learning efficient convolutional networks through network slimming. *Proceedings of the IEEE International Conference on Computer Vision*, pages 2736–2744, 2017.

[167] Ze Liu, Yutong Lin, Yue Cao, Han Hu, Yixuan Wei, Zheng Zhang, Stephen Lin, and Baining Guo. Swin transformer: Hierarchical vision transformer using shifted windows. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 10012–10022, 2021.

[168] Zhuang Liu, Hanzi Mao, Chao-Yuan Wu, Christoph Feichtenhofer, Trevor Darrell, and Saining Xie. A convnet for the 2020s. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 11976–11986, 2022.

[169] Zechun Liu, Haoyuan Mu, Xiangyu Zhang, Zichao Guo, Xin Yang, Kwang-Ting Cheng, and Jian Sun. Metapruning: Meta learning for automatic neural network channel pruning. *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 3296–3305, 2019.

[170] Zechun Liu, Xiangyu Zhang, Zhiqiang Shen, Zhe Li, Yichen Wei, Kwang-Ting Cheng, and Jian Sun. Joint multi-dimension pruning. *arXiv preprint arXiv:2005.08931*, 2020.

[171] Christos Louizos, Max Welling, and Diederik P Kingma. Learning sparse neural networks through $l\_0$ regularization. *Proceedings of International Conference on Learning Representations*, 2018.

[172] Jian-Hao Luo and Jianxin Wu. Neural network pruning with residual-connections and limited-data. *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 1458–1467, 2020.

[173] Jian-Hao Luo, Jianxin Wu, and Weiyao Lin. Thinet: A filter level pruning method for deep neural network compression. *Proceedings of the IEEE international conference on computer vision*, pages 5058–5066, 2017.

[174] Ningning Ma, Xiangyu Zhang, Jiawei Huang, and Jian Sun. Weightnet: Revisiting the design space of weight networks. *arXiv preprint arXiv:2007.11823*, 2020.

[175] Xiaolong Ma, Sheng Lin, Shaokai Ye, Zhezhi He, Linfeng Zhang, Geng Yuan, Sia Huat Tan, Zhengang Li, Deliang Fan, Xuehai Qian, et al. Non-structured dnn weight pruning–is it beneficial in any platform? *IEEE Transactions on Neural Networks and Learning Systems (TNNLS)*, 2021.

[176] Andrea Madotto, Zhaojiang Lin, Chien-Sheng Wu, and Pascale Fung. Personalizing dialogue agents via meta-learning. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pages 5454–5459, 2019.

[177] Aleksander Madry, Aleksandar Makelov, Ludwig Schmidt, Dimitris Tsipras, and Adrian Vladu. Towards deep learning models resistant to adversarial attacks. *arXiv preprint arXiv:1706.06083*, 2017.

[178] Puneet Mangla, Nupur Kumari, Abhishek Sinha, Mayank Singh, Balaji Krishnamurthy, and Vineeth N Balasubramanian. Charting the right manifold: Manifold mixup for few-shot learning. *WACV*, 2020.

[179] Xiaofeng Mao, Gege Qi, Yuefeng Chen, Xiaodan Li, Ranjie Duan, Shaokai Ye, Yuan He, and Hui Xue. Towards robust vision transformer. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 12042–12051, 2022.

[180] David Martin, Charless Fowlkes, Doron Tal, Jitendra Malik, et al. A database of human segmented natural images and its application to evaluating segmentation algorithms and measuring ecological statistics. Iccv Vancouver:, 2001.

[181] Paul Michel, Omer Levy, and Graham Neubig. Are sixteen heads really better than one? *Advances in neural information processing systems*, 32, 2019.

[182] Pavlo Molchanov, Arun Mallya, Stephen Tyree, Iuri Frosio, and Jan Kautz. Importance estimation for neural network pruning. *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 11264–11272, 2019.

[183] Norman Mu, Alexander Kirillov, David Wagner, and Saining Xie. Slip: Self-supervision meets language-image pre-training. *arXiv preprint arXiv:2112.12750*, 2021.

[184] Walter Murray and Kien-Ming Ng. An algorithm for nonlinear optimization problems with binary variables. *Computational Optimization and Applications*, 47(2):257–288, 2010.

[185] Linyong Nan, Dragomir Radev, Rui Zhang, Amrit Rau, Abhinand Sivaprasad, Chiachun Hsieh, Xiangru Tang, Aadit Vyas, Neha Verma, Pranav Krishna, et al. Dart: Open-domain structured data record to text generation. In *Proceedings of the 2021 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 432–447, 2021.

[186] Feiping Nie, Shiming Xiang, Yangqing Jia, Changshui Zhang, and Shuicheng Yan. Trace ratio criterion for feature selection. In *AAAI*, volume 2, pages 671–676, 2008.

[187] Ying Nie, Kai Han, Zhenhua Liu, An Xiao, Yiping Deng, Chunjing Xu, and Yunhe Wang. Ghostsr: Learning ghost features for efficient image super-resolution. *arXiv preprint arXiv:2101.08525*, 2021.

[188] Xuefei Ning, Tianchen Zhao, Wenshuo Li, Peng Lei, Yu Wang, and Huazhong Yang. Dsa: More efficient budgeted pruning via differentiable sparsity allocation. *Proceedings of European Conference on*

*Computer Vision*, pages 592–607, 2020.

[189] Mehdi Noroozi and Paolo Favaro. Unsupervised learning of visual representations by solving jigsaw puzzles. In *European Conference on Computer Vision*, pages 69–84. Springer, 2016.

[190] Jekaterina Novikova, Ondřej Dušek, and Verena Rieser. The e2e dataset: New challenges for end-to-end generation. In *Proceedings of the 18th Annual SIGdial Meeting on Discourse and Dialogue*, pages 201–206, 2017.

[191] NVIDIA. DeepLearningExamples. `https://github.com/NVIDIA/DeepLearningExamples`.

[192] Yassine Ouali, Céline Hudelot, and Myriam Tami. Spatial contrastive learning for few-shot classification. In *Joint European Conference on Machine Learning and Knowledge Discovery in Databases*, pages 671–686. Springer, 2021.

[193] Bowen Pan, Rameswar Panda, Yifan Jiang, Zhangyang Wang, Rogerio Feris, and Aude Oliva. Ia-red$^2$: Interpretability-aware redundancy reduction for vision transformers. In *NeurIPS*, 2021.

[194] Dimitris Papailiopoulos, Anastasios Kyrillidis, and Christos Boutsidis. Provable deterministic leverage score sampling. *Proceedings of the ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 997–1006, 2014.

[195] Deepak Pathak, Philipp Krahenbuhl, Jeff Donahue, Trevor Darrell, and Alexei A Efros. Context encoders: Feature learning by inpainting. In *Conference on Computer Vision and Pattern Recognition*, pages 2536–2544, 2016.

[196] Hanchuan Peng, Fuhui Long, and Chris Ding. Feature selection based on mutual information criteria of max-dependency, max-relevance, and min-redundancy. *IEEE Transactions on pattern analysis and machine intelligence*, 27(8):1226–1238, 2005.

[197] Jonas Pfeiffer, Aishwarya Kamath, Andreas Rücklé, Kyunghyun Cho, and Iryna Gurevych. Adapterfusion: Non-destructive task composition for transfer learning. *arXiv preprint arXiv:2005.00247*, 2020.

[198] Hieu Pham, Melody Y Guan, Barret Zoph, Quoc V Le, and Jeff Dean. Efficient neural architecture search via parameter sharing. *arXiv preprint arXiv:1802.03268*, 2018.

[199] Ofir Press, Noah A Smith, and Omer Levy. Improving transformer models by reordering their sublayers. *arXiv preprint arXiv:1911.03864*, 2019.

[200] Charles R Qi, Li Yi, Hao Su, and Leonidas J Guibas. Pointnet++: Deep hierarchical feature learning on point sets in a metric space. *arXiv preprint arXiv:1706.02413*, 2017.

[201] Alec Radford, Jong Wook Kim, Chris Hallacy, Aditya Ramesh, Gabriel Goh, Sandhini Agarwal, Girish Sastry, Amanda Askell, Pamela Mishkin, Jack Clark, et al. Learning transferable visual models from natural language supervision. In *International Conference on Machine Learning*, pages 8748–8763. PMLR, 2021.

[202] Alec Radford, Jeffrey Wu, Rewon Child, David Luan, Dario Amodei, Ilya Sutskever, et al. Language models are unsupervised multitask learners. *OpenAI blog*, 1(8):9, 2019.

[203] Rajat Raina, Alexis Battle, Honglak Lee, Benjamin Packer, and Andrew Y Ng. Self-taught learning: transfer learning from unlabeled data. *ICML*, 2007.

[204] Yongming Rao, Wenliang Zhao, Benlin Liu, Jiwen Lu, Jie Zhou, and Cho-Jui Hsieh. Dynamicvit: Efficient vision transformers with dynamic token sparsification. In *NeurIPS*, 2021.

[205] Ao Ren, Tianyun Zhang, Shaokai Ye, Jiayu Li, Wenyao Xu, Xuehai Qian, Xue Lin, and Yanzhi Wang. Admm-nn: An algorithm-hardware co-design framework of dnns using alternating direction methods of multipliers. *Proceedings of International Conference on Architectural Support for Programming Languages and Operating Systems*, pages 925–938, 2019.

[206] Mengye Ren, Eleni Triantafillou, Sachin Ravi, Jake Snell, Kevin Swersky, Joshua B Tenenbaum, Hugo Larochelle, and Richard S Zemel. Meta-learning for semi-supervised few-shot classification. *ICLR*, 2018.

[207] Pau Rodríguez, Issam Laradji, Alexandre Drouin, and Alexandre Lacoste. Embedding propagation: Smoother manifold for few-shot classification. *ECCV*, 2020.

[208] Leonid I Rudin, Stanley Osher, and Emad Fatemi. Nonlinear total variation based noise removal algorithms. *Physica D: nonlinear phenomena*, 60(1-4):259–268, 1992.

[209] Andrei A Rusu, Dushyant Rao, Jakub Sygnowski, Oriol Vinyals, Razvan Pascanu, Simon Osindero, and Raia Hadsell. Meta-learning with latent embedding optimization. *arXiv preprint arXiv:1807.05960*, 2019.

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

[211] Mert Bulent Sariyildiz, Julien Perez, and Diane Larlus. Learning visual representations with caption annotations. In *European Conference on Computer Vision*, pages 153–170. Springer, 2020.

[212] Ramprasaath R Selvaraju, Michael Cogswell, Abhishek Das, Ramakrishna Vedantam, Devi Parikh, and Dhruv Batra. Grad-cam: Visual explanations from deep networks via gradient-based localization. In *Proceedings of the IEEE international conference on computer vision*, pages 618–626, 2017.

[213] Ali Shafahi, Mahyar Najibi, Amin Ghiasi, Zheng Xu, John Dickerson, Christoph Studer, Larry S Davis, Gavin Taylor, and Tom Goldstein. Adversarial training for free! *arXiv preprint arXiv:1904.12843*, 2019.

[214] Christian Simon, Piotr Koniusz, Richard Nock, and Mehrtash Harandi. Adaptive subspaces for few-shot learning. *CVPR*, 2020.

[215] Mannat Singh, Laura Gustafson, Aaron Adcock, Vinicius de Freitas Reis, Bugra Gedik, Raj Prateek Kosaraju, Dhruv Mahajan, Ross Girshick, Piotr Dollár, and Laurens van der Maaten. Revisiting weakly

supervised pre-training of visual perception models. *arXiv preprint arXiv:2201.08371*, 2022.

[216] Jake Snell, Kevin Swersky, and Richard Zemel. Prototypical networks for few-shot learning. *NeurIPS*, 2017.

[217] David So, Wojciech Mańke, Hanxiao Liu, Zihang Dai, Noam Shazeer, and Quoc V Le. Searching for efficient transformers for language modeling. *Advances in Neural Information Processing Systems*, 34:6010–6022, 2021.

[218] Kihyuk Sohn, David Berthelot, Nicholas Carlini, Zizhao Zhang, Han Zhang, Colin A Raffel, Ekin Dogus Cubuk, Alexey Kurakin, and Chun-Liang Li. Fixmatch: Simplifying semi-supervised learning with consistency and confidence. *Advances in neural information processing systems*, 33:596–608, 2020.

[219] Kihyuk Sohn, Honglak Lee, and Xinchen Yan. Learning structured output representation using deep conditional generative models. *Advances in neural information processing systems*, 28:3483–3491, 2015.

[220] Le Song, Alex Smola, Arthur Gretton, Justin Bedo, and Karsten Borgwardt. Feature selection via dependence maximization. *Journal of Machine Learning Research*, 13(May):1393–1434, 2012.

[221] Yiping Song, Zequn Liu, Wei Bi, Rui Yan, and Ming Zhang. Learning to customize model structures for few-shot dialogue generation tasks. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pages 5832–5841, 2020.

[222] Xiu Su, Shan You, Tao Huang, Fei Wang, Chen Qian, Changshui Zhang, and Chang Xu. Locally free weight sharing for network width search. *Proceedings of International Conference on Learning Representations*, 2021.

[223] Chen Sun, Abhinav Shrivastava, Saurabh Singh, and Abhinav Gupta. Revisiting unreasonable effectiveness of data in deep learning era. In *International Conference on Computer Vision*, pages 843–852, 2017.

[224] Qianru Sun, Yaoyao Liu, Tat-Seng Chua, and Bernt Schiele. Meta-transfer learning for few-shot learning. *CVPR*, 2019.

[225] Flood Sung, Yongxin Yang, Li Zhang, Tao Xiang, Philip HS Torr, and Timothy M Hospedales. Learning to compare: Relation network for few-shot learning. *CVPR*, 2018.

[226] Christian Szegedy, Sergey Ioffe, Vincent Vanhoucke, and Alexander A Alemi. Inception-v4, inception-resnet and the impact of residual connections on learning. In *Thirty-first AAAI conference on artificial intelligence*, 2017.

[227] Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, and Andrew Rabinovich. Going deeper with convolutions. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 1–9, 2015.

[228] Mingxing Tan and Quoc Le. Efficientnet: Rethinking model scaling for convolutional neural networks. In *International Conference on Machine Learning*, pages 6105–6114. PMLR, 2019.

[229] Yehui Tang, Kai Han, Yunhe Wang, Chang Xu, Jianyuan Guo, Chao Xu, and Dacheng Tao. Patch slimming for efficient vision transformers. *arXiv preprint arXiv:2106.02852*, 2021.

[230] Yehui Tang, Yunhe Wang, Yixing Xu, Dacheng Tao, Chunjing Xu, Chao Xu, and Chang Xu. Scop: Scientific control for reliable neural network pruning. *Proceedings of Advances in Neural Information Processing Systems*, 2020.

[231] Yonglong Tian, Chen Sun, Ben Poole, Dilip Krishnan, Cordelia Schmid, and Phillip Isola. What makes for good views for contrastive learning? In *Advances in Neural Information Processing Systems*, volume 33, pages 6827–6839, 2020.

[232] Yonglong Tian, Yue Wang, Dilip Krishnan, Joshua B Tenenbaum, and Phillip Isola. Rethinking few-shot image classification: a good embedding is all you need? In *European Conference on Computer Vision*, pages 266–282. Springer, 2020.

[233] Hugo Touvron, Matthieu Cord, Matthijs Douze, Francisco Massa, Alexandre Sablayrolles, and Hervé Jégou. Training data-efficient image transformers & distillation through attention. In *ICML*, 2021.

[234] Hugo Touvron, Matthieu Cord, and Hervé Jégou. Deit iii: Revenge of the vit. *arXiv preprint arXiv:2204.07118*, 2022.

[235] Aaron Van den Oord, Yazhe Li, and Oriol Vinyals. Representation learning with contrastive predictive coding. *arXiv preprint arXiv:1807.03748*, 2018.

[236] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. *NeurIPS*, 2017.

[237] Pascal Vincent, Hugo Larochelle, Isabelle Lajoie, Yoshua Bengio, Pierre-Antoine Manzagol, and Léon Bottou. Stacked denoising autoencoders: Learning useful representations in a deep network with a local denoising criterion. In *Journal of Machine Learning Research*, volume 11, 2010.

[238] Oriol Vinyals, Charles Blundell, Timothy Lillicrap, Daan Wierstra, et al. Matching networks for one shot learning. *NeurIPS*, 2016.

[239] Elena Voita, David Talbot, Fedor Moiseev, Rico Sennrich, and Ivan Titov. Analyzing multi-head self-attention: Specialized heads do the heavy lifting, the rest can be pruned. *arXiv preprint arXiv:1905.09418*, 2019.

[240] Thang Vu, Cao Van Nguyen, Trung X Pham, Tung M Luu, and Chang D Yoo. Fast and efficient image quality enhancement via desubpixel convolutional neural networks. In *Proceedings of the European Conference on Computer Vision (ECCV)*, pages 0–0, 2018.

[241] Catherine Wah, Steve Branson, Peter Welinder, Pietro Perona, and Serge Belongie. The caltech-ucsd birds-200-2011 dataset. *Technical report*, 2011.

[242] Alex Wang, Yada Pruksachatkun, Nikita Nangia, Amanpreet Singh, Julian Michael, Felix Hill, Omer Levy, and Samuel Bowman. Superglue: A stickier benchmark for general-purpose language understanding systems. *Advances in neural information processing systems*, 32, 2019.

[243] Alex Wang, Amanpreet Singh, Julian Michael, Felix Hill, Omer Levy, and Samuel R Bowman. Glue: A multi-task benchmark and analysis platform for natural language understanding. *arXiv preprint arXiv:1804.07461*, 2018.

[244] Chaoqi Wang, Guodong Zhang, and Roger Grosse. Picking winning tickets before training by preserving gradient flow. *arXiv preprint arXiv:2002.07376*, 2020.

[245] Haohan Wang, Songwei Ge, Zachary Lipton, and Eric P Xing. Learning robust global representations by penalizing local predictive power. *Advances in Neural Information Processing Systems*, 32, 2019.

[246] Hanrui Wang, Zhekai Zhang, and Song Han. Spatten: Efficient sparse attention architecture with cascade token and head pruning. In *HPCA*, 2021.

[247] Sinong Wang, Belinda Z Li, Madian Khabsa, Han Fang, and Hao Ma. Linformer: Self-attention with linear complexity. *arXiv preprint arXiv:2006.04768*, 2020.

[248] Wenhai Wang, Enze Xie, Xiang Li, Deng-Ping Fan, Kaitao Song, Ding Liang, Tong Lu, Ping Luo, and Ling Shao. Pyramid vision transformer: A versatile backbone for dense prediction without convolutions. In *ICCV*, 2021.

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

[250] Xinlong Wang, Rufeng Zhang, Chunhua Shen, Tao Kong, and Lei Li. Dense contrastive learning for self-supervised visual pre-training. In *Conference on Computer Vision and Pattern Recognition*, pages 3024–3033, 2021.

[251] Yikai Wang, Chengming Xu, Chen Liu, Li Zhang, and Yanwei Fu. Instance credibility inference for few-shot learning. *CVPR*, 2020.

[252] Yaqing Wang, Quanming Yao, James T Kwok, and Lionel M Ni. Generalizing from a few examples: A survey on few-shot learning. *ACM computing surveys (csur)*, 53(3):1–34, 2020.

[253] Yulong Wang, Xiaolu Zhang, Lingxi Xie, Jun Zhou, Hang Su, Bo Zhang, and Xiaolin Hu. Pruning from scratch. *Proceedings of AAAI*, 2020.

[254] Andrew R Webb and David Lowe. The optimised internal representation of multilayer classifier networks performs nonlinear discriminant analysis. *Neural Networks*, 3(4):367–375, 1990.

[255] Chen Wei, Haoqi Fan, Saining Xie, Chao-Yuan Wu, Alan Yuille, and Christoph Feichtenhofer. Masked feature prediction for self-supervised visual pre-training. *arXiv preprint arXiv:2112.09133*, 2021.

[256] Thomas Wolf, Lysandre Debut, Victor Sanh, Julien Chaumond, Clement Delangue, Anthony Moi, Pierric Cistac, Tim Rault, Rémi Louf, Morgan Funtowicz, et al. Huggingface's transformers: State-of-the-art natural language processing. *arXiv preprint arXiv:1910.03771*, 2019.

[257] Sanghyun Woo, Jongchan Park, Joon-Young Lee, and In So Kweon. Cbam: Convolutional block attention module. In *Proceedings of the European conference on computer vision (ECCV)*, pages 3–19, 2018.

[258] Zhirong Wu, Shuran Song, Aditya Khosla, Fisher Yu, Linguang Zhang, Xiaoou Tang, and Jianxiong Xiao. 3d shapenets: A deep representation for volumetric shapes. *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 1912–1920, 2015.

[259] Zhirong Wu, Yuanjun Xiong, Stella X Yu, and Dahua Lin. Unsupervised feature learning via non-parametric instance discrimination. In *Conference on Computer Vision and Pattern Recognition*, pages 3733–3742, 2018.

[260] Tete Xiao, Yingcheng Liu, Bolei Zhou, Yuning Jiang, and Jian Sun. Unified perceptual parsing for scene understanding. In *Proceedings of the European Conference on Computer Vision (ECCV)*, pages 418–434, 2018.

[261] Zhenda Xie, Yutong Lin, Zheng Zhang, Yue Cao, Stephen Lin, and Han Hu. Propagate yourself: Exploring pixel-level consistency for unsupervised visual representation learning. In *Conference on Computer Vision and Pattern Recognition*, pages 16684–16693, 2021.

[262] Zhenda Xie, Zheng Zhang, Yue Cao, Yutong Lin, Jianmin Bao, Zhuliang Yao, Qi Dai, and Han Hu. Simmim: A simple framework for masked image modeling. *arXiv preprint arXiv:2111.09886*, 2021.

[263] Yunyang Xiong, Zhanpeng Zeng, Rudrasis Chakraborty, Mingxing Tan, Glenn Fung, Yin Li, and Vikas Singh. Nystromformer: A nystrom-based algorithm for approximating self-attention. In *AAAI*, 2021.

[264] Yifan Xu, Zhijie Zhang, Mengdan Zhang, Kekai Sheng, Ke Li, Weiming Dong, Liqing Zhang, Changsheng Xu, and Xing Sun. Evo-vit: Slow-fast token evolution for dynamic vision transformer. *arXiv preprint arXiv:2108.01390*, 2021.

[265] Makoto Yamada, Wittawat Jitkrittum, Leonid Sigal, Eric P Xing, and Masashi Sugiyama. High-dimensional feature selection by feature-wise kernelized lasso. *Neural computation*, 26(1):185–207, 2014.

[266] Makoto Yamada, Jiliang Tang, Jose Lugo-Martinez, Ermin Hodzic, Raunak Shrestha, Avishek Saha, Hua Ouyang, Dawei Yin, Hiroshi Mamitsuka, Cenk Sahinalp, et al. Ultra high-dimensional nonlinear feature selection for big biological data. *IEEE Transactions on Knowledge and Data Engineering*, 30(7):1352–1365, 2018.

[267] Brandon Yang, Gabriel Bender, Quoc V Le, and Jiquan Ngiam. Condconv: Conditionally parameterized convolutions for efficient inference. *arXiv preprint arXiv:1904.04971*, 2019.

[268] H Yang and John Moody. Feature selection based on joint mutual information. In *Proceedings of international ICSC symposium on advances in intelligent data analysis*, pages 22–25. Citeseer, 1999.

[269] Huanrui Yang, Hongxu Yin, Pavlo Molchanov, Hai Li, and Jan Kautz. Nvit: Vision transformer compression and parameter redistribution. *arXiv preprint arXiv:2110.04869*, 2021.

[270] Tien-Ju Yang, Andrew Howard, Bo Chen, Xiao Zhang, Alec Go, Mark Sandler, Vivienne Sze, and Hartwig Adam. Netadapt: Platform-aware neural network adaptation for mobile applications. *Proceedings of European Conference on Computer Vision*, pages 285–300, 2018.

[271] Lewei Yao, Runhui Huang, Lu Hou, Guansong Lu, Minzhe Niu, Hang Xu, Xiaodan Liang, Zhenguo Li, Xin Jiang, and Chunjing Xu. Filip: Fine-grained interactive language-image pre-training. *arXiv preprint arXiv:2111.07783*, 2021.

[272] Han-Jia Ye, Hexiang Hu, De-Chuan Zhan, and Fei Sha. Few-shot learning via embedding adaptation with set-to-set functions. *CVPR*, 2020.

[273] Mao Ye, Chengyue Gong, Lizhen Nie, Denny Zhou, Adam Klivans, and Qiang Liu. Good subnetworks provably exist: Pruning via greedy forward selection. *Proceedings of International Conference on Machine Learning*, pages 10820–10830, 2020.

[274] Li Yi, Vladimir G Kim, Duygu Ceylan, I-Chao Shen, Mengyan Yan, Hao Su, Cewu Lu, Qixing Huang, Alla Sheffer, and Leonidas Guibas. A scalable active framework for region annotation in 3d shape collections. *ACM Transactions on Graphics*, 35(6):1–12, 2016.

[275] Haoran You, Chaojian Li, Pengfei Xu, Yonggan Fu, Yue Wang, Xiaohan Chen, Richard G Baraniuk, Zhangyang Wang, and Yingyan Lin. Drawing early-bird tickets: Towards more efficient training of deep networks. *Proceedings of International Conference on Learning Representations*, 2020.

[276] Zhonghui You, Kun Yan, Jinmian Ye, Meng Ma, and Ping Wang. Gate decorator: Global filter pruning method for accelerating deep convolutional neural networks. *Proceedings of Advances in Neural Information Processing Systems*, 2019.

[277] Hao Yu and Jianxin Wu. A unified pruning framework for vision transformers. *arXiv preprint arXiv:2111.15127*, 2021.

[278] Ruichi Yu, Ang Li, Chun-Fu Chen, Jui-Hsin Lai, Vlad I Morariu, Xintong Han, Mingfei Gao, Ching-Yung Lin, and Larry S Davis. Nisp: Pruning networks using neuron importance score propagation. *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 9194–9203, 2018.

[279] Shixing Yu, Tianlong Chen, Jiayi Shen, Huan Yuan, Jianchao Tan, Sen Yang, Ji Liu, and Zhangyang Wang. Unified visual transformer compression. In *ICLR*, 2022.

[280] Tiezheng Yu, Zihan Liu, and Pascale Fung. Adaptsum: Towards low-resource domain adaptation for abstractive summarization. In *Proceedings of the 2021 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 5892–5904, 2021.

[281] Li Yuan, Yunpeng Chen, Tao Wang, Weihao Yu, Yujun Shi, Zi-Hang Jiang, Francis E.H. Tay, Jiashi Feng, and Shuicheng Yan. Tokens-to-token vit: Training vision transformers from scratch on imagenet. In *ICCV*, 2021.

[282] Xin Yuan, Zhe Lin, Jason Kuen, Jianming Zhang, Yilin Wang, Michael Maire, Ajinkya Kale, and Baldo Faieta. Multimodal contrastive training for visual representation learning. In *Conference on Computer Vision and Pattern Recognition*, pages 6995–7004, 2021.

[283] Xin Yuan, Pedro Henrique Pamplona Savarese, and Michael Maire. Growing efficient deep networks by structured continuous sparsification. *Proceedings of International Conference on Learning Representations*, 2021.

[284] Sergey Zagoruyko and Nikos Komodakis. Wide residual networks. *arXiv preprint arXiv:1605.07146*, 2016.

[285] Elad Ben Zaken, Shauli Ravfogel, and Yoav Goldberg. Bitfit: Simple parameter-efficient fine-tuning for transformer-based masked language-models. *arXiv preprint arXiv:2106.10199*, 2021.

[286] Jure Zbontar, Li Jing, Ishan Misra, Yann LeCun, and Stéphane Deny. Barlow twins: Self-supervised learning via redundancy reduction. In *International Conference on Machine Learning*, pages 12310–12320, 2021.

[287] Roman Zeyde, Michael Elad, and Matan Protter. On single image scale-up using sparse-representations. In *International conference on curves and surfaces*, pages 711–730. Springer, 2010.

[288] Chi Zhang, Yujun Cai, Guosheng Lin, and Chunhua Shen. Deepemd: Few-shot image classification with differentiable earth mover's distance and structured classifiers. *CVPR*, 2020.

[289] Kai Zhang, Wangmeng Zuo, and Lei Zhang. Learning a single convolutional super-resolution network for multiple degradations. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 3262–3271, 2018.

[290] Richard Zhang, Phillip Isola, and Alexei A Efros. Colorful image colorization. In *European Conference on Computer Vision*, pages 649–666. Springer, 2016.

[291] Saizheng Zhang, Emily Dinan, Jack Urbanek, Arthur Szlam, Douwe Kiela, and Jason Weston. Personalizing dialogue agents: I have a dog, do you have pets too? In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 2204–2213, 2018.

[292] Tianyun Zhang, Shaokai Ye, Xiaoyu Feng, Xiaolong Ma, Kaiqi Zhang, Zhengang Li, Jian Tang, Sijia Liu, Xue Lin, Yongpan Liu, et al. Structadmm: Achieving ultrahigh efficiency in structured pruning for dnns. *IEEE Transactions on Neural Networks and Learning Systems (TNNLS)*, 2021.

[293] Xiangyu Zhang, Xinyu Zhou, Mengxiao Lin, and Jian Sun. Shufflenet: An extremely efficient convolutional neural network for mobile devices. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 6848–6856, 2018.

[294] Yulun Zhang, Yapeng Tian, Yu Kong, Bineng Zhong, and Yun Fu. Residual dense network for image super-resolution. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 2472–2481, 2018.

[295] Hengshuang Zhao, Jianping Shi, Xiaojuan Qi, Xiaogang Wang, and Jiaya Jia. Pyramid scene parsing network. *CVPR*, 2017.

[296] Mengjie Zhao, Tao Lin, Fei Mi, Martin Jaggi, and Hinrich Schütze. Masking as an efficient alternative to finetuning for pretrained language models. *arXiv preprint arXiv:2004.12406*, 2020.

[297] Yuekai Zhao, Li Dong, Yelong Shen, Zhihua Zhang, Furu Wei, and Weizhu Chen. Memory-efficient differentiable transformer architecture search. In *Findings of the Association for Computational Linguistics: ACL-IJCNLP 2021*, pages 4254–4264, 2021.

[298] Bolei Zhou, Hang Zhao, Xavier Puig, Sanja Fidler, Adela Barriuso, and Antonio Torralba. Scene parsing through ade20k dataset. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2017.

[299] Bolei Zhou, Hang Zhao, Xavier Puig, Sanja Fidler, Adela Barriuso, and Antonio Torralba. Scene parsing through ade20k dataset. In *Conference on Computer Vision and Pattern Recognition*, 2017.

[300] Jinghao Zhou, Chen Wei, Huiyu Wang, Wei Shen, Cihang Xie, Alan Yuille, and Tao Kong. ibot: Image bert pre-training with online tokenizer. *arXiv preprint arXiv:2111.07832*, 2021.

[301] Mingjian Zhu, Kai Han, Yehui Tang, and Yunhe Wang. Visual transformer pruning. In *KDD*, 2021.

[302] Tao Zhuang, Zhixuan Zhang, Yuheng Huang, Xiaoyi Zeng, Kai Shuang, and Xiang Li. Neuron-level structured pruning using polarization regularizer. *Proceedings of Advances in Neural Information Processing Systems*, 2020.

[303] Zhuangwei Zhuang, Mingkui Tan, Bohan Zhuang, Jing Liu, Yong Guo, Qingyao Wu, Junzhou Huang, and Jinhui Zhu. Discrimination-aware channel pruning for deep neural networks. *Proceedings of Advances in Neural Information Processing Systems*, pages 883–894, 2018.

[304] Imtiaz Ziko, Jose Dolz, Eric Granger, and Ismail Ben Ayed. Laplacian regularized few-shot learning. *ICML*, 2020.

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

[306] Barret Zoph, Vijay Vasudevan, Jonathon Shlens, and Quoc V Le. Learning transferable architectures for scalable image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 8697–8710, 2018.

# Chapter 9

# Appendix

## 9.1 Convergence analysis of CHEX

### 9.1.1 Problem formulation

Training deep neural networks can be formulated into minimizing the following problem:

$$\min_{\mathbf{W}\in\mathbb{R}^d} \quad F(\mathbf{W}) = \mathbb{E}_{x\sim\mathcal{D}}[f(\mathbf{W};x)] \tag{9.1}$$

where $\mathbf{W} \in \mathbb{R}^d$ is the model parameter to be learned, and $f(\mathbf{W};x)$ is a non-convex loss function in terms of $\mathbf{W}$. The random variable $x$ denotes the data samples that follow the distribution $\mathcal{D}$. $\mathbb{E}_{x\sim\mathcal{D}}[\cdot]$ denotes the expectation over the random variable $x$.

### 9.1.2 Notation

We clarify several notions to facilitate the convergence analysis.

- $\mathbf{W}_t$ denotes the complete model parameter at the $t$-th iteration.
- $m_t \in \mathbb{R}^d$ is a mask vector at the $t$-th iteration.
- $x_t$ is the data sampled at the $t$-th iteration.

### 9.1.3 Algorithm formulation

With notations introduced in the above subsection, the proposed CHEX method can be generalized in a way that is more friendly for convergence analysis, see Algorithm 8.

**Algorithm 8:** CHEX (A math-friendly version)

**Input**: Initialize $\mathbf{W}_0$ and $m_0$ randomly ;
**for** *iteration* $t = 0, 1, \cdots, T$ **do**
  Sample data $x_t$ from distribution $\mathcal{D}$ ;
  Generate a mask $m_t$ following some rules ;
  Update $\mathbf{W}_{t+1} = \mathbf{W}_t - \eta \nabla f(\mathbf{W}_t \odot m_t; x_t) \odot m_t$
**Output**: The pruned model parameter $\mathbf{W}_T \odot m_T$;

**Remark 3.** *Note that Algorithm 8 is quite general. It does not specify the rule to generate the mask $m_t$. This implies the convergence analysis established in Sec. 9.1.5 does not rely on what specific $m_t$ is utilized. In fact, it is even allowed for $m_t$ to remain unchanged during some period.*

### 9.1.4 Assumptions

We now introduce several assumptions on the loss function and the gradient noise that are standard in the literature.

**Assumption 1** (Smoothness)**.** *We assume $F(\mathbf{W})$ is L-smooth, i.e., it holds for any $\mathbf{W}_1, \mathbf{W}_2 \in \mathbb{R}^d$ that*

$$\|\nabla F(\mathbf{W}_1) - \nabla F(\mathbf{W}_2)\| \leq L\|\mathbf{W}_1 - \mathbf{W}_2\| \tag{9.2}$$

*or equivalently,*

$$\begin{aligned} &F(\mathbf{W}_1) - F(\mathbf{W}_2) \\ &\leq \langle \nabla F(\mathbf{W}_2), \mathbf{W}_1 - \mathbf{W}_2 \rangle + \frac{L}{2}\|\mathbf{W}_1 - \mathbf{W}_2\|^2 \end{aligned} \tag{9.3}$$

**Assumption 2** (Gradient noise)**.** *We assume*

$$\mathbb{E}\{\nabla f(\mathbf{W}; x_t)\} = \nabla F(\mathbf{W}), \tag{9.4}$$

$$\mathbb{E}\|\nabla f(\mathbf{W}; x_t) - \nabla F(\mathbf{W})\|^2 \leq \sigma^2, \tag{9.5}$$

*where $\sigma > 0$ is a constent. Moreover, we assume the data sample $x_t$ is independent of each other for any $t$.*

This assumption implies that the stochastic filter-gradient is unbiased and has bounded variance.

**Assumption 3** (Mask-incurred Error)**.** *It holds for any $\mathbf{W}$ and $m_t$ that*

$$\|\mathbf{W} - \mathbf{W} \odot m_t\|^2 \leq \delta^2 \|\mathbf{W}\|^2 \tag{9.6}$$

$$\|\nabla F(\mathbf{W}) - \nabla F(\mathbf{W}) \odot m_t\|^2 \leq \zeta^2 \|\nabla F(\mathbf{W})\|^2 \tag{9.7}$$

*where constants* $\delta \in [0,1]$ *and* $\zeta \in [0,1]$.

With (9.7), we have

$$\begin{aligned} & \zeta\|\nabla F(\mathbf{W})\| \\ \geq & \|\nabla F(\mathbf{W}) - \nabla F(\mathbf{W}) \odot m_t\| \\ \geq & \|\nabla F(\mathbf{W})\| - \|\nabla F(\mathbf{W}) \odot m_t\| \end{aligned} \tag{9.8}$$

which implies

$$\|\nabla F(\mathbf{W}) \odot m_t\|^2 \geq (1-\zeta)^2 \|\nabla F(\mathbf{W})\|^2 \tag{9.9}$$

for any $\mathbf{W}$ and $m_t$.

### 9.1.5 Convergence analysis

Now we are ready to establish the convergence property.

**Theorem 5** (Convergence property)**.** *Under Assumptions 1 – 3, if learning rate* $\eta = \frac{\sqrt{2C_0}}{\sigma\sqrt{L(T+1)}}$ *in which* $C_0 = \mathbb{E}[F(\mathbf{W}_0)]$*, it holds that*

$$\begin{aligned} & \frac{1}{T+1}\sum_{t=0}^{T} \mathbb{E}\|\nabla F(\mathbf{W}_t \odot m_t)\|^2 \\ \leq & \frac{4\sigma\sqrt{LC_0}}{(1-\zeta)^2\sqrt{T+1}} + \frac{2L^2\delta^2}{(T+1)(1-\zeta)^2}\sum_{t=0}^{T}\mathbb{E}\|\mathbf{W}_t\|^2 \end{aligned} \tag{9.10}$$

*Proof.* With inequality (9.3) and Line 6 in Algorithm 8, it holds that

$$\begin{aligned} & F(\mathbf{W}_{t+1}) - F(\mathbf{W}_t) \\ \leq & -\eta\langle \nabla F(\mathbf{W}_t), [\nabla f(\mathbf{W}_t \odot m_t; x_t)] \odot m_t\rangle \\ & + \frac{\eta^2 L}{2}\|[\nabla f(\mathbf{W}_t \odot m_t; x_t)] \odot m_t\|^2 \end{aligned} \tag{9.11}$$

With Assumption 2, it holds that

$$\begin{aligned} & \mathbb{E}\langle \nabla F(\mathbf{W}_t), \nabla f(\mathbf{W}_t \odot m_t; x_t) \odot m_t\rangle \\ \overset{(9.4)}{=} & \mathbb{E}\langle \nabla F(\mathbf{W}_t), \nabla F(\mathbf{W}_t \odot m_t) \odot m_t\rangle \\ = & \mathbb{E}\langle \nabla F(\mathbf{W}_t) \odot m_t, \nabla F(\mathbf{W}_t \odot m_t) \odot m_t\rangle \\ = & \frac{1}{2}\mathbb{E}\|\nabla F(\mathbf{W}_t) \odot m_t\|^2 + \frac{1}{2}\mathbb{E}\|\nabla F(\mathbf{W}_t \odot m_t) \odot m_t\|^2 \\ & - \frac{1}{2}\mathbb{E}\|\nabla F(\mathbf{W}_t) \odot m_t - \nabla F(\mathbf{W}_t \odot m_t) \odot m_t\|^2 \\ \geq & \frac{1}{2}\mathbb{E}\|\nabla F(\mathbf{W}_t) \odot m_t\|^2 + \frac{1}{2}\mathbb{E}\|\nabla F(\mathbf{W}_t \odot m_t) \odot m_t\|^2 \end{aligned}$$

$$
\begin{aligned}
&- \frac{1}{2}\mathbb{E}\|\nabla F(\mathbf{W}_t) - \nabla F(\mathbf{W}_t \odot m_t)\|^2 \\
\overset{(9.2)}{\geq} &\frac{1}{2}\mathbb{E}\|\nabla F(\mathbf{W}_t) \odot m_t\|^2 + \frac{1}{2}\mathbb{E}\|\nabla F(\mathbf{W}_t \odot m_t) \odot m_t\|^2 \\
&- \frac{L^2}{2}\mathbb{E}\|\mathbf{W}_t - \mathbf{W}_t \odot m_t\|^2 \\
\overset{(9.6)}{\geq} &\frac{1}{2}\mathbb{E}\|\nabla F(\mathbf{W}_t) \odot m_t\|^2 + \frac{1}{2}\mathbb{E}\|\nabla F(\mathbf{W}_t \odot m_t) \odot m_t\|^2 \\
&- \frac{L^2\delta^2}{2}\mathbb{E}\|\mathbf{W}_t\|^2 \\
\overset{(9.9)}{\geq} &\frac{(1-\zeta)^2}{2}\mathbb{E}\|\nabla F(\mathbf{W}_t)\|^2 + \frac{(1-\zeta)^2}{2}\mathbb{E}\|\nabla F(\mathbf{W}_t \odot m_t)\|^2 \\
&- \frac{L^2\delta^2}{2}\mathbb{E}\|\mathbf{W}_t\|^2
\end{aligned} \tag{9.12}
$$

Furthermore, with Assumption 2, it holds that

$$
\begin{aligned}
&\mathbb{E}\|[\nabla f(\mathbf{W}_t \odot m_t; x_t)] \odot m_t\|^2 \\
\leq\ &\mathbb{E}\|\nabla f(\mathbf{W}_t \odot m_t; x_t)\|^2 \\
\overset{(9.5)}{\leq}\ &\mathbb{E}\|\nabla F(\mathbf{W}_t \odot m_t)\|^2 + \sigma^2
\end{aligned} \tag{9.13}
$$

Substituting (9.12) and (9.13) into (9.11), we achieve

$$
\begin{aligned}
&\mathbb{E}[F(\mathbf{W}_{t+1}) - F(\mathbf{W}_t)] \\
\leq\ &- \frac{\eta(1-\zeta)^2}{2}\mathbb{E}\|\nabla F(\mathbf{W}_t)\|^2 \\
&- \frac{\eta(1-\zeta)^2}{2}\mathbb{E}\|\nabla F(\mathbf{W}_t \odot m_t)\|^2 \\
&+ \frac{\eta L^2\delta^2}{2}\mathbb{E}\|\mathbf{W}_t\|^2 \\
&+ \frac{\eta^2 L}{2}\mathbb{E}\|\nabla F(\mathbf{W}_t \odot m_t)\|^2 + \frac{\eta^2 L\sigma^2}{2} \\
\leq\ &- \frac{\eta(1-\zeta)^2}{4}\mathbb{E}\|\nabla F(\mathbf{W}_t)\|^2 \\
&- \frac{\eta(1-\zeta)^2}{4}\mathbb{E}\|\nabla F(\mathbf{W}_t \odot m_t)\|^2 \\
&+ \frac{\eta L^2\delta^2}{2}\mathbb{E}\|\mathbf{W}_t\|^2 + \frac{\eta^2 L\sigma^2}{2}
\end{aligned} \tag{9.14}
$$

where the last inequality holds by setting $\eta \leq \frac{(1-\zeta)^2}{2L}$. The above inequality will lead to

$$
\begin{aligned}
&\frac{1}{T+1}\sum_{t=0}^{T}\mathbb{E}\|\nabla F(\mathbf{W}_t)\|^2 + \mathbb{E}\|\nabla F(\mathbf{W}_t \odot m_t)\|^2 \\
\leq\ &\frac{4}{\eta(1-\zeta)^2(T+1)}\mathbb{E}[F(\mathbf{W}_0)] \\
&+ \frac{2L^2\delta^2}{(1-\zeta)^2(T+1)}\sum_{t=0}^{T}\mathbb{E}\|\mathbf{W}_t\|^2 + \frac{2\eta L\sigma^2}{(1-\zeta)^2}
\end{aligned}
$$

$$\leq \frac{4\sigma\sqrt{LC_0}}{(1-\zeta)^2\sqrt{T+1}} + \frac{2L^2\delta^2}{(T+1)(1-\zeta)^2}\sum_{t=0}^{T}\mathbb{E}\|\mathbf{W}_t\|^2 \tag{9.15}$$

where $C_0 = \mathbb{E}[F(\mathbf{W}_0)]$ and the last equality holds when $\eta = \frac{\sqrt{2C_0}}{\sigma\sqrt{L(T+1)}}$. The above inequality will lead to (9.10). □

## 9.2 List of software

- `https://github.com/zejiangh/DI-iterative-pruning`:

  An iterative channel pruning algorithm is implemented to compress convolution neural networks. The implementation includes (1) a mathematically guided pruning criterion, discriminant information, to evaluate the channel importance very efficiently, (2) an automated structural learning method that can evolve the model structure to satisfy user-defined compression ratios.

- `https://github.com/zejiangh/Filter-GaP`:

  A novel Channel Exploration methodology, dubbed as CHEX, is implemented to repeatedly prune and regrow the channels throughout the training process, and evolve a random sub-network to the final optimal one. The implementation includes (1) channel pruning by the column subset selection, (2) orthogonality based importance sampling to regrow channels, (3) dynamically re-allocating the number of channels across all the layers under a global channel sparsity constraint.

- `https://github.com/zejiangh/Multi-dimensional-vit-compression`:

  A multi-dimensional ViT compression method is implemented to compress attention head, neuron and tokens jointly in vision transformers. The implementation includes (1) a statistical dependence based pruning criterion that can prune different dimensions, (2) the expected improvement algorithm to solve the joint optimization for searching the optimal compression policy.

- `https://github.com/zejiangh/PEDConv`:

  A parameter efficient dynamic convolution operator, dubbed as PEDConv, is implemented which generates input-specific weights on-the-fly to achieve dynamic neural network. The implementation includes (1) a tensor decomposition based input-dependent reparameterization, (2) application of PEDConv to different CNN backbones.

- `https://github.com/zejiangh/meta-learning-the-difference`:

A meta-learning framework is implemented to prepare large language models for data- and parameter-efficient adaptation. The implementation includes (1) dynamic low-rank reparameterization for efficient weight adaptation in the inner-loop of meta-learning, (2) task-adaptive model structure by learning architecture controller in the outer-loop, (2) scripts for experiments on few-shot dialogue completion, low-resource abstractive summarization, and multi-domain language modeling.

- `https://github.com/zejiangh/Semi-FSL`:

  A semi-supervised few-shot learning framework is implemented to exploit unlabeled data for improving few-shot performance. The implementation includes (1) an unsupervised dependency maximization loss function based on the Hilbert-Schmidt norm of the cross-covariance operator, (2) an iterative outlier removal method based on discriminant information to evaluate the credibility of each pseudo-labeled example and select the credible ones to augment the labeled set.

- `https://github.com/zejiangh/MILAN`:

  A cross-modal reconstruction based self-supervised learning method, called MILAN, is implemented. The implementation includes: (1) reconstructing the image features with substantial semantic signals that are obtained using cross-modal text-image data, (2) a efficient prompting decoder architecture, (3) semantic aware mask sampling mechanism, (4) pretrained ViT models that achieve the state-of-the-art 87.5% top-1 accuracy on ImageNet-1K.