ABSTRACT

Title of Dissertation: PRUNING FOR EFFICIENT DEEP LEARNING: FROM CNNS TO GENERATIVE MODELS

Alireza Ganjdanesh
Doctor of Philosophy, 2025

Dissertation Directed by: Professor Heng Huang
Department of Computer Science

Deep learning models have shown remarkable success in visual recognition and generative modeling tasks in computer vision in the last decade. A general trend is that their performance improves with an increase in the size of their training data, model capacity, and training iterations on modern hardware. However, the increase in model size naturally leads to higher computational complexity and memory footprint, thereby necessitating high-end hardware for their deployment. This trade-off prevents the deployment of deep learning models in resource-constrained environments such as robotic applications, mobile phones, and edge devices employed in the Artificial Internet of Things (AIoT). In addition, private companies and organizations have to spend significant resources on cloud services to serve deep models for their customers. In this dissertation, we develop model pruning and Neural Architecture Search (NAS) methods to improve the inference efficiency of deep learning models for visual recognition and generative modeling applications. We design our

methods to be tailored to the unique characteristics of each model and its task.

In the first part, we present model pruning and efficient NAS methods for Convolutional Neural Network (CNN) classifiers. We start by proposing a pruning method that leverages interpretations of a pretrained model's decisions to prune its redundant structures. Then, we provide an efficient NAS method to learn kernel sizes of a CNN model using their training dataset and given a parameter budget for the model, enabling designing efficient CNNs customized for their target application. Finally, we develop a framework for simultaneous pretraining and pruning of CNNs, which combines the first two stage of the pretrain-prune-finetune pipeline commonly used in model pruning and reduces its complexity.

In the second part, we propose model pruning methods for visual generative models. First, we present a pruning method for conditional Generative Adversarial Networks (GANs) in which we prune the generator and discriminator models in a collaborative manner. We then address the inference efficiency of diffusion models by proposing a method that prunes a pretrained diffusion model into a mixture of efficient experts, each handling a separate part of the denoising process. Finally, we develop an adaptive prompt-tailored pruning method for modern text-to-image diffusion models. It prunes a pretrained model like Stable Diffusion into a mixture of efficient experts such that each expert specializes in certain type of input prompts.

PRUNING FOR EFFICIENT DEEP LEARNING: FROM CNNS TO GENERATIVE MODELS

by

Alireza Ganjdanesh

Dissertation submitted to the Faculty of the Graduate School of the
University of Maryland, College Park in partial fulfillment
of the requirements for the degree of
Doctor of Philosophy
2025

Advisory Committee:
Professor Heng Huang, Chair/Advisor
Professor Abhinav Shrivastava
Professor Furong Huang
Professor Tianyi Zhou
Professor Shuvra S. Bhattacharyya, Dean's Representative

To the dearest people in my life, Melina, Kasra, Ayli, Soheila, and Yousef.

Acknowledgments

I start by expressing my appreciation to my Ph.D. advisor, Dr. Heng Huang, for his guidance and constant support during my Ph.D. journey. He provided me with the invaluable opportunity to enter the exciting field of deep learning research, which I will always be grateful for. I am especially thankful for his patience and encouragement during the early stages of my Ph.D., as he guided me in building the foundational knowledge and skills necessary to conduct meaningful research. His mentorship has been instrumental in shaping my growth as a researcher and inspiring me to push the boundaries of my work.

I extend my gratitude to my committee members, Dr. Abhinav Shrivastava, Dr. Furong Huang, Dr. Tianyi Zhou, and Dr. Shuvra S. Bhattacharyya, for their generosity in serving on my dissertation committee. Their insightful feedback and constructive comments have been invaluable in refining my research and strengthening my dissertation.

I express my heartfelt appreciation to my collaborators, whose invaluable contributions have been instrumental to the success of my research. First and foremost, I want to thank Dr. Shangqian Gao, who has been both a mentor and a guiding figure, akin to an older brother in my professional journey. He introduced me to the fascinating field of deep learning efficiency and was an outstanding collaborator on most of the projects presented in this dissertation. I am deeply thankful for the hard work we shared, the countless hours dedicated to brainstorming ideas, implementing them, conducting experiments, and

the late-night meetings essential to achieving these milestones together. I also thank Reza Shirkavand for his dedication during our collaboration in the final year of my Ph.D. His efforts were instrumental in bringing our project to fruition.

I am profoundly thankful to Yan Kang for giving me with the opportunity to join Adobe Research as an intern in Summer 2023. This internship marked my first experience working at a major technology company, where I was fortunate to have Dr. Yuchen Liu, Dr. Richard Zhang, and Dr. Zhe Lin, along with Yan, as my mentors. Through this experience, I also had the privilege of meeting my current manager, Dr. Soheil Darabi, which led to securing my first full-time position with the brilliant Adobe Firefly team.

I would also like to thank my former collaborators, Dr. Wei Chen, Dr. Kamran Ghasedi, Dr. Liang Zhan, and Jipeng Zhang. Their expertise and dedication were invaluable in completing the projects during the early stages of my Ph.D.

I thank my former and current lab-mates at Huang Lab at the University of Pittsburgh and the University of Maryland, College Park. I was lucky to be a part of such a talented group of researchers including: Feihu Huang, Bin Gu, Lei Luo, Kamran Ghasedi, Hongchang Gao, Xiaoqian Wang, Zhouyuan Huo, Shangqian Gao, Yanfu Zhang, Runxue Bao, Wenhan Xian, Chao Li, An Xu, Xiaotian Dou, Guodong Liu, Peiran Yu, Zhenyi Wang, Junfeng Guo, Zhengmian Hu, Junyi Li, Yihan Wu, Xidong Wu, Reza Shirkavand, Hirad Alipanah, Lichang Chen, Yanshuo Chen, Ruibo Chen, Chenxi Liu, and Tianyi Xiong. I had my first hot pot at our group's lunch gathering, and since then, my wife and I have had multiple Mediterranean-style hot pot meals together. I also will not forget the fun times we had with Ruibo Chen, Chenxi Liu, and Tianyi Xiong at the Yahentamitsi dining hall at UMD. Thank you all for the great memories.

I thank Dr. Natasa Miskov-Zivanov, Dr. Azime Can-Cimino, and Dr. Steven Jacobs, under whom I had the privilege of serving as a teaching assistant at the University of Pittsburgh. They entrusted me with the opportunity to lead several lecture sessions, allowing me to experience teaching in a language other than my mother tongue. These unique experiences were both challenging and rewarding, and I am grateful for their trust in me.

I also thank Handa Ding, who was a friend and a colleague during my time as a teaching assistant for ECE Analytical Methods under Natasa. It was the first semester of my Ph.D., and I was having a hard time balancing coursework and research while adapting to a new environment. Handa's support during that period was truly invaluable. I truly appreciate his willingness to step in and cover for me when I needed it most. I will always cherish the enjoyable moments we shared while grading exams with Natasa and Handa.

Next, I would like to thank my friends who made my Ph.D. journey not only tolerable but also memorable during my years in Pittsburgh and College Park. I am particularly grateful to Kazem Meidani, my best friend and roommate during my time in Pittsburgh. He stood by me through all the ups and downs, particularly during the challenging COVID-19 pandemic lockdowns, which began just six months after we arrived in the United States. His unwavering support during such a difficult time was invaluable, and I could not have remained sane without him.

I extend my heartfelt thanks to Kamran Ghasedi and Najmeh Sadoughi, who were like an older brother and sister to me and Kazem during our first year of the Ph.D. program. Their willingness to help and the wonderful memories we created together in Pittsburgh mean so much to me. I feel fortunate to still be in touch with them in Seattle.

I sincerely thank Mohsen Tabrizi and Amirreza Hashemi for their unwavering willingness to selflessly support the younger generation of students in the Iranian community in Pittsburgh. Their humility and kindness have left a lasting impression on me. The world would undoubtedly be a better place with more individuals like you. Thank you both for your efforts and support during my time in Pittsburgh.

I am also grateful for the fun times and cherished memories shared with my other friends in Pittsburgh: Parshin Shojaee, Soroosh Shafieezadeh Abadeh, Hirad Alipanah, Reza Shirkavand, Maryam Hakimzadeh, Fahimeh Dehghan, Yasin Karimi, Sanaz Saadatifar, Saba Dadsetan, Alireza Golestaneh, Parand Akbari, Masoud Zamani, Tahere Mokhtari, Sina Malakouti, and Mohammad Bakhshalipour.

Additionally, I thank Matin Mortaheb and Maryam Maghsoudi Shaghaghi for the great memories we shared during the last year of my Ph.D in College Park and Washington, DC. Matin, a longtime friend since high school, has been a pillar of support and friendship through all these years, and I deeply appreciate him for always being there.

Finally, I want to express my heartfelt gratitude to my family for their unwavering love and support throughout this journey. Their encouragement has been my anchor, providing me with the strength to persevere through challenges and celebrate successes.

I am especially grateful to my wife, Melina Emily Adrangi, for her constant love, patience, and understanding. She has been my confidant, cheerleader, and partner in every sense, helping me navigate the highs and lows of this journey with grace and resilience. Her sacrifices, from enduring my late working hours to supporting me through stressful times, have not gone unnoticed, and I am deeply thankful for her unwavering commitment to our shared dreams. I am truly fortunate to have her by my side, and this accomplishment

would not have been possible without her.

I express my deepest gratitude to my parents, Yousef and Soheila, whose unwavering encouragement and support have been the foundation of my success. They have always inspired me to work hard, dream big, and pursue my aspirations, including my journey to the United States. From them, I learned the fundamental principles of classical liberalism, free markets, and individual freedom—values that have contributed to the unprecedented prosperity of the United States in human history. My parents have selflessly stood by me through every stage of my life, offering their love and guidance during both triumphs and challenges. I am endlessly grateful to have such incredible parents, and I truly could not have achieved this without you. Thank you for everything.

I thank my siblings, Kasra (Amirreza) and Ayli, for all the joy and laughter you have brought into my life. The fun times we've shared together have been a source of happiness and balance during this journey. I am incredibly proud of both of you and have cherished every moment of our conversations and the opportunity to share my experiences with you. Your support throughout these years has meant the world to me. Thank you for always being there.

I would also like to thank my in-law parents, Farid and Emilia, for their support and kindness over the years. Your constant encouragement and love have been a source of strength for both Melina and me throughout this journey. Your presence has provided us with confidence and reassurance during challenging times, and I am deeply grateful for everything you have done.

To all of you and the others that I met during these years, thank you for supporting me throughout this journey.

Table of Contents

List of Tables

List of Figures

List of Abbreviations

ADAM	Adaptive Moment Estimation
AEM	Amortized Explanation Models
AIoT	Artificial Internet of Things
APTP	Adaptive Prompt-Tailored Pruning
CLIP	Contrastive Language-Image Pre-training
CMMD	CLIP Maximum Mean Discrepancy
CNN	Convolutional Neural Network
DDIM	Denoising Diffusion Implicit Models
DDPM	Denoising Diffusion Probabilistic Models
DNN	Deep Neural Network
DPM	Diffusion Probabilistic Model
EIE	Efficient Inference Engine
ERA	Expert Routing Agent
FID	Fréchet inception distance
FLOPs	Floating-point Operations
GAN	Generative Adversarial Network
GPU	Graphics Processing Unit
GRU	Gated Recurrent Unit
GPT	Generative Pre-trained Transformer
I2IGAN	Image-to-Image Translation Generative Adversarial Network
ISP	Interpretations Steered Pruning
IWFS	Instance-Wise Feature Selection
KD	Knowledge Distillation
KDE	Kernel Density Estimation
KL	Kullback-Leibler Divergence
LDM	Latent Diffusion Model
LHS	Left Hand Side
LIME	Local Interpretable Model-agnostic Explanations
LLM	Large Language Model
LLaVA	Large Language and Vision Assistant
MACs	Multiply Accumulate Operations
MGGC	Manifold Guided GAN Compression
MLP	Multi-layer Perceptron
MoE	Mixture of Experts
NAS	Neural Architecture Search
RBF	Radial Basis Function

ResNet	Residual Network
RHS	Right Hand Side
RL	Reinforcement Learning
SAC	Soft Actor-Critic
SD	Stable Diffusion
SGD	Stochastic Gradient Descent
SHAP	Shapley Additive explanations
STE	Straight-Through Estimator
T2I	Text-to-Image
TPU	Tensor Processing Unit
VAE	Variational Autoencoder

Chapter 1: Introduction

1.1 Motivation

Deep learning methods have achieved unprecedented capabilities for visual recognition and generative modeling tasks in computer vision in the past decade. They have significantly outperformed hand-crafted baselines on traditional image classification (assigning a label to an input image) and object detection (localizing as well as classifying objects within an image) benchmarks. Moreover, deep vision language models like GPT-4 [14], Gemini [15], and LLaVA [16] have made significant strides, enabling them to provide fine-grained text descriptions that not only identify objects in their input images, but also explain their relationships and attributes. In addition, modern deep generative models like DALL-E [17], Stable Diffusion [1], Imagen [18], and Adobe Firefly [19] can generate high-quality, realistic, and detailed images given input text prompts. They have also shown an impressive performance for prompt-based image editing. Thus, deploying deep learning models in various real-world applications like disease diagnosis, autonomous driving, robotics, and content creation is of a great interest. The key ingredient of their success is that they can learn to extract or generate useful patterns for downstream tasks from vast amounts of data.

Empirical trends in the literature indicate that the performance of deep models im-

proves when they benefit from **1)** an increased training sample size, **2)** higher architectural capacity (also known as model size), and **3)** longer training schedules using modern hardwares. These factors have driven the development of large-scale models with billions of parameters, feuling a race among private companies to push the peformance boundaries of deep models by increasing the model size. For instance, the vision language LLaVA V1.5 [20] and QWEN2-VL [21] models are among the top performing models in multimodal vision and language modeling tasks like object localization and visual question answering. LLaVA V1.5 has two variants with 7 and 13 Billion parameters, and QWEN2-VL has three variants of 2, 7, and 72 Billion parameters. Similarly, the Stable Diffusion (SD) models have shown improved performance in image generation and editing tasks from SD-V1 with about 980M to SD-XL [22] and SD-V3 [23] having about 3.5B and 8B parameters, respectively.

However, the increase in model size leads to a natural trade-off between model performance *vs.* computational complexity and memory footprint, thereby making the deployment of these models challenging or even infeasible in various real-world scenarios. On the one hand, organizations and private companies need to invest in expensive GPU or TPU clusters or spend excessive budgets to rent them from cloud providers to serve their models for their customers. On the other hand, directly deploying large-scale models for edge applications like smartphones, robotics, self-driving cars, and the Artificial Internet of Things (AIoT) exhausts their limited memory, computational resources, and battery. Further, these applications demand real-time inference and low-latency responses, which are infeasible to achieve if one runs large-scale models with the limited computational resources available in these applications. Therefore, compressing and reducing the computational burden of deep models while maintaining their performance is crucial before

deploying them in practice.

Model pruning is an efficient and effective technique for compressing trained over-parameterized deep models to improve their inference efficiency. In fact, it has been shown [24, 25] that over-parameterization is beneficial for the optimization and generalization of deep models. Further, one can prune these models to a simple one without significantly affecting their performance, but directly training the pruned model from scratch typically results in worse performance due to the optimization difficulties [25]. Model pruning can be fine-grained, called weight pruning, in which individual weights of a model are removed. It can achieve high compression rates, significantly reducing required storage memory, but weight pruning usually cannot provide inference speed up in practice. The reason is that GPUs and TPUs cannot effectively utilize irregular sparsity patterns, and one needs to use inference engines like EIE [26] to exploit the sparsity to accelerate the inference. Differently, structural pruning removes channels or depth layers of the model, thereby both reducing the model size and accelerating its inference on modern hardware like GPUs without requiring post-processing or special inference libraries. Thus, structural pruning is more practical for real-world applications and has been widely used in different domains.

In this dissertation, we develop structural pruning and architecture search techniques to reduce the memory footprint and improve the inference efficiency of deep visual recognition and generative models, tailored to their unique characteristics and requirements. Therefore, we focus on the following two main research directions that are:

I. **Inference Efficiency of Visual Recognition Models,** where we develop struc-

tural pruning and efficient architecture search methods for Convolutional Neural Networks (CNN) classifiers to achieve compact models given different constraints on the model's computational requirements like the model's number of Multiply-Accumulate operations (MACs) and parameter count. We mainly contribute to three main aspects of this research direction:

First, we approach the pruning problem from a novel perspective and aim to answer whether one can use interpretations of a CNN classifier's decisions to prune it. This is in contrast with the prominent techniques that either focus on the model's outputs or weights to prune it. We discuss in Chapter 2 that existing interpretation techniques are either shown to be indepedent of the model's decisions or are computationally expensive for pruning. Thus, we develop an amortized explanation model tailored for CNN classifiers and employ it in our framework to guide the pruning process.

Second, we introduce an efficient architecture search method to find kernel sizes for CNNs (Chapter 3). Although kernel sizes are crucial design choices for CNNs' performance and efficiency, existing architectures usually contain convolution layers with fixed kernel sizes stacked on top of each other. This design choice is suboptimal since it does not consider the target task. We propose a differentiable architecture search method that determines kernel sizes given a training dataset and parameter budget, securing up to 60× speed up compared to the baseline methods while achieving superior final performance.

Third, we develop a method that reduces the complexity of the pruning process for CNNs. Typically, structural pruning methods perform a three-step process of

pretraining the model, pruning it, and then fine-tuning the pruned model, and each step has its own design choices and hyperparameters. We propose a method that accomplishes the first two steps at the same time using a reinforcement learning agent that learns to determine the optimal structure of the model during the pretraining phase (Chapter 4). By doing so, we improve the efficiency of the pruning process.

II. **Inference Acceleration for Deep Generative Models:** Due to their fundumental differences, existing pruning methods for discriminative models are not directly applicable to generative models, and heuristically stacking them for pruning generative models usually leads to unsatisfactory performance. Therefore, We design pruning techniques for conditional Generative Adversarial Networks (GANs) and modern diffusion models while taking their specific characteristics into account.

First, we address pruning a conditional GAN model. In contrast with previous works that mainly apply distillation, we focus on the learned density structure of a pretrained GAN model as a generative model. Specifically, we propose a structural pruning method that encourages the pruned model to preserve local density structures of the original model on neighborhoods of its learned manifold, resembling the kernel density estimation method. Further, we design a collaborative pruning scheme in which two agents prune both the generator and discriminator by exchanging feedback. Thus, our method can properly maintain the balance between capacities of two models during pruning and alleviate mode collapse during the pruning process, which is a common challenge with baselines.

Second, we leverage the gradual denoising process of modern diffusion models to

prune them into a mixture of efficient experts, each one handling a separate part of the denoising path of the model's sampling process. We propose a dataset-specific approach to cluster the denoising timesteps into intervals using their alignment scores and assign a separate expert to each interval. We introduce a framework in which we prune the experts for the intervals simultaneously, thereby allocating compute resources between them automatically.

Finally, we design a pruning approach tailored for Text-to-image (T2I) diffusion models. Precisely, our method prunes a pretrained T2I diffusion model into a set of efficient experts such that each expert is a specialized model for the prompts routed to it. Our method is the first one that enables using different amounts of compute resources for various prompt types. We do so using a prompt router model that routes input prompts to a set of architecture codes that determine the sub-network of the model to be used. We design a framework in which we train both the prompt router and the architecture codes in an end-to-end manner.

1.2 Dissertation Outline

The structure of this dissertation follows the organization of the research topics discussed in the Sec. 1.1. Accordingly, we divide the dissertation into two parts, addressing inference efficiency of visual recognition models in Part I (Chapter 2, Chapter 3, and Chapter 4) and deep generative models in Part II (Chapter 5, Chapter 6, and Chapter 7).

In general, pruning and architecture search for neural networks can be formulated as a combinatorial "selection" problem in which one should determine whether to preserve or

remove each structural component of the model. In the case of deep over-parameterized models, the search space of the model configurations is discrete, complex, and exponentially large, and the design choices are highly non-trivial. Further, the evaluation of the model configurations is extremely costly as each evaluation requires training the model from scratch. In this dissertation, we design algorithms to tackle the model pruning and architecture search problems efficiently. The general framework of our ideas is that we convert the discrete optimization problem of pruning and architecture search as a continuous one, and by doing so, we leverage gradient-based optimization techniques to efficiently search for compact, high-performing architectures. In more details, we implement our "selection" scheme using a neural network that can be trained end-to-end by employing differentiable selection gates and propagating gradients using the straight-through estimator. We briefly describe our design choices to adapt our framework for different discriminative and generative models in the following.

In the part I, we address inference efficiency of CNNs. CNNs have consistently shown state-of-the-art performance on various computer vision tasks, surpassing Transformers [27, 28] and other counterparts [29, 30]. Therefore, optimizing CNNs' architectures for inference efficiency is practically crucial. In Chapter 2, Chapter 3, and Chapter 4, we develop techniques for pruning and designing CNN architectures to make them efficient for inference.

In Chapter 2, we propose a pruning method that leverages interpretations of a CNN's predictions to guide its pruning process. Existing channel pruning algorithms approach the pruning problem from various perspectives and use different metrics to guide the pruning process. However, these metrics mainly focus on the model's 'outputs' or 'weights' and ne-

glect its 'interpretations' information. To fill in this gap, we propose to address the channel pruning problem from a novel perspective by leveraging the interpretations of a model to steer the pruning process, thereby utilizing information from both inputs and outputs of the model. However, existing interpretation methods cannot get deployed to achieve our goal as either they are inefficient for pruning or may predict non-coherent explanations. We tackle this challenge by introducing a selector model that predicts real-time smooth saliency masks for pruned models. We parameterize the distribution of explanatory masks by Radial Basis Function (RBF)-like functions to incorporate geometric prior of natural images in our selector model's inductive bias. Thus, we can obtain compact representations of explanations to reduce the computational costs of our pruning method. We leverage our selector model to steer the network pruning by maximizing the similarity of explanatory representations for the pruned and original models.

Chapter 3 presents an efficient kernel size learning method for CNNs. Determining kernel sizes of a CNN model is a crucial and non-trivial design choice and significantly impacts its performance and efficiency. The majority of existing kernel size design methods rely on complex heuristic tricks or leverage neural architecture search that requires extreme computational resources. Thus, learning kernel sizes, using methods such as modeling kernels as a combination of basis functions, jointly with the model weights has been proposed as a workaround. However, previous methods cannot achieve satisfactory results or are inefficient for high-resolution and large-scale datasets. To fill this gap, we design an efficient kernel size learning method in which a size predictor model learns to predict optimal kernel sizes for a classifier given a desired number of parameters. It does so in collaboration with a kernel predictor model that predicts the weights of the kernels - given kernel sizes

predicted by the size predictor - to minimize the training objective, and both models are trained end-to-end. Our method only needs a small fraction of the training epochs of the original CNN to train these two models and find proper kernel sizes for it. Thus, it offers an efficient and effective solution for the kernel size learning problem.

In Chapter 4, we introduce a method to reduce the complexity of the model pruning process. The majority of structural pruning ideas require a pretrained model before pruning, which is costly to secure. We propose a novel structural pruning approach to jointly learn the weights and structurally prune architectures of CNN models. The core element of our method is a Reinforcement Learning (RL) agent whose actions determine the pruning ratios of the CNN model's layers, and the resulting model's accuracy serves as its reward. We conduct the joint training and pruning by iteratively training the model's weights and the agent's policy, and we regularize the model's weights to align with the selected structure by the agent. The evolving model's weights result in a dynamic reward function for the agent, which prevents using prominent episodic RL methods with stationary environment assumption for our purpose. We address this challenge by designing a mechanism to model the complex changing dynamics of the reward function and provide a representation of it to the RL agent. To do so, we take a learnable embedding for each training epoch and employ a recurrent model to calculate a representation of the changing environment. We train the recurrent model and embeddings using a decoder model to reconstruct observed rewards. Such a design empowers our agent to effectively leverage episodic observations along with the environment representations to learn a proper policy to determine performant sub-networks of the CNN model.

In the part II, we propose model pruning techniques to improve the inference efficiency

of deep generative models. First, we introduce a method to prune conditional GANs in Chapter 5. Although diffusion models have achieved state-of-the-art performance on various generative modeling tasks, they are still computationally expensive and slow to sample from, requiring tens to hundreds of forward passes to generate a sample. In contrast, GANs can generate samples in a single forward pass, making them more practical for real-time applications. Therefore, pruning GANs can prepare them to be deployed in low-latency applications. Then, we address the inference efficiency of diffusion models in Chapter 6 and Chapter 7. We propose a pruning approach for diffusion models by leveraging their gradual sampling process in Chapter 6. Finally, we develop a prompt-based pruning framework for text-to-image diffusion models in Chapter 7.

We present our method for pruning conditional GANs in Chapter 5. GANs have shown remarkable success in modeling complex data distributions for image-to-image translation. Still, their high computational demands prohibit their deployment in practical scenarios like edge devices. Existing GAN compression methods mainly rely on knowledge distillation or convolutional classifiers' pruning techniques. Thus, they neglect the critical characteristic of GANs: their *local density structure over their learned manifold*. Accordingly, we approach GAN compression from a new perspective by explicitly encouraging the pruned model to preserve the density structure of the original parameter-heavy model on its learned manifold. We facilitate this objective for the pruned model by partitioning the learned manifold of the original generator into local neighborhoods around its generated samples. Then, we propose a pruning objective to regularize the pruned model to preserve the local density structure over each neighborhood, resembling the kernel density estimation method. Also, we develop a collaborative pruning scheme in which the discrim-

inator and generator are pruned by two pruning agents. We design the agents to capture interactions between the generator and discriminator by exchanging their peer's feedback when determining their corresponding models' architectures. Thanks to such a design, our pruning method can efficiently find performant sub-networks and can maintain the balance between the generator and discriminator more effectively compared to baselines during pruning, thereby showing more stable pruning dynamics.

In Chapter 6, we propose a pruning method to reduce the sampling cost of diffusion models. Diffusion models have shown better mode coverage and superior image generation quality compared to GANs. Yet, their sampling process requires numerous denoising steps, making it slow and computationally intensive. We propose to reduce the sampling cost by pruning a pretrained diffusion model into a mixture of efficient experts. First, we study the similarities between pairs of denoising timesteps, observing a natural clustering, even across different datasets. This suggests that rather than having a single model for all time steps, separate models can serve as "experts" for their respective time intervals. As such, we separately fine-tune the pretrained model on each interval, with elastic dimensions in depth and width, to obtain experts specialized in their corresponding denoising interval. To optimize the resource usage between experts, we introduce our Expert Routing Agent, which learns to select a set of proper network configurations. By doing so, our method can allocate the computing budget between the experts in an end-to-end manner without requiring manual heuristics. Finally, with a selected configuration, we fine-tune our pruned experts to obtain our mixture of efficient experts.

We present our prompt-based pruning method for text-to-image diffusion models in Chapter 7. Text-to-image (T2I) diffusion models have demonstrated impressive image

generation capabilities, synthesizing novel images given an input text prompt. Still, their computational intensity prohibits resource-constrained organizations from deploying T2I models after fine-tuning them on their internal *target* data. While pruning techniques offer a potential solution to reduce the computational burden of T2I models, static pruning methods use the same pruned model for all input prompts, overlooking the varying capacity requirements of different prompts. Dynamic pruning addresses this issue by utilizing a separate sub-network for each prompt, but it prevents batch parallelism on GPUs. To overcome these limitations, we introduce Adaptive Prompt-Tailored Pruning (APTP), a novel prompt-based pruning method designed for T2I diffusion models. Central to our approach is a *prompt router* model, which learns to determine the required capacity for an input text prompt and routes it to an architecture code, given a total desired compute budget for prompts. Each architecture code represents a specialized model tailored to the prompts assigned to it, and the number of codes is a hyperparameter. We train the prompt router and architecture codes using contrastive learning, ensuring that similar prompts are mapped to nearby codes. Further, we employ optimal transport to prevent the codes from collapsing into a single one. We demonstrate APTP's effectiveness by pruning Stable Diffusion (SD) V2.1 using CC3M and COCO as *target* datasets. APTP outperforms the single-model pruning baselines in terms of FID, CLIP, and CMMD scores. Our analysis of the clusters learned by APTP reveals they are semantically meaningful. We also show that APTP can automatically discover previously empirically found challenging prompts for SD, *e.g.,* prompts for generating text images, assigning them to higher capacity codes.

Finally, we conclude the dissertation in Chapter 8 by summarizing our contributions, discussing the future directions, and putting our work in the context of the broader research

landscape.

Part I

Pruning and Efficient Architecture Search Techniques for Convolutional Neural Networks

Chapter 2: Interpretations Steered Network Pruning via Amortized Inferred Saliency Maps

2.1 Introduction

Convolutional Neural Networks (CNNs) have been continuously achieving state-of-the-art results on various computer vision tasks [31, 32, 33, 34, 35, 36, 37], but the required resources of popular deep models [38, 39, 40] are also exploding. Their substantial computational and storage costs prohibit deploying these models in edge and mobile devices, making the CNN compression problem a crucial task. Many ideas have attempted to address this problem to reduce models' sizes while maintaining their prediction performance. These ideas can usually be classified into one of the model compression methods categories: weight pruning [41], weight quantization [42, 43], structural pruning [44], knowledge distillation [45], neural architecture search [46], *etc.*

We focus on pruning channels of CNNs (structural pruning) since it can effectively and practically reduce the computational costs of a deep model without any post-processing steps or specifically designed hardware. Although existing channel pruning methods have achieved excellent results, they do not consider the model's interpretations during the pruning process. They tackle the pruning problem from various perspectives such as rein-

forcement learning [46], greedy search [47], and evolutionary algorithms [48]. In addition, they have utilized a wide range of metrics like channels' norm [44], loss [49], and accuracy [50] as guidance to prune the model. Thus, they emphasize the model's outputs or weights but ignore its valuable interpretations' information.

We aim to approach the structural model pruning problem from a novel perspective by exploiting the model's interpretations (a subset of input features called saliency maps) to steer the pruning. Our intuition is that the saliency maps of the pruned model should be similar to the ones for the original model. However, the existing interpretation methods are either inefficient or unreliable for pruning. Firstly, locally linear models (*e.g.*, LIME [51] and SHAP [52]) fit a separate linear model to explain the behavior of a nonlinear classifier in the vicinity of each data point. However, they need to fit a new model in each iteration of pruning that the classifier's architecture changes, which makes them inefficient for pruning. Secondly, previous works [53, 54] empirically observed that a feature importance assignment of Gradient-based methods (*e.g.*, Grad-CAM [55] and DeepLIFT [56]) might not be more meaningful than random. Moreover, Srinivas and Fleuret [57] theoretically showed that the input gradients used by these methods might seem explanatory as they are related to an implicit generative model hidden in the classifiers [58], not their discriminative function. Thus, their usage for interpreting classifiers should be avoided. Finally, perturbation-based methods [59, 60] need multiple forward passes and rely on perturbed samples that are out-of-distribution for the trained model [53] to obtain its explanations. Hence, they are neither efficient nor reliable for pruning. Different from the mentioned methods, **Amortized Explanation Models (AEMs)** [4, 61, 62] provide a theoretical framework to obtain a model's interpretations. They train a fast saliency prediction model

that can be applied in real-time systems as it can provide saliency maps with a single forward pass, making them suitable for pruning. We refer to section 2.2 for more discussion on interpretation methods.

In this chapter, at first, we provide a new AEM method that overcomes the disadvantages of previous AEM models, and then, we employ it to prune convolutional classifiers. Previous AEMs [4, 61, 62] cannot be applied to guide pruning due to several key drawbacks. REAL-X [4] proved that L2X [61] and INVASE [62] could suffer from degenerate cases where the saliency map selector predicts meaningless explanations. Although REAL-X overcomes this problem, it generates masks independently for each input feature (pixel). Thus, it neglects the geometric prior [63] in natural images that adjacent features (pixels) often correlate to each other. We empirically show in Section 2.3.3 and Fig. 2.4 that the saliency maps predicted by REAL-X may lack visual interpretability. In addition, the provided explanations have the same size as the input image, which also adds non-trivial computational costs when used for pruning. We propose a novel AEM model to tackle these problems. In contrast with REAL-X, which assumes features' independence, we employ a proper geometric prior in our model. We use a Radial Basis Function (RBF)-like function to parameterize saliency masks' distribution. By doing so, the mask generation is no longer independent for each pixel in our framework. Moreover, it enables us to infer explanations for each image with only three parameters (center coordinates and kernel expansion), saving lots of computations. We utilize such compact saliency representations to steer network pruning by reconstruction in real-time. We also find that merging guidance from the model's interpretations and outputs can further improve the pruning results. Our experimental results on benchmark datasets illustrate that our new interpretation steered

pruning method can consistently achieve superior performance compared to baselines. Our contributions are as follows:

- We propose a novel structural pruning method for CNNs designed from a new and different perspective compared to existing methods. We utilize the model's decisions' interpretations to steer the pruning procedure. By doing so, we effectively merge the guidance from the model's interpretations and outputs to discover the high-performance sub-networks.

- We introduce a new Amortized Explanation Model (AEM) such that we embed a proper geometric prior for natural images in the inductive bias of our model and enable it to predict smooth explanations for input images. We parameterize the distribution of saliency masks using RBF-like functions. Thus, our AEM can provide compact explanatory representations and save computational costs. Further, it empowers us to dynamically obtain saliency maps of pruned models and leverage them to steer the pruning procedure.

The contents of this chapter are based on our work [64] published in ECCV 2022.

2.2 Related Works

2.2.1 Interpretation Methods

Interpretation methods can get classified into four [4] main categories:

1. Gradient-based methods such as CAM [65], Grad-CAM [55], DeepLIFT [56], and LRP [66] rely on the gradients of outputs of a model $w.r.t$ input features and assume

features with larger gradients have more influence on the model's outcome [67, 68, 69], which is shown is not necessarily a valid assumption [70]. In addition, their feature importance assignment might not be more meaningful than random assignment [53, 54, 57], which makes them unreliable for pruning. Further, Srinivas and Fleuret [57] theoretically proved that input gradients are equal to the score function for the implicit generative model in classifiers [58] and are not related to the discriminative function of classifiers. Thus, they are not interpretations of the model's predictions.

2. Perturbation-based models explore the effect of perturbing input features on the model's output or inner layers to conclude their importance [59, 60, 71]. Yet, they are inefficient for pruning as they need multiple forward passes to obtain importance scores. Also, they may underestimate features' importance [56].

3. Locally Linear Models fit a linear model to approximate the behavior of a classifier in the vicinity of each data point [51, 52]. However, they require to fit a new model for each sample when the model's architecture changes during pruning, which makes them inefficient for pruning. Also, they rely on the classifier's output for out-of-distribution samples to train the linear model [53], which makes them undependable.

4. Amortized Explanation Models (AEMs) [4, 61, 62, 72] overcome the inefficiencies of the previous methods by training a *global* model - called *selector* [4] - that *amortizes* the cost of inferring saliency maps for each sample by *selecting* salient input features with a single forward pass. AEMs [4, 61, 62] provide a theoretical framework to train the selector model. To do so, they use a second *predictor* model that estimates the classifier's output target distribution given an input masked by the selector model's predicted mask. L2X [61] and INVASE [62] jointly train the selector and predictor. How-

ever, REAL-X [4] proved that doing so results in degenerate cases. REAL-X overcame this problem by training the predictor model separately with random masks. However, we show in section 2.3.3 that its predicted masks may not be interpretable for complex image classifiers. Our conjecture for a reason is that it neglects geometric prior [63] of natural images that nearby pixels correlate more to each other.

2.2.2 Network Compression

Weight pruning [41] and quantization [42, 43], structural pruning [44, 73, 74, 75, 76, 77, 78, 79, 80, 81], knowledge distillation [45], and NAS [46] are popular directions for compressing CNNs. Structural pruning has attracted more attention as it can readily decrease the computational burden of CNN models without any specific hardware changes. Early channel pruning methods [44] propose that the channels with larger norms are more critical and remove weights/filters with small L_1/L_2 norm. L_1 penalty can also be applied to scaling factors of batchnorm [82] to remove redundant channels [83]. Recent channel pruning methods adopt more sophisticated designs. Automatic model compression [46] learns the width of each layer with reinforcement learning. Metapruning [50] generates parameters for subnetworks and uses evolutionary algorithms to find the best subnetwork. Greedy subnetwork selection [47] greedily chooses each channel based on their L_2 norm. Pruning can be also used for fairness [84]. We refer to [85] for a more detailed discussion of pruning techniques.

2.2.3 Network Pruning Using Interpretations

There are a few recent works that attempt to use interpretations of a model to determine importance scores of its weights. Sabih *et al.* [86] leverage DeepLIFT [56]; Yeom *et al.* [87] use LRP [66]; and Yao *et al.* [88] utilize activation maximization [59] to determine weights' importance. However, all these methods use gradient-based methods that, as mentioned above, their predictions are unreliable and should not be used as the model's interpretations. Alqahtani *et al.* [89] visualize feature maps in the input space and use a segmentation model to find the filters that have the highest alignment with visual concepts. Nonetheless, their method needs an accurate segmentation model to find reliable importance scores for filters, which may not be available in some domains. We develop a new AEM model that is theoretically supported and improves REAL-X [4]. Moreover, in contrast with these methods, our pruning method finds the optimal subnetwork end-to-end. We also show in section 2.4.2 that our model outperforms [89].

2.3 Methodology

2.3.1 Overview

We present a novel pruning method in which we steer the pruning process of CNN classifiers using feature-wise interpretations of their decisions. At first, we develop a new intuitive AEM model that overcomes the limitations of REAL-X [4] (state-of-the-art AEM). The reason is that we incorporate the geometric prior of high correlation between adjacent input features (pixels) [63] in the images in the inductive bias of our AEM model. We

parameterize the distribution of saliency masks using Radial Basis Function (RBF)-style functions. By doing so, we can represent interpretations (saliency maps) of input images compactly. Then, we elaborate on our pruning method in which we leverage our AEM model to provide interpretations of the original and pruned classifiers. Our intuition is that saliency maps of the original and pruned models should be similar. Thus, we propose a new loss function for pruning that encourages the pruned model to have similar saliency explanations to the original one. In the following sub-sections, we introduce AEM methods and empirically show the limitations of REAL-X. Then, we elaborate on our method and its intuitions to tackle the drawbacks of previous AEMs. Finally, we present our pruning scheme.

2.3.2 Notations

We denote our dataset as $\mathcal{D} = \{(x^{(i)}, y^{(i)})\}_{i=1}^{N}$ such that $(x, y) \sim \mathcal{P}(\mathbf{x}, \mathbf{y})$ where $\mathcal{P}$ is the unknown underlying joint distribution over features and targets, and we assume that $x \in \mathbb{R}^D$ and $y \in \{1, 2, \ldots, K\}$. We show the jth feature of sample x by x_j and represent a mask m by the indices of the input features that it preserves, *i.e.*, $m \subseteq \{1, 2, \ldots, D\}$ and a masked input $m(x)$ is defined as follows:

$$m(x) = mask(x, m) = \begin{cases} x_j & j \in m \\ 0^{[1]} & \text{Otherwise} \end{cases} \tag{2.1}$$

We call the model that we aim to prune as the 'classifier' in following sections.

[1] We use zero values for the masked input features following the literature.[4, 61, 62]

2.3.3 Amortized Explanation Models (AEMs)

AEMs are a subgroup of Instance-Wise Feature Selection (IWFS) methods that aim to compute a mask with minimum cardinality for each input sample that preserves its outcome-related features. An outcome may be a classifier's predictions (usually calculated as a softmax distribution) for interpretation purposes. It can also be the population distribution of the targets (one-hot representations) when performing dimensionality reduction on the original raw data [4, 61, 62]. Although previous works [4, 61, 62] describe their formulation for the latter, we focus on the former here.

Concretely, if $\mathcal{Q}_{class}(\mathbf{y}|\mathbf{x})$ be the classifier's conditional distribution of targets given input features, the objective of AEM models is to find a mask $m(x)$ for each sample x such that

$$\mathcal{Q}_{class}(\mathbf{y}|\mathbf{x} = x) = \mathcal{Q}_{class}(\mathbf{y}|\mathbf{x} = m(x)) \tag{2.2}$$

AEMs tackle this problem by training a *global* model called *selector* that learns to predict a *local* (sample dependent) mask $m(x)$ for each sample x [4]. They train the selector by encouraging it to follow Eq. 2.2. To do so, one should quantify the discrepancy between the RHS and LHS of Eq. 2.2 when the selector model generates the mask m in the RHS. The LHS can be readily calculated by forwarding the sample x into the classifier. However, the classifier should not be used to compute the RHS because the masked sample $m(x)$ is an out-of-distribution input for it [4]. AEMs solve this issue by training a *predictor* model that predicts the conditional distribution of the classifier given a masked input. (RHS of Eq. 2.2) Then, they train the selector guided by the supervision from the predictor. We

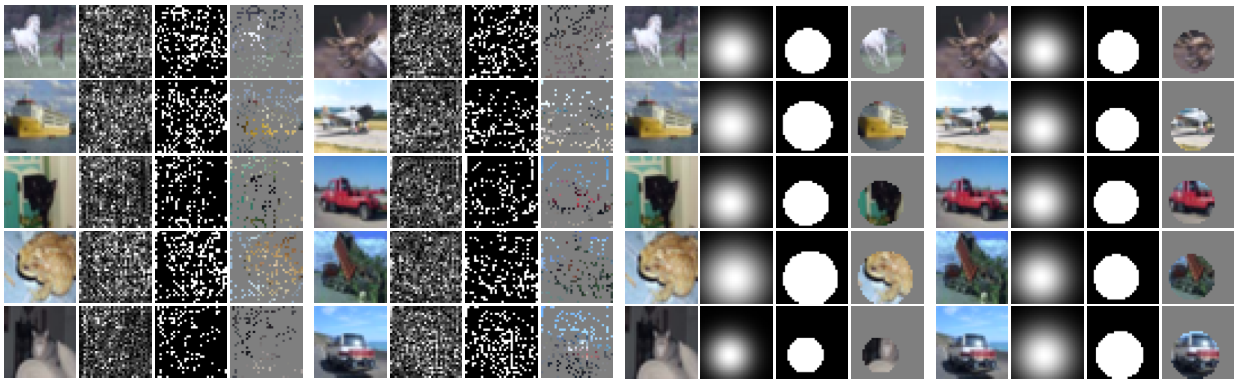
present the formulation of REAL-X [4] in supplementary S2.1.

2.3.3.1 Visualization of REAL-X Predictions

We visualize predicted explanations of REAL-X for a ResNet-56 model [39] trained on CIFAR-10 [5] in Fig. 2.4(a). (we refer to supplementary S2.3 for implementation details) As can be seen, the formulation of REAL-X cannot guide the selector model to learn to select a coherent subset of input pixels of the salient parts of the images. Thus, it may not provide interpretable explanations for the classifier. Our conjecture for the cause is that the formulation of REAL-X does not include a proper inductive bias related to natural images in the selector model. Typically, nearby pixels' values and semantic information are more correlated in natural images, known as their geometric prior [63]. REAL-X does not have such a prior in its formulation because it factorizes the explanatory masks' distribution given an input x as:

$$q_{sel}(m|x;\beta) = \prod_{i=1}^{D} q_i(m_i|x;\beta) \tag{2.3}$$

where $q_i(m_i|x;\beta) \sim Bernoulli((f_\beta(x))_i)$, *i.e.*, the distribution over the selector's output mask is factorized as a product of marginal Bernoulli distributions over mask's elements, and the parameter for each element gets calculated independently. ($f_\beta(x)$ is the selector model parameterized by β). Hence, the selector model does not have the inductive bias that parameters of nearby Bernoulli distributions should be close to each other to make the sampled masks coherent. Instead, it should 'discover' such prior during training, which is infeasible with limited data and training epochs in practice.



(a) **REAL-X**[4] (b) **Ours**

Figure 2.1: Input features selected by **a) REAL-X** [4] and **b) our model** to explain decisions of a ResNet-56 classifier for samples from CIFAR-10 [5]. In the sub-figures from left to right: 1st column shows the original image. Both models output an array (2nd columns) that each value of it is the parameter of the predicted Bernoulli distribution over the corresponding mask pixel. In the 3rd column, we show the masks generated such that a pixel's value is one provided that its predicted Bernoulli parameter is bigger than 0.5 and zero otherwise. The 4th columns show the masked inputs. Our model's explanations are easier to interpret than the ones by REAL-X that may seem random for some samples.

2.3.4 Proposed AEM Model

We introduce a new selector scheme that respects the proximity geometric prior. To do so, we assume that the parameters of the Bernoulli distributions of mask pixels should have a Radial Basis Function (RBF) style functional form over the pixels. The center of the RBF kernel should be on the salient part of the image most relevant to the classifier's prediction, and the Bernoulli parameters should decrease as the pixel location gets far from the kernel's center. A parameter σ controls the area of a mask. Our assumption is reasonable for multi-class classifiers in which, typically, a single object/region in their input image determines the target class. Formally, considering a 2D mask that its coordinates are parametrized by (z, t) and the parameters of a 2D RBF kernel being (c_z, c_t, σ), we calculate the Bernoulli parameter (BP) of a pixel at location (z, t) as follows:

$$f_{BP}(z,t;c_z,c_t,\sigma)=\exp\Big\{(\frac{-1}{2\sigma^2}[(z-c_z)^2+(t-c_t)^2])\Big\} \tag{2.4}$$

This formulation has two crucial benefits: 1) It ensures that Bernoulli parameters of a mask's proximal pixels are close to each other. Thus, the resulting sampled masks will be much more coherent and smooth than REAL-X. 2) It simplifies the selector model's task significantly. In REAL-X, the selector should learn how to calculate Bernoulli parameters for each pixel that, for instance, will be $224 \times 224 = 50176$ independent functions for the standard ImageNet [31] training. In contrast, in our formulation, the selector should only learn to accurately estimate three values corresponding to the center's coordinates (c_z, c_t) and an expanding parameter σ for the RBF kernel. Given the estimated values, Bernoulli parameters of the output mask's pixels can be readily calculated by Eq. 2.4. In other words, if the input images have spatial dimensions $M * N$, and we denote the selector function (implemented by a deep neural network) with $f_{sel}(x;\beta)$, our selector's distribution over masks given input images is:

$$\begin{aligned}
[c_z,c_t,\sigma] &= f_{sel}(x;\beta) \\
q_{i,j}(m_{i,j}|x;\beta) &= Bernoulli(f_{BP}(i,j;c_z,c_t,\sigma)) \\
q_{sel}(m|x;\beta) &= \prod_{i=1}^{M}\prod_{j=1}^{N} q_{i,j}(m_{i,j}|x;\beta)
\end{aligned} \tag{2.5}$$

In Eq. 2.5, β denotes the selector's parameters, and we illustrate a predicted RBF kernel by our selector in Fig. 2.2. In summary, our intuition is that by incorporating the geometric prior in the inductive bias of our framework, the selector will search for proper functional form for Bernoulli parameters over pixels' locations in the RBF family of functions, not all

possible ones. As a result, it can find the optimal functional form more readily and robustly. Moreover, our selector model can provide a real-time and compact representation (RBF parameters) for saliency maps, which enables us to efficiently compare the interpretations of the original and pruned models to steer the pruning process. (section 2.3.6, Fig. 2.3)

2.3.5 AEM Training

We train our selector model by encouraging it to generate an explanatory mask m for each sample x such that it follows Eq. 2.2. To do so, as mentioned in section 2.3.3, we need to estimate the classifier's conditional distribution of targets given masked inputs (RHS of Eq. 2.2) to train our selector model. Such an estimate can quantify the quality of a mask generated by the selector model by measuring the discrepancy between the LHS and RHS of Eq. 2.2.

2.3.5.1 Predictor Model

We train a predictor model to calculate the classifier's conditional distribution of targets given a masked input. (RHS of Eq. 2.2) As we designed our selector to predict RBF-style masks (Eq. 2.5), we train our predictor to predict the classifier's output distribution when the input is masked by a random RBF-style mask. Using random RBF masks allows us to mimic any potential RBF-masked input. Hence, our predictor's training objective is:

$$\min_{\theta} \mathcal{L}_{pred}(\theta) = \mathbb{E}_{x\sim\mathcal{P}(\mathbf{x})}\mathbb{E}_{c'_z,c'_t,\sigma'}[\mathbb{E}_{m'\sim\mathcal{B}(m|c'_z,c'_t,\sigma')}L_\theta(x, m'(x))] \tag{2.6}$$

where $L_\theta(\cdot,\ \cdot)$ and $\mathcal{B}(\cdot)$ are defined as:

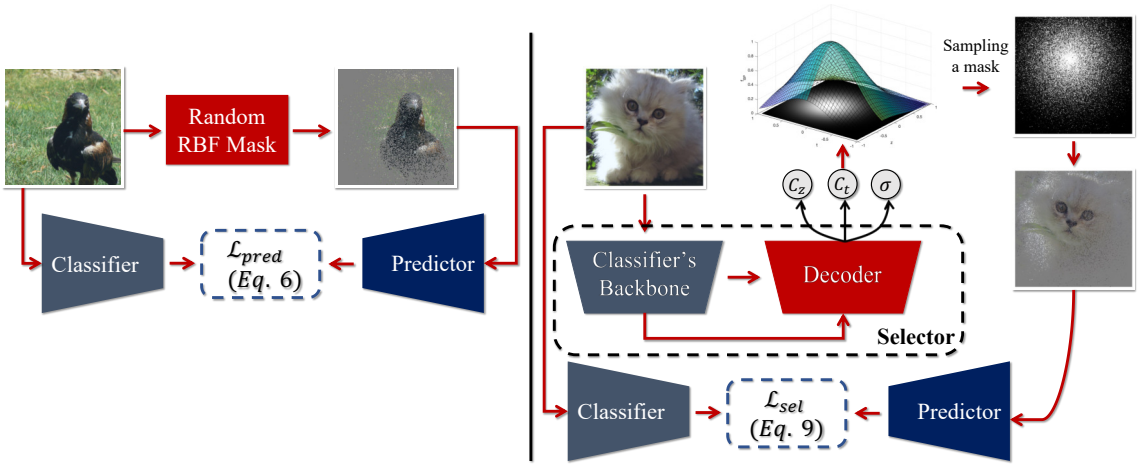


Figure 2.2: Our AEM model. The goal is to train the selector model on the right (U-Net model in dashed line) to predict interpretations (saliency maps) of the classifier for each input sample. We train the selector by encouraging it to follow Eq. 2.2. **(Left):** We train a predictor model that learns to predict the classifier's output distribution given a masked input (RHS of Eq. 2.2). We do so using inputs masked by random RBF masks as our selector's masks have RBF-style. (Sec. 2.3.4) **(Right):** Given the trained predictor, we train the selector model using obj. 2.8 that enforces it to follow Eq. 2.2. We use the classifier's convolutional backbone as the encoder of the selector and only train its decoder for computational efficiency. Then, we use the trained decoder to prune the encoder. (Fig. 2.3)

$$
\begin{aligned}
L_\theta(x, m'(x)) &= KL(\mathcal{Q}_{class}(\mathbf{y}|\mathbf{x}=x),\ q_{pred}(\mathbf{y}|\mathbf{x}=m'(x);\theta)) \\
\mathcal{B}(m|c'_z, c'_t, \sigma') &= \prod_{i=1}^{M}\prod_{j=1}^{N} Bernoulli(f_{BP}(i,j;c'_z,c'_t,\sigma'))
\end{aligned}
\tag{2.7}
$$

Eq. (2.6), L_θ form the predictor's objective to learn the conditional distribution of the classifier for targets given masked inputs (RHS of Eq. 2.2). $\mathcal{B}(\cdot)$ generates random masks with random RBF style (f_{BP}), and KL denotes Kullback-Leibler divergence [90]. Now, we should define the distribution for the parameters c'_z, c'_t, and σ' for a random RBF function. Let us assume that the origin of our 2D coordinate system is the top left of an input image with spatial dimensions M, N. In theory, c'_z and c'_t can have any real values, and the σ' can be any positive real number in Eq. 2.4. However, considering that the salient part[s]

is inside the image region, we are interested that the predictor learns to correctly estimate $\mathcal{Q}_{class}(\mathbf{y}|\mathbf{x} = m(x))$ (RHS of Eq. 2.2) when the selector predicts that the center of the RBF kernel is inside the image area. Hence, we assume that the distributions of c'_z and c'_t are uniform across image dimensions, *i.e.*, $c'_z \sim U[0, M]$ and $c'_t \sim U[0, N]$. In addition, the parameter σ' determines the degree that an RBF kernel expands on the image, and the values $\sigma' \geq 2 * max\{M, N\}$ practically provide the same Bernoulli parameters for all the mask's pixels when c'_z and c'_t are in the image region. Thus, we can reasonably assume that $\sigma' \sim U[0,\ 2 * max\{M,\ N\}]$ for training the predictor in practice.

2.3.5.2 Selector Training

Given a predictor model denoted by q_{pred} and trained with random RBF masks, we train our selector model with the following objective:

$$\min_{\beta} \mathcal{L}_{sel}(\beta) = \mathbb{E}_{x\sim\mathcal{P}(\mathbf{x})}\mathbb{E}_{m'\sim q_{sel}(m|x;\beta)}[L(x, m'(x)) + \lambda_1\mathcal{R}(m') + \lambda_2\mathcal{S}(m')] \tag{2.8}$$

such that $L(\cdot,\ \cdot)$, $\mathcal{R}(\cdot)$, and $\mathcal{S}(\cdot)$ are defined as:

$$\begin{aligned} &L(x, m'(x)) = KL(\mathcal{Q}_{class}(\mathbf{y}|\mathbf{x} = x),\ q_{pred}(\mathbf{y}|\mathbf{x} = m'(x))), \\ &\mathcal{R}(m') = ||m'||_0,\ \mathcal{S}(m') = \sum_{i=1}^{M}\sum_{j=1}^{N}[(m'_{i,j} - m'_{i+1,j})^2 + (m'_{i,j} - m'_{i,j+1})^2] \end{aligned} \tag{2.9}$$

$L(x, m'(x))$ encourages the selector to follow Eq. 2.2 as $q_{pred}(\mathbf{y}|\mathbf{x} = m'(x))$ approximates the RHS of Eq. 2.2 given an input masked by the RBF mask predicted by the selector. $\mathcal{R}(m')$ regularizes the number of selected features. We add the smoothness loss $\mathcal{S}(m')$

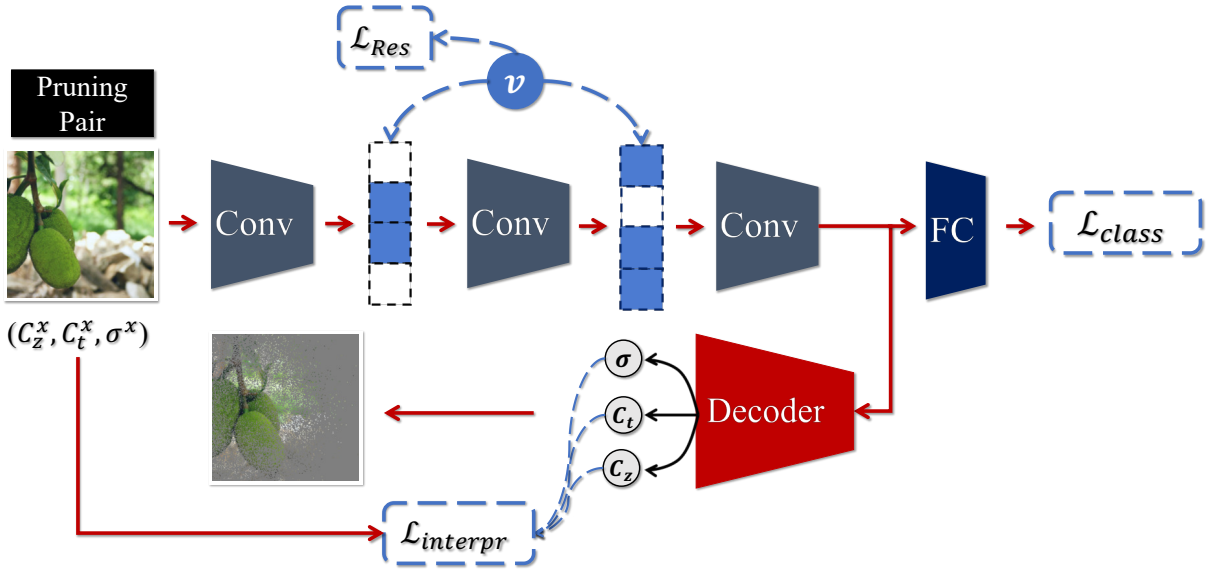


Figure 2.3: Our pruning method. The classifier to be pruned is shown on top. (Conv layers and FC). The U-Net (Conv layers and the Decoder) is our trained selector model that can predict RBF parameters of the saliency map of each input for the classifier. The selector model is trained such that the pretrained backbone of the classifier is used as its encoder (Conv layers) and kept frozen during training. (see Fig. 2.2) Thus, we freeze the selector and classifier's weights and insert our pruning gates between the selector's encoder layers for pruning the classifier. Given a pruning pair (a sample and its RBF saliency map's parameters for the original classifier), we train the gate parameters to prune the classifier such that the pruned model have similar interpretations ($\mathcal{L}_{interpr}$) and accuracy ($\mathcal{L}_{class}$) to the original classifier while requiring lower computational resources ($\mathcal{L}_{Res}$).

to further encourage the selector to output smooth masks. As Eq. 2.8 requires sampling from predicted distribution by the selector, direct backpropagation of gradients to train its parameters, β, is not possible. Thus, we use the Gumbel-Sigmoid [91, 92] trick to train the model. We use a U-Net [93] architecture to implement the selector module of our AEM model, as shown in Fig. 2.2. We refer to supplementary S2.3 for more details of our AEM training.

2.3.6 Pruning

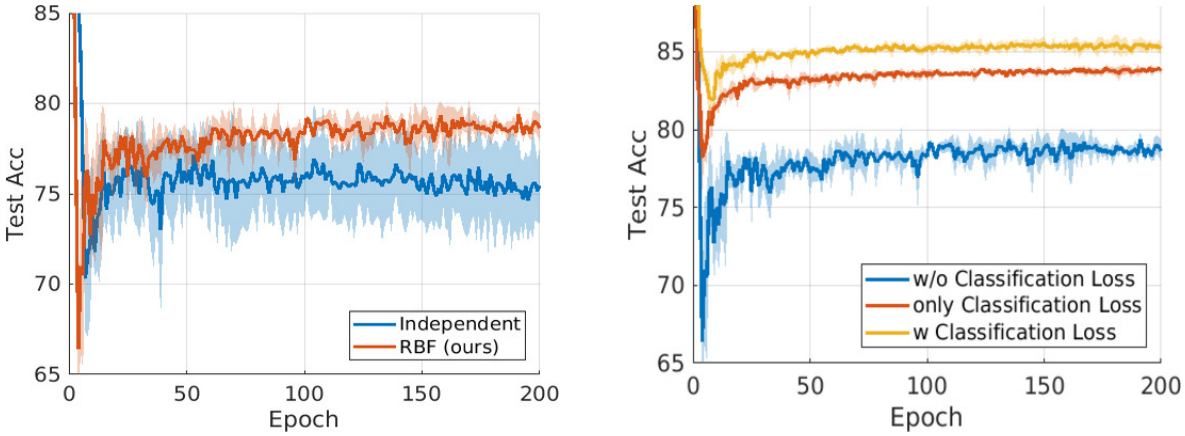
In this section, we introduce our pruning method that leverages interpretations of a classifier to steer its pruning process. Our intuition is that the interpretations (saliency maps) of the original and pruned classifiers should be similar. Thus, we design our pruning method as follows. As discussed in section 2.3.5 and Fig. 2.2, we use the convolutional backbone of the classifier as the encoder of the U-Net architecture for the selector model. We keep the encoder weights frozen and only train the decoder when training the selector model for computational efficiency. (Fig. 2.2) Furthermore, doing so provides us the flexibility to keep the decoder frozen and prune the encoder such that the pruned model should have similar output RBF parameters to the original model. (Fig. 2.3)

Formally, we employ our trained selector model to predict saliency maps of the original classifier for training samples. For each sample x_k, it provides the parameters of the RBF kernel for its saliency map as $\mathcal{C}_{x_k} = [c_z^k, c_t^k, \sigma^k]$. Then, we insert our pruning gates, parameterized by θ_g, between the layers of the encoder. We represent the architectural vector generated by the gates with $\mathbf{v}$. Finally, we prune the encoder (classifier's backbone) by regularizing the gate parameters to maintain the interpretations and accuracy of the pruned classifier similar to the original one while reducing its computational budget as follows:

$$\min_{\theta_g} L(f(x; \mathcal{W}, \mathbf{v}), y) + \gamma_1 \|\mathcal{C}_x - f_{sel}(x; \beta, \mathbf{v})\|_2^2 \ + \gamma_2 \mathcal{R}_{res}(T(\mathbf{v}), pT_{all}) \tag{2.10}$$

where $L(\cdot,\cdot)$ is the classification loss, $f(\cdot;\mathcal{W},\mathbf{v})$ denotes our classifier (encoder of the U-Net and the FC layer in Fig. 2.3) parameterized by weights $\mathcal{W}$ and the subnetwork selection vector $\mathbf{v}$. $f_{sel}(x;\beta,\mathbf{v})$ is our trained selector model ($f_{sel}(x;\beta)$ in Eq. 2.5) augmented by the architecture vector $\mathbf{v}$ after inserting the pruning gates into its encoder. We calculate $\mathbf{v}$ using Gumbel-sigmoid function $g(\cdot)$: $\mathbf{v} = g(\theta_g)$ [91, 92], which controls openness or closeness of a channel. The second term in Eq. 2.10 utilizes the interpretations of the original and pruned classifiers to steer pruning through the selector model $f_{sel}(x;\beta,\mathbf{v})$ by encouraging the similarity of their predicted RBF parameters. $\mathcal{R}_{res}$ is the FLOPs regularization to ensure the pruned model reaches the desired FLOPs rate pT_{all}. T_{all} is the total prunable FLOPs of a model, $T(\mathbf{v})$ is the current FLOPs rate determined by the subnetwork vector $\mathbf{v}$, and p controls the pruning rate. γ_1 and γ_2 are hyperparameters to control the strength of related terms. During pruning, we only optimize θ_g and keep $\mathcal{W}$ and β frozen.

We emphasize that our amortized explanation prediction selector model, $f_{sel}(x;\beta,\mathbf{v})$, enables us to readily perform interpretation-steered pruning because it can dynamically predict each sample's saliency map's RBF parameters ($[c_z, c_t, \sigma]$) given the current subnetwork vector $\mathbf{v}$ with a single forward pass. In contrast, optimization-based explanation methods [51, 52] need to fit a new model, and perturbation-based methods [59, 60, 71] have to make multiple forward passes for the newly selected subnetwork to obtain its explanations. Therefore, they are inefficient to achieve the same goal. We provide the detailed parameterization of channels ($g(\cdot)$) and $\mathcal{R}_{res}$ in supplementary S2.3.1.



(a) (b)

Figure 2.4: **(a):** Test accuracy of different masks' parameterization schemes. (RBF (ours) *vs.* Independent (REAL-X [4])) **(b):** Test accuracy w/wo using the classification loss. All results are for 3 run times with ResNet-56 on CIFAR-10. Shaded areas represent variance.

2.4 Experiments

We use CIFAR-10 [5] and ImageNet [31] to validate the effectiveness of our proposed model. We refer to supplementary S2.3 for details of our experimental setup. We call our method ISP (**I**nterpretations **S**teered **P**runing) in the experiments.

2.4.1 Analysis of Different Settings

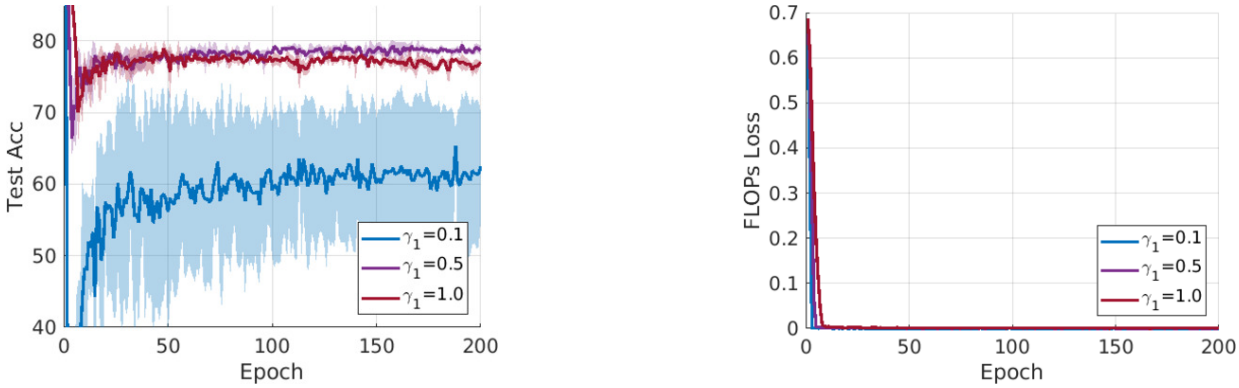
Before we formally present our experimental results compared to competitive methods, we study the effect of different design choices for our model's components on its performance. We keep the resource regularization ($\mathcal{R}_{res}$) term in obj. 2.10 and add/drop other ones in all settings.

In our first experiment, we explore the impact of γ_1 by only using interpretations (second term of Obj. 2.10) to steer the pruning. Fig. 2.5(a,b) and Fig. 2.4(a) demonstrate

the results. We can observe in Fig. 2.5(a,b) that small γ_1 values (*e.g.,* 0.1) result in a weaker supervision signal from the interpretations and make the exploration of subnetworks unstable (showing high variance), whereas larger ones make the training smooth. Fig. 2.4(a) illustrates the influence of RBF/independent masks' parameterization scheme in Eq. 2.5 (ours)/Eq. 2.3 (REAL-X [4]). Our RBF-style model brings better performance than independent parameterization. The latter becomes unstable and less effective when the training proceeds. The instability happens possibly because the pruning gets trapped in some local minima due to noisy and unstructured masks. We can also observe that interpretations on their own provide stable and efficient signals for pruning.

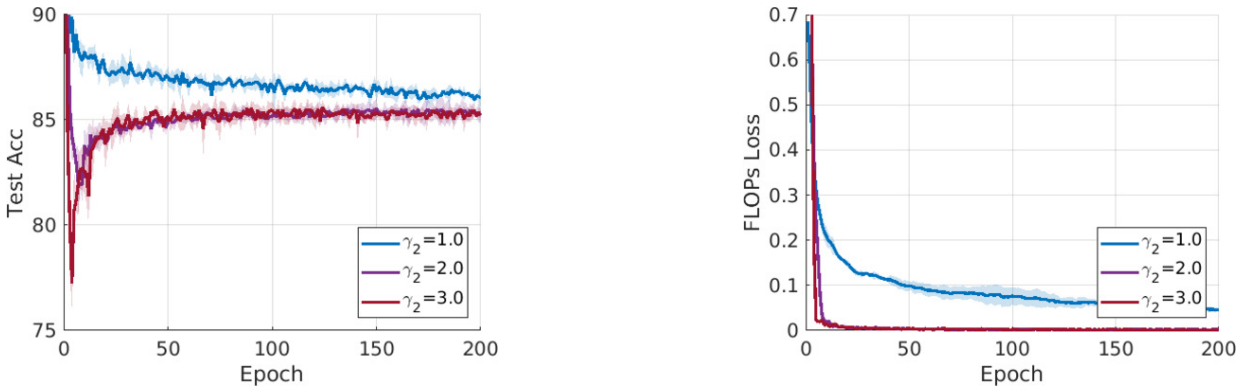
In our second experiment, we examine the impact of γ_2 while utilizing all three terms in objective 2.10 for pruning. Fig. 2.5(c, d) indicates that small γ_2 (*e.g.,* 1.0) shows higher accuracy but may not be able to push the FLOPs regularization to 0, *i.e.*, reach the predefined pruning rate p. Larger values can satisfy the resource constraint while showing acceptable performance.

Finally, we examine the performance of different combinations of components in objective 2.10. The results are available in Fig. 2.4(b). Specifically, 'w/o Classification Loss' represents using the second and third terms, 'only Classification Loss' indicates using the first and third ones, and 'w Classification Loss' means using the full objective function. It is plausible that 'only Classification Loss' performs better than only interpretations ('w/o Classification Loss') since the loss function is a 'less noisy' signal for accuracy compared to the interpretations. Furthermore, incorporating interpretations enhances the supervision signal and yields the best performance. This observation indicates that interpretations contain guidance from different perspectives complementary to the classification loss that



(a) γ_1: accuracy

(b) γ_1: $\mathcal{R}_{res}$ loss



(c) γ_2: accuracy

(d) γ_2: $\mathcal{R}_{res}$ loss

Figure 2.5: **(a), (b):** The model's test accuracy and the FLOPs regularization term when changing γ_1, and **(c), (d):** when varying γ_2. All results are run for 3 times with ResNet-56 on CIFAR-10. Shaded areas represent variance.

only focuses on the model's outputs.

2.4.2 Comparasion Results

CIFAR-10 Results: Tab. 2.1 summarizes the results on CIFAR-10. For **ResNet-56**, ISP outperforms baselines with a similar FLOPs pruning rate. It has a pruning rate on par with EEMC [80] while it shows higher Δ-Acc (+0.18% *vs.* +0.06%). For **MobileNet-V2**, ISP simultaneously prunes 18% more FLOPs than DCP and Uniform. It also achieves a better accuracy improvement (+0.10% higher Δ-Acc) than DCP.

Table 2.1: Comparison of results on CIFAR-10. Δ-Acc represents the performance changes relative to the baseline, and +/− indicates an increase/decrease, respectively.

Model	Method	Baseline Acc	Pruned Acc	Δ-Acc	Pruned FLOPs
ResNet-56	DCP-Adapt [76]	93.80%	93.81%	+0.01%	47.0%
	SCP [94]	93.69%	93.23%	−0.46%	51.5%
	FPGM [74]	93.59%	92.93%	−0.66%	52.6%
	SFP [95]	93.59%	92.26%	−1.33%	52.6%
	FPC [96]	93.59%	93.24%	−0.25%	52.9%
	HRank [97]	93.26%	92.17%	−0.09%	50.0%
	EEMC [80]	93.62%	93.68%	+0.06%	**56.0%**
	ISP (ours)	93.56%	**93.74%**	**+0.18%**	54.0%
MobileNetV2	Uniform [76]	94.47%	94.17%	−0.30%	26.0%
	DCP [76]	94.47%	94.69%	+0.22%	26.0%
	ISP (ours)	94.53%	**94.85%**	**+0.32%**	**44.0%**

ImageNet Results: We present the results on ImageNet in Tab. 2.2. For **ResNet-34**, ISP achieves the best trade-off between the performance and FLOPs reduction. It achieves Δ Top-1 close to Taylor [75], but ISP can prune 19.8% more FLOPs. Also, with similar FLOPs pruning rate, ISP outperforms FPGM [74] by 0.84% Δ Top-1. For **ResNet-50**, our model can achieve the largest pruning rate, 56.6%, with the best Δ Top-1/Top-5 being $-0.16\%/-0.12\%$ showing 0.33%/0.23% improvement compared to EEMC [80]. For **ResNet-101**, ISP is the only method that its pruned network has better accuracy than the original one. Also, it accomplishes the highest pruning rate, 56.8%, with a significant 11.7% gap with PFP [79]. For **MobileNetV2**, ISP has a pruning rate competitive (+29.0% *vs.* +30.7%) to MetaPruning [50] and reaches the highest Δ Top-1, with a 0.65% margin with MetaPruning. We also note that ISP significantly outperforms QI [89] (in terms of both accuracy improvement and pruning rate for ResNet-50/101) that aims to perform interpretable pruning by finding filters that are aligned with visual concepts, which illustrates the superiority of our proposed AEM model for pruning compared to other interpretation techniques.

Table 2.2: Comparison results on ImageNet with ResNet-34/50/101 and MobileNet-V2.

Model	Method	Baseline Top-1 Acc	Baseline Top-5 Acc	Δ-Acc Top-1	Δ-Acc Top-5	Pruned FLOPs
ResNet-34	FPGM [74]	73.92%	91.62%	−1.29%	−0.54%	41.1%
	Taylor [75]	73.31%	-	−0.48%	-	24.2%
	ISP (ours)	73.31%	91.42%	**−0.45%**	**−0.40%**	**44.0%**
ResNet-50	DCP [76]	76.01%	92.93%	−1.06%	−0.61%	55.6%
	CCP [77]	76.15%	92.87%	−0.94%	−0.45%	54.1%
	FPGM [74]	76.15%	92.87%	−1.32%	−0.55%	53.5%
	ABCP [78]	76.01%	92.96%	−2.15%	−1.27%	54.3%
	QI [89]	74.90%	92.10%	−1.31%	−0.27%	50.0%
	PFP [79]	76.13%	92.86%	−0.92%	−0.45%	44.0%
	EEMC [80]	76.15%	92.87%	−0.49%	−0.35%	56.0%
	ISP (ours)	76.13%	92.86%	**−0.16%**	**−0.12%**	**56.6%**
ResNet-101	FPGM [74]	77.37%	93.56%	−0.05%	0.00%	41.1%
	Taylor [75]	77.37%	-	−0.02%	-	39.8%
	QI [89]	76.40%	92.80%	−2.31%	−0.86%	50.0%
	PFP [79]	77.37%	93.56%	−0.94%	−0.44%	45.1%
	ISP (ours)	77.37%	93.56%	**+0.40%**	**+0.22%**	**56.8%**
MobileNet-V2	Uniform [98]	71.80%	91.00%	−2.00%	1.40%	30.0%
	AMC [46]	71.80%	-	−1.00%	-	30.0%
	CC [99]	71.88%	-	−0.97%	-	28.3%
	MetaPruning [50]	72.00%	-	−0.80%	-	**30.7%**
	ISP (ours)	71.88%	90.29%	**−0.15%**	**−0.08%**	29.0%

2.5 Chapter Summary and Conclusions

We proposed a novel neural network pruning method that utilizes interpretations of the model as guidance for its pruning procedure. We showed that Amortized Explanation Models (AEM) are suitable for our purpose as they can provide real-time explanations of a model. We empirically showed that explanation masks of REAL-X [4], state-of-the-art AEM, might lack a meaningful structure and not be interpretable. Thus, we introduced a new AEM model that overcomes this problem by respecting the geometric prior of natural images and finding the optimal functional form over pixel's Bernoulli parameters of explanatory masks in the RBF functions' family. Finally, we leverage the predictions of our AEM model to steer the pruning process in our formulation. Our experimental results on benchmark data demonstrate that the interpretations of a parameter-heavy classifier provide valuable information to steer its pruning process, complementing the guidance from its outputs, which are the main focus of previous methods.

Supplementary Materials for Chapter 2

S2.1 REAL-X Formulation Development for Interpretation of Classifiers

This section provides details about the connections of REAL-X formulation presented in [4] for dimensionality reduction of samples and its version for interpreting a classifier. At first, we show the REAL-X formulation for dimensionality reduction for completeness and then derive its formulation for the interpretation of classifiers.

S2.1.1 REAL-X for Dimensionality Reduction

Amortized Explanation Models (AEMs) [4, 61, 62, 72] aim to learn to predict a 'sample-specific' mask for each sample such that preserved features contain all information related to an outcome. An outcome in REAL-X [4] is the target (label) of the sample, *i.e.*,

$$\mathcal{P}(\mathbf{y}|\mathbf{x} = x) = \mathcal{P}(\mathbf{y}|\mathbf{x} = m(x)) \tag{2.11}$$

Here $\mathcal{P}$ is the joint population distribution on inputs and targets that is unknown in practice.

We emphasize that the goal is not to explain a classifier's predictions in this formulation. Instead, it is dimensionality reduction by only keeping input features (pixels in images) that preserve the information related to labels of samples.

REAL-X trains a selector model $f_\beta(\cdot)$ implemented with a Deep Neural Network (DNN) function parameterized by β that predicts a distribution over possible explanatory

masks for a given sample, and $f_\beta(\cdot) \in \mathbb{R}^D$. The distribution is factorized as a product of marginal Bernoulli distributions over mask's pixels, *i.e.*,

$$q_{sel}(m|x;\beta) = \prod_{i=1}^{D} q_i(m_i|x;\beta) \tag{2.12}$$

$$q_i(m_i|x;\beta) \sim Bernoulli((f_\beta(x))_i)$$

During training, the selector model is encouraged to predicts masks that follow Eq. (2.11). To do so, a predictor model is used that estimates population conditional distribution of targets given masked inputs $\mathcal{P}(\mathbf{y}|\mathbf{x} = m(x))$ and trained by the following objective:

$$\max_\theta \mathbb{E}_{(x,y)\sim\mathcal{P}}\mathbb{E}_{m'\sim\mathcal{B}(0.5)}[\log(q_{pred}(\mathbf{y} = y|\mathbf{x} = m'(x);\theta))] \tag{2.13}$$

This is equivalent to

$$\min_\theta \mathbb{E}_{(x,y)\sim\mathcal{P}}\mathbb{E}_{m'\sim\mathcal{B}(0.5)}[KL(\mathcal{P}(\mathbf{y}|\mathbf{x} = x)\|q_{pred}(\mathbf{y}|\mathbf{x} = m'(x);\theta))] \tag{2.14}$$

where we represent the conditional population distribution $\mathcal{P}(\mathbf{y}|\mathbf{x} = x)$ with one-hot vectors. Finally, given a pretrained predictor model, REAL-X trains a selector model to maximize:

$$\max_\beta \mathbb{E}_{(x,y)\sim\mathcal{P}}\mathbb{E}_{m'\sim\ q_{sel}(m|x;\beta)}[\log(q_{pred}(\mathbf{y} = y|\mathbf{x} = m'(x))) - ||m'||_0] \tag{2.15}$$

Similar to Eqs. (2.13, 2.14), the log-likelihood term in Eq. (2.15) can be replaced with

$$KL(\mathcal{P}(\mathbf{y}|\mathbf{x}=x)||q_{pred}(\mathbf{y}|\mathbf{x}=m'(x))) \tag{2.16}$$

S2.1.2 REAL-X for Interpretation of Classifiers

Now, we develop the formulation of REAL-X for interpreting a classifier's predictions. The new goal is to learn to find a mask for each sample such that it preserves information related to the classifier's predictions in the remaining input features, *i.e.*,

$$\mathcal{Q}_{class}(\mathbf{y}|\mathbf{x}=x^{(i)}) = \mathcal{Q}_{class}(\mathbf{y}|\mathbf{x}=m(x^{(i)})) \tag{2.17}$$

Therefore, similar to Eq. (2.14), the objective for training a predictor model that estimates the conditional distribution of the classifier given masked inputs will be:

$$\min_{\theta} \mathbb{E}_{x\sim\mathcal{P}(\mathbf{x})}\mathbb{E}_{m'\sim\mathcal{B}(0.5)}L_\theta(x, m'(x)) \tag{2.18}$$

$$L_\theta(x, m'(x)) = KL(\mathcal{Q}_{class}(\mathbf{y}|\mathbf{x}=x), q_{pred}(\mathbf{y}|\mathbf{x}=m'(x);\theta)) \tag{2.19}$$

where we have replaced the population conditional distribution $\mathcal{P}(\mathbf{y}|\mathbf{x}=x)$ in Eq. (2.14) with the one for the classifier $\mathcal{Q}_{class}(\mathbf{y}|\mathbf{x}=x)$, which is usually implemented as a softmax distribution.

Lastly, we can train a selector model guided by a pretrained predictor to obey Eq. (2.17) by minimizing:

$$\min_{\beta} \mathbb{E}_{x\sim\mathcal{P}(\mathbf{x})}\mathbb{E}_{m'\sim\ q_{sel}(m|x;\beta)}[L(x, m'(x)) + \lambda||m'||_0] \tag{2.20}$$

Again, this is equivalent to Eq. (2.15) by replacing the population conditional distribution $\mathcal{P}(\mathbf{y}|\mathbf{x} = x)$ in Eq. (2.16) with the one for the classifier, *i.e.*, $\mathcal{Q}_{class}(\mathbf{y}|\mathbf{x} = x)$. We use Eqs. (2.18, 2.20) to train REAL-X to explain decisions of a ResNet-56 architecture on CIFAR-10 in section 2.3.3.1.

S2.2 Implementation Details of Our AEM

As a recall, the formulation for training our AEM model is the selector minimizing:

$$\min_{\beta} L_{sel}(\beta) \quad = \quad \mathbb{E}_{x\sim\mathcal{P}(\mathbf{x})}\mathbb{E}_{m'\sim q_{sel}(m|x;\beta,\mathbf{v})}[L(x, m'(x)) \ + \ \lambda_1\mathcal{R}(m') \ + \ \lambda_2\mathcal{S}(m')] \tag{2.21}$$

$$\mathcal{R}(m') = ||m'||_0$$

$$\mathcal{S}(m') = \sum_{i=1}^{M}\sum_{j=1}^{N}[(m'_{i,j} - m'_{i+1,j})^2 + (m'_{i,j} - m'_{i,j+1})^2]$$

such that we factorize $q_{sel}(m|x;\beta,\mathbf{v})$ as a product of marginal Bernoulli distributions over the pixels. The parameters of Bernoulli distributions have RBF form over pixel locations, *i.e.*, we calculate the parameter for a pixel at location (z,t) as:

$$f_{BP}(z,t;c_z,c_t,\sigma) = \exp\Big\{(\frac{-1}{2\sigma^2}[(z-c_z)^2+(t-c_t)^2])\Big\} \tag{2.22}$$

In this section, we provide more practical details about the implementation of our AEM, namely the U-Net architecture's layers and training procedure of the selector model for ImageNet [100] and CIFAR-10 [5].

S2.2.1 U-Net Architecture

We use a U-Net [93] architecture with a feature filter module proposed in [72] to implement the selector module of our AEM model. It provides the flexibility to use the feature extractor backbone of the pretrained classifier (*e.g.*, scales 1-5 of ResNet [39] before its global average pooling layer) as the encoder of U-Net and only train the decoder part for computational efficiency. The feature filter is an embedding layer (denoted by C) learned along with the decoder that performs the initial localization of the target class by attenuating spatial locations that do not contain the target [72]. It applies such operation on the output of the encoder before inputting it to the decoder. Formally, the output filtered feature Z at spatial location (i,j) given input features X and target class embedding C_y is calculated as:

$$Z_{i,j} = X_{i,j}\sigma(X_{i,j}^T C_y) \tag{2.23}$$

S2.3 Experimental Setup

We use CIFAR-10 [5] and ImageNet [31] to validate the effectiveness of our proposed model. For all experiments, we train the original classifier from scratch (CIFAR-10) or adopt PyTorch [101] pretrained models (ImageNet). To train the AEM model, we follow two main steps: 1) We train the predictor with Eq. 2.6 using the same architecture as the classifier starting from scratch (CIFAR-10) or PyTorch pretrained checkpoint (ImageNet). 2) We use the convolutional feature extraction backbone of the classifier as the encoder of the selector's U-Net architecture and keep it frozen during training the decoder for computational efficiency (Fig. 2.2). The decoder is trained with objective 2.9 and steered by the trained predictor from the previous step.

CIFAR-10: For CIFAR-10 experiments, we evaluate our model on ResNet-56 [39] and MobileNetV2 [98]. We train the predictor model for 300 epochs for both models with a mini-batch size of 128 using ADAM optimizer [102] with a learning rate of 0.0001, exponential decay rates $(\beta_1, \beta_2) = (0.9, 0.999)$, and weight decay of 0.0001. We train the selector model for 10 epochs with mini-batch size 16 and the same optimization configuration. We found that the parameter setting $\lambda_1 = 0.2$ and $\lambda_2 = 0.001$ perform well for both models. We randomly partition the official training set with a 0.9/0.1 ratio to form our training/validation sets and use the official test set as our test partition. During pruning, we select 5% of the official training set as the subset for pruning. We choose $\gamma_1 = 0.5$ and $\gamma_2 = 2.0$ for pruning. We optimize Eq. 2.10 for pruning for 200 epochs by using the subset with ADAM optimizer. After pruning, we finetune the model for 200 epochs with SGD optimizer of momentum 0.9, weight decay 0.0001, and start learning rate 0.1. We decay

the learning rate to 0.01 and 0.001 at epoch 50 and 100. The mini-batch size is 128 for both pruning and finetuning. As mentioned in section 2.3.5 and Fig. 2.2, we use the convolutional feature extraction backbone of the classifier as the encoder of the selector's U-Net architecture and keep it frozen during training the decoder for computational efficiency. ResNet-56 [39] and MobileNetV2 [98] architectures generate three scales of representations before their average pooling layer considering (number of channels, spatial dimensions) from a raw input with $3 \times 32 \times 32$ spatial dimensions. ResNet-56 scales' representations have $\{(16, 32 \times 32), (32, 16 \times 16), (64, 8 \times 8)\}$ dimensions, and these values for MobileNetV2 are $\{(32, 32 \times 32), (96, 16 \times 16), (1280, 8 \times 8)\}$. Therefore, their feature filter embedding layers have 10×64 and 10×1280 dimensions respectively (CIFAR-10 has 10 classes). We use two upsampling blocks for CIFAR-10 experiments. These blocks are characterized by 3 parameters: number of their input features' channels, number of pass-through features' channels (input features from the U-Net's encoder), and number of their output channels [72]. These values are $\{(64, 32, 32), (32, 16, 16)\}$ and $\{(1280, 96, 96), (96, 32, 32)\}$ for upsampling layers of ResNet-56 and MobileNetV2 respectively. An upsampling layer, at first, upsamples a low-resolution feature map by a factor of 2 using $2D$-Convolution and Pixel Shuffle blocks [103]. Then, it concatenates upsampled features with pass-through features and applies three Bottleneck blocks [39] on them.

Finally, we use a $2D$-Convolution layer followed by three non-linearities that map the last upsampling layer's output to 3 values corresponding to predicted (c_z, c_t, σ) parameters for the output RBF kernel. We set the kernel size of the convolution filter to the upsampling layer's output spatial dimension (32). In addition, we set the number of output channels to 3. We use two different non-linear activation functions, namely one for calculating c_z, c_t

and the other for σ that we elaborate on them in Section S2.3.0.1.

ImageNet: Most of the details are similar to the CIFAR-10 experiments described above. We use ResNet-34, 50, 101 [39] and MobileNetV2 [98] to assess our model's performance on ImageNet [100]. Training on the full dataset is computationally intensive. We randomly select 0.1/0.02 of the official training set as our training/validation partitions and use the official validation set as our test set for our AEM model's training and evaluation. We train the predictor for 100 epochs with batch size 128 and the selector for 5 epochs with batch size 64. The optimization parameters are the same as CIFAR-10 experiments. The configuration $\lambda_1 = 1.0$ and $\lambda_2 = 0.0001$ showed convincing performance for all architectures. In ImageNet, we use the previously mentioned subset for pruning. During pruning, we optimize Eq. 2.10 for 100 epochs with the ADAM optimizer. γ_1 and γ_2 are the same as the CIFAR-10 setting. After pruning, we finetune all architectures for 100 epochs by using SGD with a momentum of 0.9. For ResNet models, we use a start learning rate of 0.1 and decay it to 0.01, 0.001, 0.0001 at epoch 30, 60, and 90. For MobileNet-V2, we use a start learning rate of 0.05 and a cosine annealing learning rate scheduler, as mentioned in the original paper [98]. We set the weight decay to 0.0001 for ResNet models and 0.00004 for MobileNetV2. We implement our method using PyTorch [101]. Given an input image with 3 channels and 224×224 spatial dimensions, ResNet-34, 50, and 101 calculate 5 scales with (number of channels, spatial dimensions) as follows:

- $\{(64,\ 56 \times 56),\ (64,\ 56 \times 56),\ (128,\ 28 \times 28),\ (256,\ 14 \times 14),\ (512,\ 7 \times 7)\}$ for ResNet-34,
- $\{(64,\ 56 \times 56),\ (256,\ 56 \times 56),\ (512,\ 28 \times 28),\ (1024,\ 14 \times 14),\ (2048,\ 7 \times 7)\}$ for

ResNet-50 and ResNet 101,

- $\{(32,\ 112 \times 112),\ (24,\ 56 \times 56),\ (32,\ 28 \times 28),\ (96,\ 14 \times 14),\ (1280,\ 7 \times 7)\}$ for MobileNetV2.

Thus, the embedding layer of their feature filter layer is 1000×512 for ResNet-34, 1000×2048 for ResNet-50 as well as ResNet-101, and 1000×1280 for MobileNetV2.

We use 3 upsampling blocks for these architectures. The values of (# input features' channels, # pass-through features' channels, # output channels) for upsampling layers are as follows:

- $\{(512,\ 256,\ 256),\ (256,\ 128,\ 128),\ (128,\ 64,\ 64)\}$ for ResNet-34.
- $\{(2048, 1024, 1024), (1024, 512, 512), (512, 256, 256)\}$ for ResNet-50 and ResNet-101.
- $\{(1280,\ 96,\ 96),\ (96,\ 32,\ 32),\ (32,\ 24,\ 24)\}$ for MobileNetV2.

Finally, we use a $2D$-Convolution with kernel size 56 and 3 output channels to map the last upsampling layer's outputs to 3 numbers for (c_z, c_t, σ).

S2.3.0.1 Proposed Output Nonlinearities for the Selector Model

Center's Coordinates Nonlinearity: As mentioned in Section 2.3.5, if we consider a two-axis coordinate system for an image's spatial dimensions and the system's origin being on the center of it, the values c_z, c_t can take any real values theoretically. However, we know that the salient part of the image is within its spatial dimensions, *e.g.*, $c_z, c_t \in [-16, 16]$ for CIFAR-10 images with 32×32 size. Therefore, we use `Tanh` non-linearity to ensure that the output values are in the range of image dimensions. In addition, to prevent the

vanishing gradients phenomenon in `Tanh`, we set its 'active' range being roughly equal to spatial dimensions of an input image, *i.e.*, we calculate c_z, c_t as:

- $c_z = 14 * \texttt{Tanh}(u_z/14) + 16$
- $c_t = 14 * \texttt{Tanh}(u_t/14) + 16$

We show this functional form in Fig. 2.6. As can be seen, when its input u lies in the interval $[-14, 14]$, the output lies in $[2, 30]$, which corresponds to the 28×28 frame into a 32×32 image. Moreover, when $u \in [-14, 14]$, the `Tanh` function is in its active form, which prevents the known vanishing gradient challenge with `Tanh`.

For ImageNet experiments, we use the following form to ensure that the center of predicted RBF is in the central 220×220 frame of a 224×224 image, and we show it in Fig. 2.8.

- $c_z = 108 * \texttt{Tanh}(u_z/108) + 112$
- $c_t = 108 * \texttt{Tanh}(u_t/108) + 112$

Expansion Parameter's (σ) Nonlinearity: the parameter σ in Eq. (2.22) determines the degree of RBF expansion on an input image's surface and should be a positive real number. Therefore, we use 'Softplus' non-linearity to calculate it, which is a smooth approximation of `ReLU` [104] and is always positive. It is calculated with the formula $SoftPlus(u) = \log(1 + \exp(u))$ and is shown in the Fig. 2.7. We use this function to calculate the expansion parameter for both sets of experiments on CIFAR-10 and ImageNet.

Selector Network's Training Initialization: We use the standard network initialization implemented in PyTorch [101] to train a decoder of a selector's U-Net. However, this

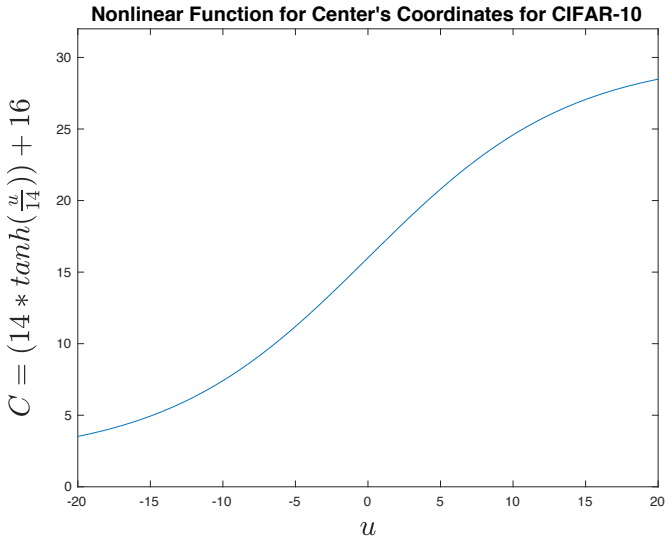


Figure 2.6: Our proposed nonlinear function to calculate the center's coordinates of a predicted RBF Kernel of the selector for the CIFAR-10 dataset.

initialization makes the output values of the output $2D$-convolution layer close to zero at the beginning of training. As a result, the output σ will be close to zero, and it may cause instability in the starting iterations of training. To prevent such instability, we empirically found that setting the $2D$-convolution layer's bias weight corresponding to σ to be about a third of the input image's spatial dimension can make the model's convergence faster and training more stable. Thus, we initialize the bias weight to be 10 for CIFAR-10 experiments and 80 for ImageNet ones.

S2.3.1 Gumbel-Sigmoid Reparameterization Strategy for Training Selector Model

We use Eq. (2.8) as the objective to train our selector model. Thus, we need to minimize the expectation on masks that the selector model parameterizes their distribution. Empirically, we use Monte-Carlo sampling and sample one mask for each input image x.

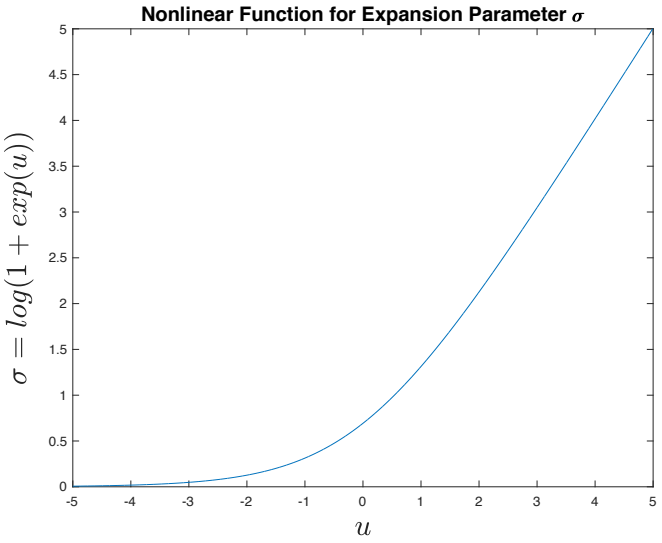


Figure 2.7: Our proposed nonlinear function to calculate the expansion parameter of a predicted RBF Kernel of the selector for the CIFAR-10 dataset.

However, sampling is not a differentiable operation. Hence, it is impossible to train the selector's parameters by optimizing them using backpropagation schemes when we directly sample from the predicted distribution. A workaround to this problem is to replace non-differentiable sampling from a categorical distribution with a differentiable sampling from the Gumbel-Sigmoid distribution [91, 92]. In summary, the binary mask can be generated by using the following function with the Gumbel-Sigmoid trick:

$$m_{i,j} = \frac{1}{1 + \exp\Big(-\frac{\log(f_{BP}(i,j;c_z,c_t,\sigma)) + g_j}{\tau}\Big)} \tag{2.24}$$

such that g_i values are sampled from the Gumbel distribution, and τ is called the 'temperature' parameter that determines 'sharpness' of the sample. Low τ values result in samples close to Bernoulli distribution (binary), but higher τ values make the output distribution more similar to uniform. We set $\tau = 1$ for training our selector model for all experiments.

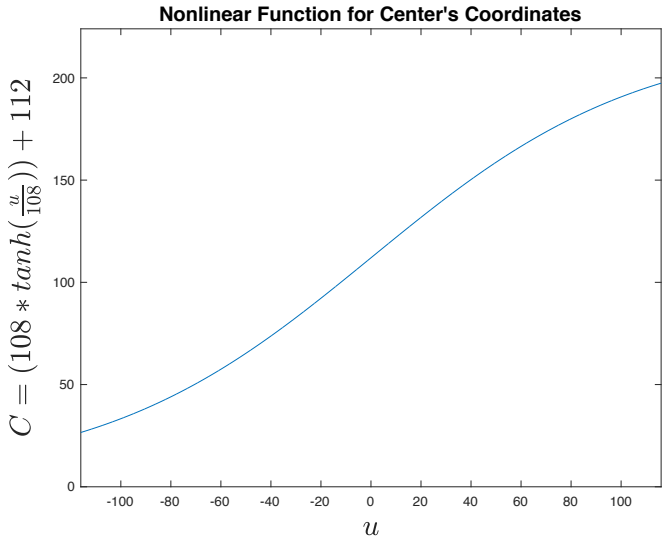


Figure 2.8: Our proposed nonlinear function to calculate the center's coordinates of a predicted RBF Kernel of the selector for the ImageNet dataset.

We then can insert Eq. (2.24) into Eq. (2.8) to optimize the selector model.

S2.3.2 More Details about Pruning

Similarly, we also use Gumbel-Sigmoid trick to characterize each channel:

$$v_l = \frac{1}{1 + \exp\left(-\frac{\theta_{g_l} + g_j + b}{\tau}\right)} \tag{2.25}$$

where θ_{g_l} is the parameters for l-th layer, and $\mathbf{v} = [v_1, \cdots, v_L]$. We also insert a constant b for starting pruning from the whole model. In experiments, we set $b = 3$ and $\tau = 0.4$ for pruning channels. To achieve pruning, we multiply v_l to its corresponding feature map $\mathcal{F}_l$ after activation functions:

$$\hat{\mathcal{F}}_l = v_l \odot \mathcal{F}_l \tag{2.26}$$

where $\odot$ is element-wise product, and v_l is first expanded to the same size of $\mathcal{F}_l$.

In practice, we let $\mathcal{R}_{res}(x, y) = \log(\max(x, y)/y)$, which effectively push $\mathcal{R}_{res}$ to 0. Other regression loss functions like MSE and MAE can not achieve similar functions, and they often fail when applied to compact models like MobileNet-V2 [98].

S2.3.3 More Samples of our AEM's Predictions

We show visual examples of our proposed AEM's predictions on ImageNet and CIFAR-10 in the following pages. In each row from left to right, we show an input image, the predicted distribution of explanatory masks over the input, the predicted distribution shown over the input image, a mask sampled from the predicted distribution, and the input masked by the sampled mask. ImageNet images have higher resolution than CIFAR-10 ones. Thus, their sampled masks look more coherent than CIFAR-10 images. However, we can see that our selector model almost always puts the mode of its RBF kernel on the salient part of the input.

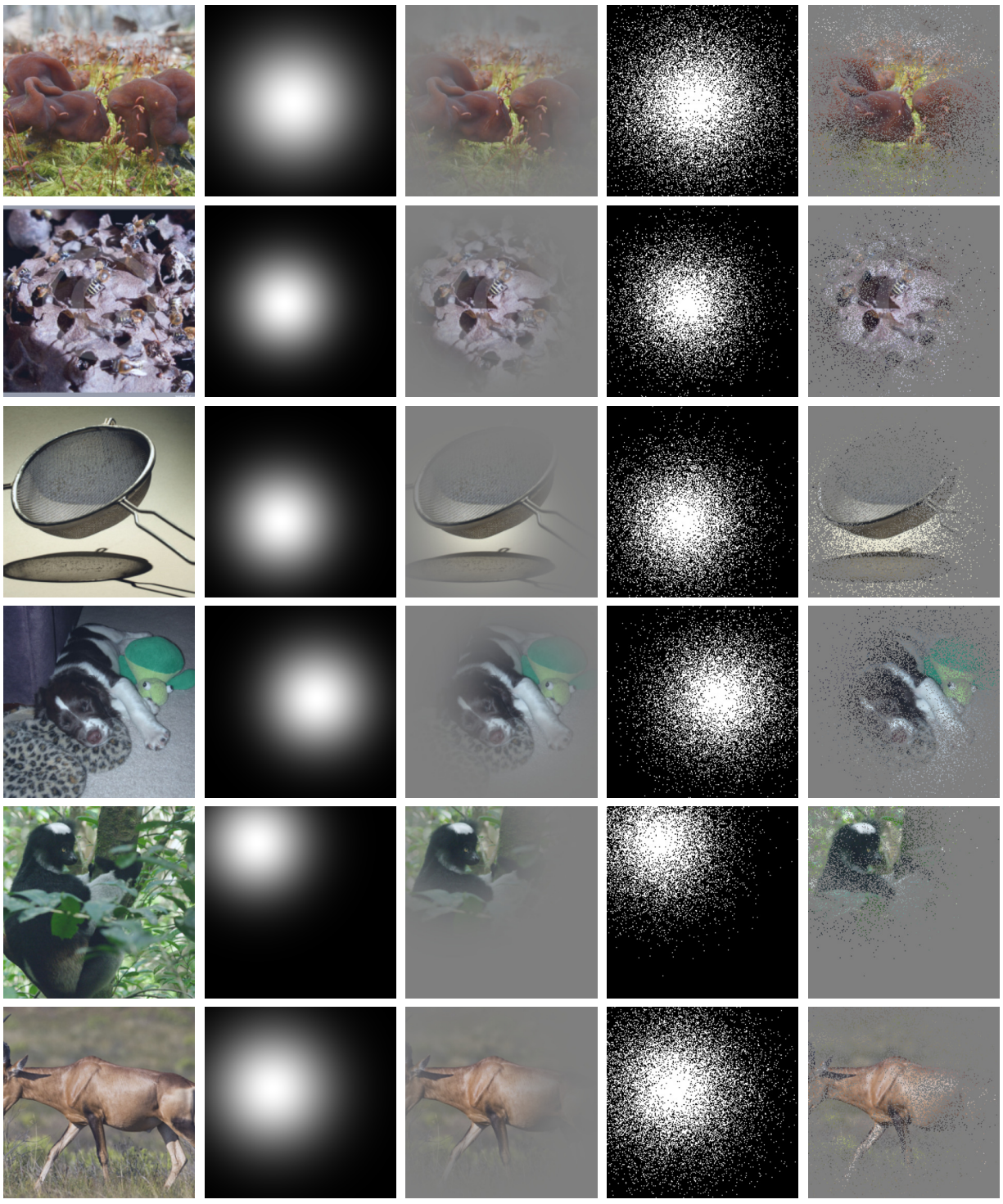


Figure 2.9: **ImageNet Examples. Columns from left to right:** input image, distribution over explanatory masks predicted by selector, predicted distribution shown over input, a sampled mask from the predicted distribution, and input image masked by the sampled mask. **Class of input images from top to bottom:** 'Gyromitra', 'Honeycomb', 'Strainer', 'English springer', 'Indri brevicaudatus', 'Hartebeest'.

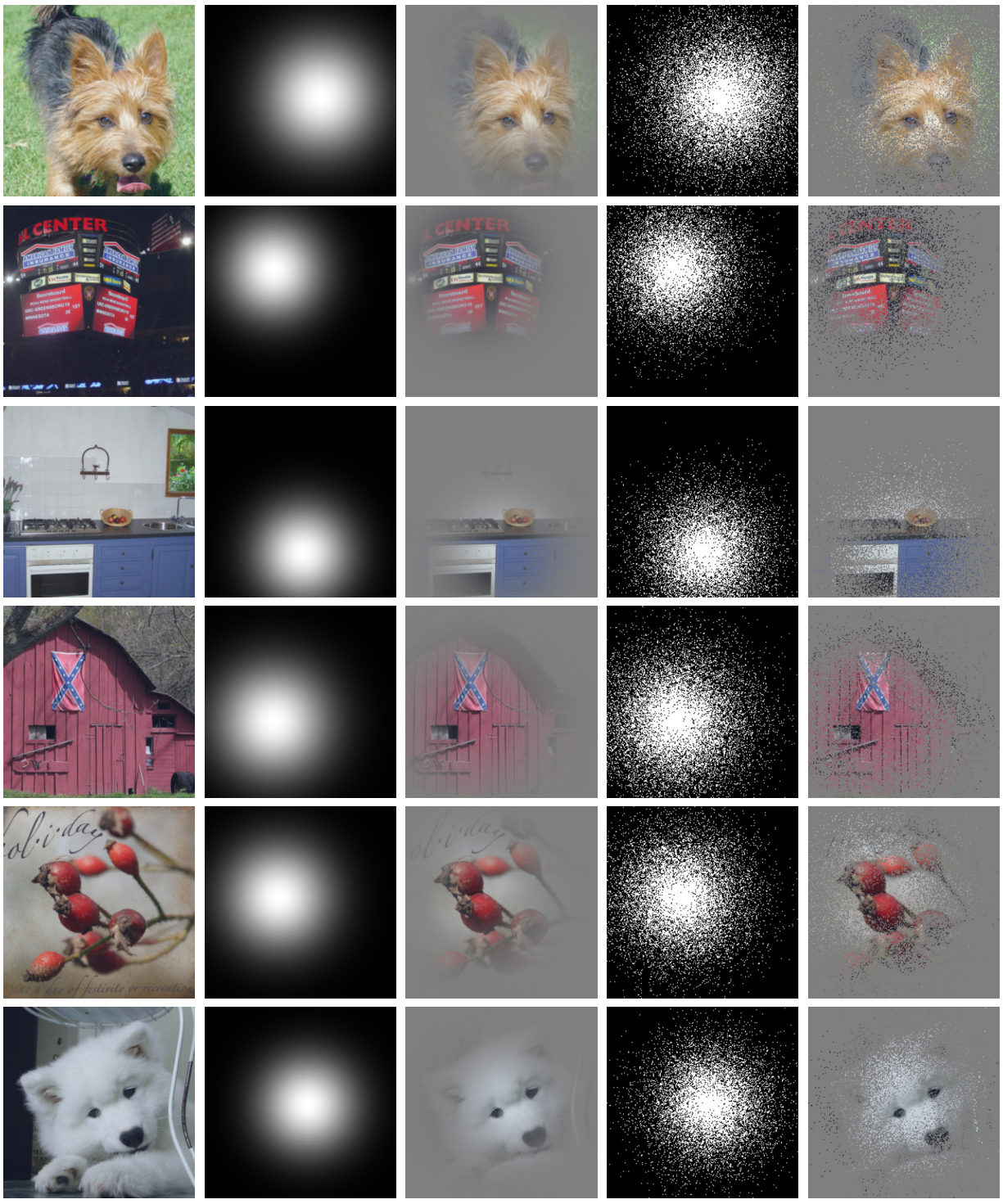


Figure 2.10: **ImageNet Examples. Columns from left to right:** input image, distribution over explanatory masks predicted by selector, predicted distribution shown over input, a sampled mask from the predicted distribution, and input image masked by the sampled mask. **Class of input images from top to bottom:** 'Australian terrier', 'Scoreboard', 'Microwave oven', 'Barn', 'Rosehip', 'Samoyed'.

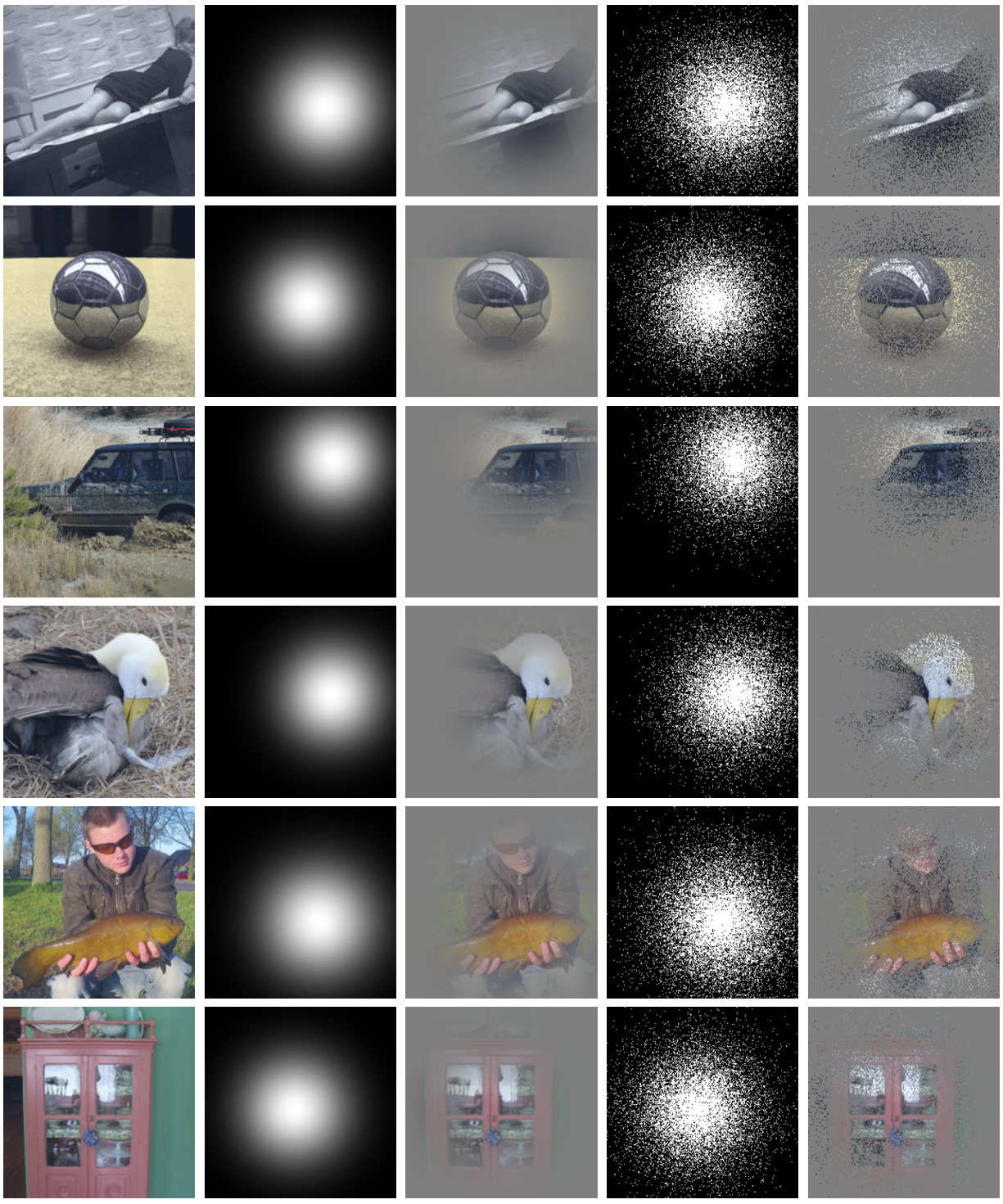


Figure 2.11: **ImageNet Examples. Columns from left to right:** input image, distribution over explanatory masks predicted by selector, predicted distribution shown over input, a sampled mask from the predicted distribution, and input image masked by the sampled mask. **Class of input images from top to bottom:** 'Miniskirt', 'Soccer ball', 'Jeep', 'Albatross', 'Tench', 'China cabinet'.

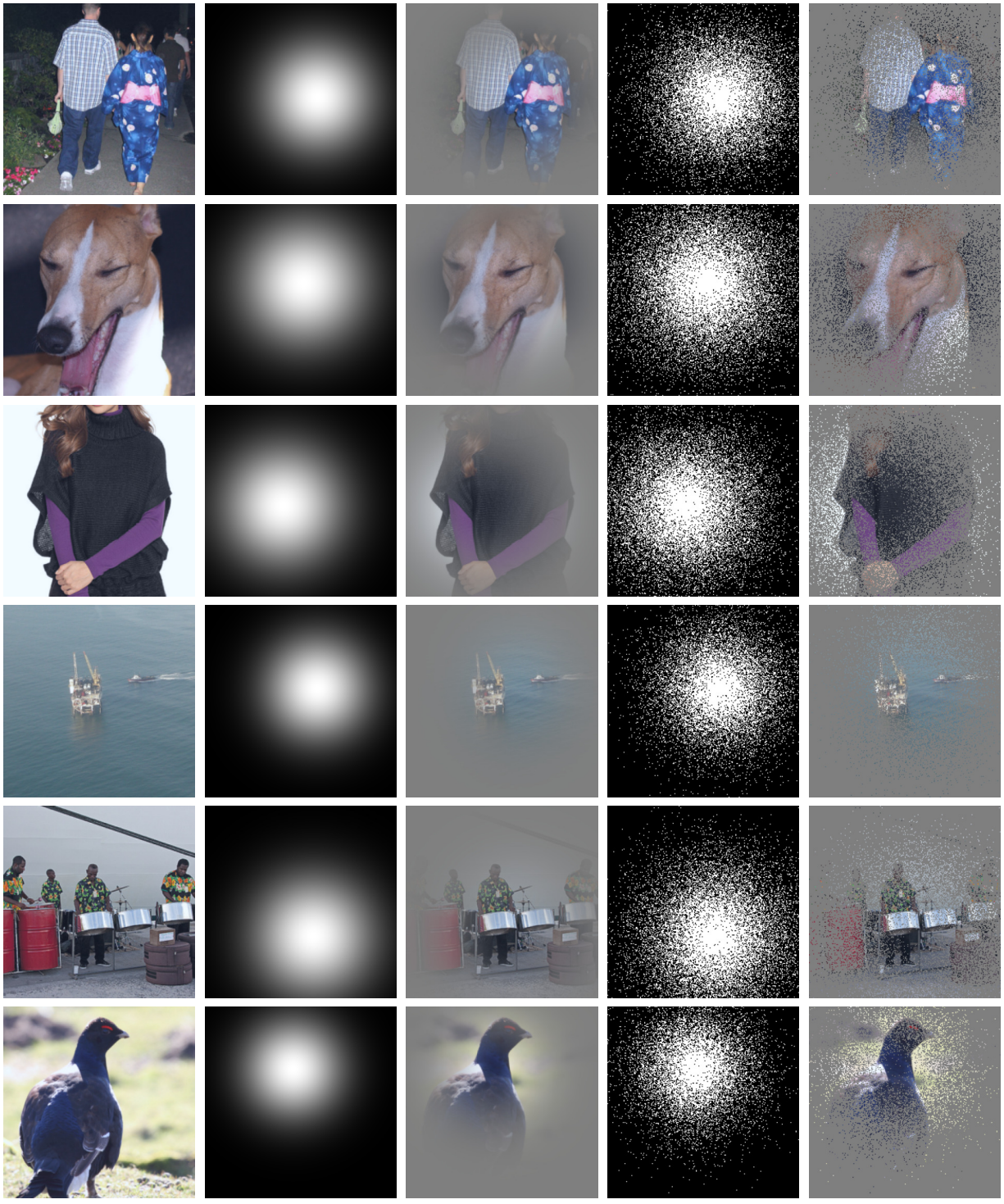


Figure 2.12: **ImageNet Examples. Columns from left to right:** input image, distribution over explanatory masks predicted by selector, predicted distribution shown over input, a sampled mask from the predicted distribution, and input image masked by the sampled mask. **Class of input images from top to bottom:** 'Kimono', 'Whippet', 'Poncho', 'Drilling Platform', 'Steel Drum', 'Black Grouse'.

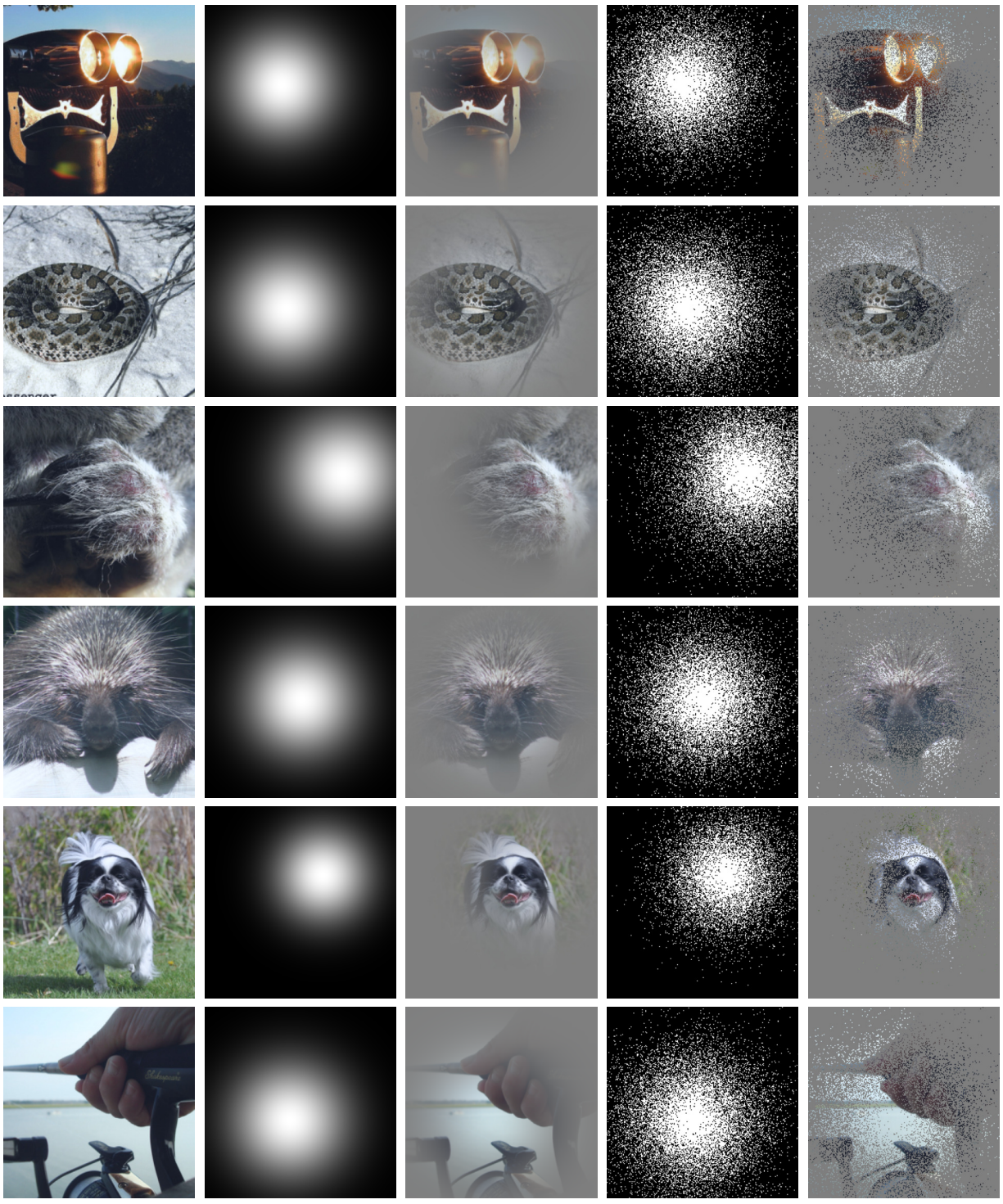


Figure 2.13: **ImageNet Examples. Columns from left to right:** input image, distribution over explanatory masks predicted by selector, predicted distribution shown over input, a sampled mask from the predicted distribution, and input image masked by the sampled mask. **Class of input images from top to bottom:** 'Binoculars', 'Horned viper', 'Native bear', 'Hedgehog', 'Japanese spaniel', 'Reel'.

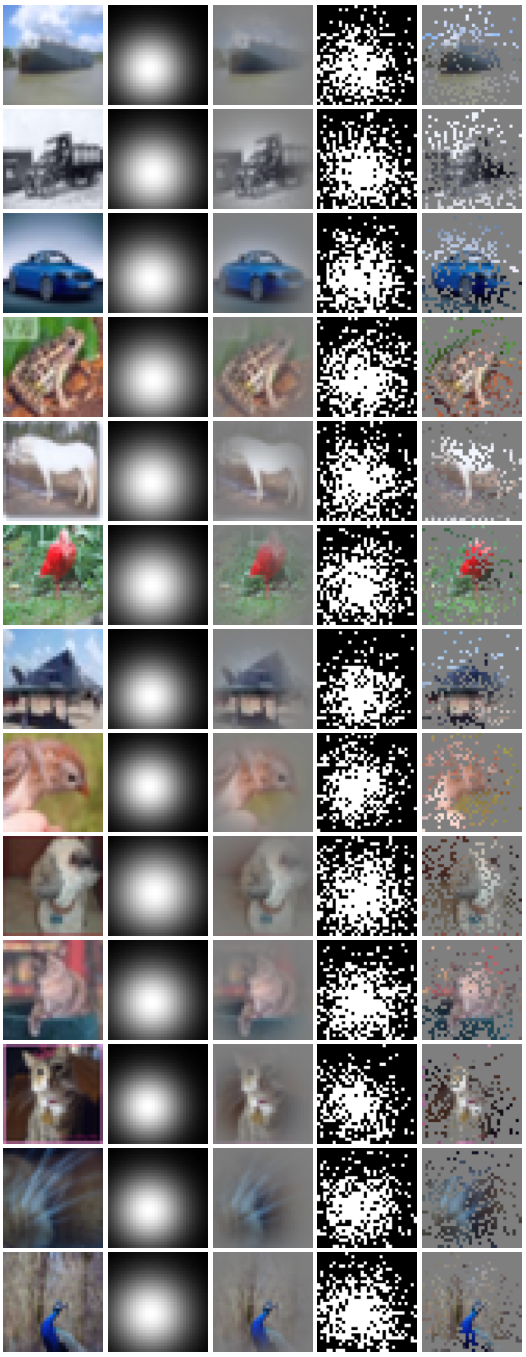


Figure 2.14: **CIFAR-10 Examples. Class of input images from top to bottom:** 'Ship', 'Truck', 'Automobile', 'Frog', 'Horse', 'Bird', 'Airplane', 'Bird', 'Dog' 'Cat', 'Cat', 'Cat', 'Bird'.

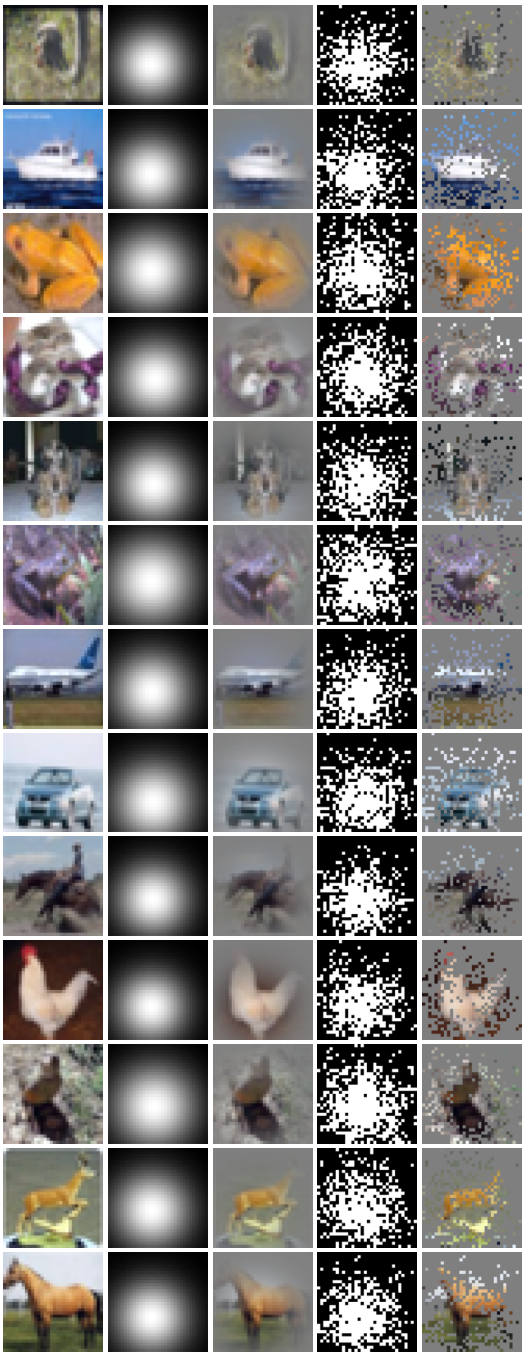


Figure 2.15: **CIFAR-10 Examples. Class of input images from top to bottom:** 'Bird', 'Ship', 'Frog', 'Cat', 'Dog', 'Frog', 'Airplane', 'Automobile', 'Horse' 'Bird', 'Bird', 'Deer', 'Horse'.

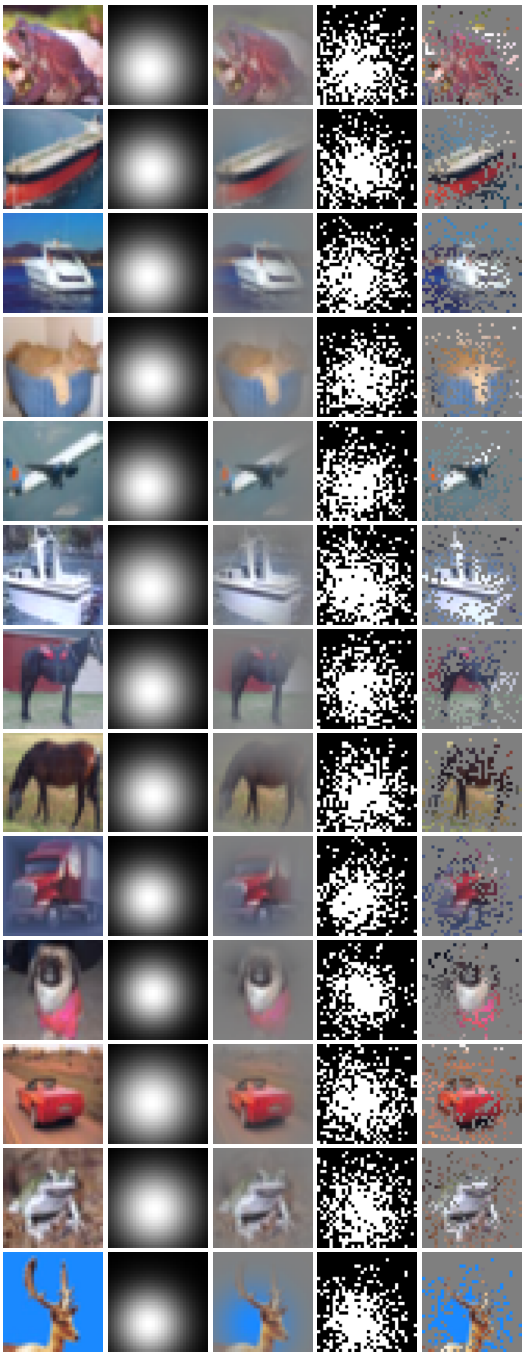


Figure 2.16: **CIFAR-10 Examples. Class of input images from top to bottom:** 'Frog', 'Ship', 'Ship', 'Cat', 'Airplane', 'Ship', 'Horse', 'Horse', 'Truck', 'Dog', 'Automobile', 'Frog', 'Deer'.

Figure 2.17: **CIFAR-10 Examples. Class of input images from top to bottom:** 'Dog', 'Airplane', 'Horse', 'Automobile', 'Horse', 'Ship', 'Ship', 'Automobile', 'Cat', 'Airplane', 'Ship', 'Airplane', 'Dog'.

Chapter 3: Efficient Learning of Kernel Sizes for Convolution Layers of CNNs

3.1 Introduction

Convolutional neural network (CNNs) have consistently shown top performance in image classification tasks in the last decade [38, 39, 105, 106, 107]. They consist of multiple layers of convolution operators with learnable weights that, combined with pooling layers and strided convolutions, downsample an input image gradually. Such a design blended with normalization and nonlinear layers enables them to make an information bottleneck to extract low-resolution task-relevant information from high-resolution low-level details such as pixels' intensities. Despite their success, developing new high-accuracy architectures is a complex task involving numerous hyperparameters. One of the critical design choices that directly affects the model's performance is the size of the kernels for convolutional layers.

A prominent design for CNNs is to use numerous layers of blocks with predetermined kernel sizes [38, 39, 106, 108]. However, by doing so, the network's predictions are not differentiable $w.r.t$ kernel sizes. Thus, finding the optimal setting for them (given a desired number of parameters) becomes a hyperparameter tuning problem that can be solved using cross-validation, casting it as a constrained discrete optimization problem, or leveraging

heuristic design tricks. Cross-validation gets infeasible as the number of combinations grows exponentially *w.r.t* number of layers and possible kernel sizes. The constrained optimization problem should be solved with techniques such as Neural Architecture Search (NAS) [109] that require powerful computational resources that may not be available for low-resourced applications. Although using heuristic tricks [107, 108, 110] to design kernel sizes has shown remarkable results, their intuitions are mainly based on an architecture's performance on ImageNet [111], which is shown does not necessarily translate to a more effective model on other domains, with even negative correlations between the metrics [112, 113] in some cases.

Learning a model's kernel sizes jointly with its weights during training is a promising direction to overcome the mentioned challenges. A group of methods take a set of basis functions (*e.g.,* Delta-Diracs [114] or Gaussian functions [115]) and learn dilation factors for these functions to alter a kernel size. However, dilation limits the learned kernels' bandwidth and hurts the model's performance. Recently, FlexConv [116] proposed to learn high-bandwidth kernels by defining them as a product of a predicted kernel by a neural network (MLP) and a Gaussian kernel with learnable parameters. For each convolution layer, its corresponding MLP predicts a kernel with the same spatial dimensions as the input image. Then, the predicted kernel is multiplied by the Gaussian kernel. Finally, the resulting kernel is cropped according to a threshold. Nevertheless, doing so has several key drawbacks. First, if we denote the input size as N, FlexConv needs to compute N^2 MLP forward passes for each convolution layer in each iteration. Thus, its training time grows quadratically *w.r.t* input size, making it highly inefficient even for datasets with moderate input size such as STL-10 [117]. It also makes using FlexConv prohibitively expensive for

very deep architectures. Secondly, cropping by thresholding wastes the computations of the MLP and is an inefficient way to determine kernel sizes.

We propose a new kernel size learning method that has significantly lower computational requirements than previous methods and, at the same time, outperforms them on different tasks. We develop a model called *size predictor* that learns to determine optimal kernel sizes for a CNN. During training, the size predictor collaborates with a hypernetwork [118] named *kernel predictor*. In each iteration, at first, the size predictor decides the size of kernels for the CNN. Then, the kernel predictor adaptively predicts kernel weights given the specified sizes by the size predictor. This scheme prevents wasting computations by the kernel predictor that is happening in FlexConv after cropping. Moreover, our method needs orders of magnitude lower forward passes during training, making our method dramatically faster than FlexConv. Finally, the weights of both models are optimized to minimize the classification objective while keeping the parameter count of the CNN at the desired number. By doing so, our method can find a high-performance configuration for kernel sizes of a CNN with a small fraction of training epochs needed to train it from scratch. After finding a proper configuration, we discard the size predictor as well as kernel predictor and train the resulting CNN with the same training settings as the original CNN. Therefore, our method is able to discover and train a high accuracy setting for the kernel sizes with much lower computations than other methods. Our experimental results on MNIST, CIFAR-10, STL-10, and ImageNet-32 demonstrate that our method consistently shows better accuracy *vs.* training time trade-off compared to other kernel size learning methods, even outperforming them in both metrics in most cases. We summarize our contributions as follows:

- We introduce a novel kernel size learning method for CNNs that requires a significantly lower computational budget than existing methods.
- We design a size predictor model that is trained jointly with a kernel predictor model to predict optimal kernel sizes of a classifier. Training these models requires only a small fraction of the training epochs of the CNN.
- Our experiments on several benchmark datasets illustrate that our method can achieve more competent architectures in much lower training epochs than other kernel size learning methods, providing an efficient and effective solution for the kernel size learning problem.

The contents of this chapter are based on our work [119] published in AAAI 2023.

3.2 Related Works

Innovation of novel performant CNN architectures for different applications is a complicated task consisting of numerous factors. One of the crucial factors is determining kernel sizes for convolution layers. Most of the proposed ideas design architectures consisting of multiple layers with fixed preset sizes. Early works [38, 39, 105, 106] use blocks with kernels of sizes 1-7px. The main downside of using fixed architectures is that the training objective will not be differentiable *w.r.t* kernel sizes. Thus, an optimal setting should be found by cross validation, solving a constrained discrete optimization, or developing heuristic design tricks. However, the number of potential architectures increases exponentially *w.r.t* the number of layers and kernel sizes range, making cross validation impractical. The high

number of possible architectures also translates into a vast search space for NAS methods [109, 120, 121, 122] used to solve the constrained discrete optimization problems for finding optimal kernel sizes. Thus, NAS methods require an extreme computation budget to find the desired configuration. Using heuristics to design kernel sizes has shown significant results recently [107, 108, 110, 123]. RepLKNet. [108] introduces five guidelines to design CNNs with large kernel sizes up to 31px. It uses small kernels parallel to large ones and achieves comparable results to Swin Transformer [124]. ConvNeXt [107] follows the design of Swin Transformer to develop large kernel CNNs. However, these models' intuitions and design tricks mainly rely on improving the Top-1 accuracy of the resulting architectures on ImageNet [111]. Thus, their superior accuracy on ImageNet may not reflect on other domains, as shown by recent studies [112, 113].

Learning kernel sizes during training can alleviate the weaknesses for predesigned architectures. Deformable ConvNets [114] learn offsets for each pixel of a filter to obtain deformable kernels. Also, [115, 125] model kernels as a learned combination of Gaussian derivative filters with a tunable variance parameter to control its effective size. OFS-CNN [126] use learnable padding operations, and [127, 128, 129] model filters as learnable dilated Gaussian functions to learn kernel sizes. Yet, dilated basis functions/kernels result in kernels with limited bandwidths, restricting the final model's performance. Recently, FlexConv [116], inspired by continuous kernel convolutions [130, 131, 132, 133, 134, 135], proposed to model kernels of a CNN as a product of a kernel predicted by a continuous function (implemented by a MLP) and a learned Gaussian kernel. They train a separate MLP and Gaussian kernel for each kernel. The MLP predicts kernels with the same resolution of an input. Then, the Gaussian kernel is multiplied with it, and the resulting

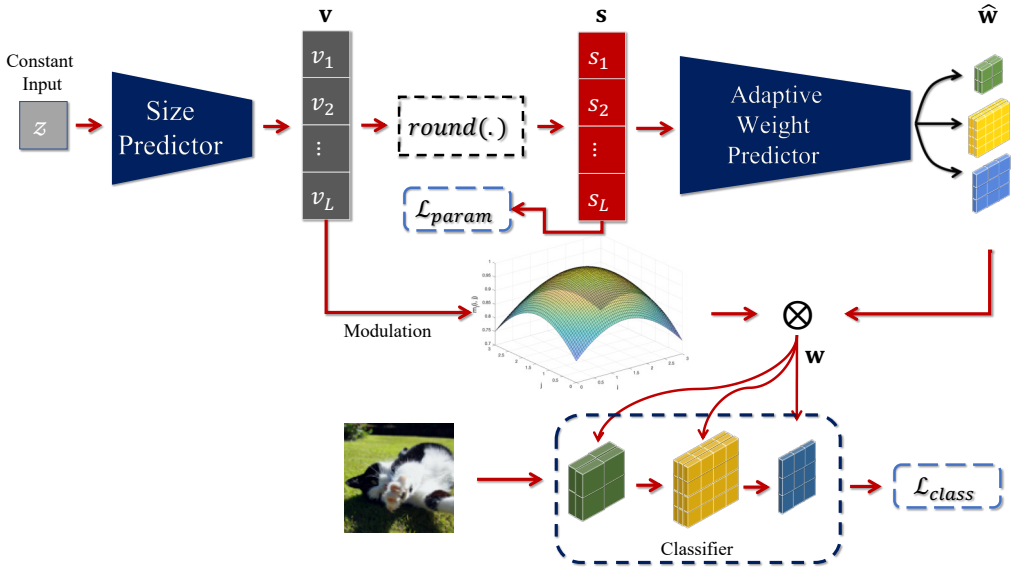


Figure 3.1: Overview of our method. Our size predictor model learns to predict the kernel sizes for the classifier. It predicts *soft* kernel sizes $\mathbf{v}$ that are rounded to integer values. Then, our adaptive weight predictor model predicts optimal kernel weights $\hat{\mathbf{w}}$ given the predicted sizes. We modulate the predicted weights using masks m_l parameterized by the soft sizes $\mathbf{v}$ to make the resulting weights $\mathbf{w}$ differentiable $w.r.t$ the size predictor's weights. Finally, the weights $\mathbf{w}$ are used as the kernel weights of the classifier, and the training is guided by the classification objective ($\mathcal{L}_{class}$) and the parameters budget loss ($\mathcal{L}_{param}$).

kernel is cropped using a tunable threshold. Nevertheless, such scheme requires multiple forward passes and the cropping method wastes computations. Our method can effectively overcome these limitations such that our kernel predictor adaptively predicts kernels with sizes predicted by the size predictor. Thus, it only needs forward passes proportional to predicted sizes and not the size of input images, thereby making our method much more efficient compared to FlexConv.

3.3 Methodology

We elaborate on different components of our method in the following sub-sections. We show an overview of our approach in Fig. 3.1.

3.3.1 Size Predictor

We develop a size predictor model that dynamically predicts kernel sizes during training. We implement our size predictor with a GRU [136] unit followed by feed forward layers. In each step, it takes a fixed input z and predicts the kernel sizes as follows:

$$\mathbf{v} = \sigma(f_{sp}(z; \theta_{sp})) \tag{3.1}$$

$$\mathbf{s} = round(\mathbf{v}) \tag{3.2}$$

f_{sp} represents the size predictor, and $round(\cdot)$ rounds the input to its nearest integer. σ is the nonlinearity that we use to ensure the predicted kernel sizes are in the designed range. We implement it with $\sigma(x) = k * tanh(x/\tau + b) + k + 1$. With such a design, predicted sizes in $\mathbf{s}$ will be in $[1, 2k + 1]$. We set τ to enlarge the active region of $tanh$ and prevent the vanishing gradients problem. We choose b such that the initially predicted sizes be close to the original design of the architectures. Given a predicted size vector $\mathbf{s}$, the number of parameters for the resulting model will be:

$$p_{pr} = \sum_{l=1}^{L} c_l \times c_{l-1} \times s_l \times s_l \tag{3.3}$$

where c_l is the number of output channels for the l-th convolution layer. To regulate the size predictor to fix the parameter count of the classifier to the desired number p_d, we use the following objective:

$$\mathcal{L}_{param} = \begin{cases} log(\frac{p_{pr}}{p_d}) & p_{pr} > p_d \\ log(\frac{p_d}{p_{pr}}) & \text{Otherwise} \end{cases} \tag{3.4}$$

The regularization in Eq. 3.4 has also been utilized for network pruning [137, 138]. As the $round(\cdot)$ function is not differentiable, we calculate the gradients of $\mathcal{L}_{param}$ $w.r.t$ θ_{sp} using the Straight Through Estimator [139].

3.3.2 Kernel Predictor

We use a kernel predictor that estimates proper weights for the CNN given the kernel sizes determined by the size predictor. Our motivation for such a design is that the kernel sizes change dynamically during training, and using a fixed set of trainable parameters for kernel weights is challenging. Thus, we utilize a model that can effectively adjust its output size given input sizes. We implement our kernel predictor with a GRU [136] block for each convolution layer in the model. We represent the number of input/output channels of the l-th convolution layer with c_{l-1}/c_l. Accordingly, we set the output size of its corresponding GRU being $c_{l-1} \times c_l$. Now, if the predicted size for this layer is s_l, its GRU can adaptively predict the weights for it by performing $s_l \times s_l$ time step forward passes, predicting the weights for each spatial dimension of the kernel one at a time:

$$\hat{w}_l = f^l_{kp}(z'_l; s_l; \theta^l_{kp}) \tag{3.5}$$

z'_l is the constant input of the GRU. Such a design remarkably reduces the computation

needed to predict a kernel's weights compared to the continuous convolution kernel of FlexConv [116]. The reason is that our kernel predictor needs to compute s_l^2 time steps (forward passes) to predict a kernel's weights, but FlexConv requires N^2 forward passes (N is the size of the input to the CNN), regardless of the final size of the kernel, making it extremely inefficient even for datasets with an average input size such as STL-10 [117].

3.3.2.1 Modulating Kernel Size in the Predicted Weights

Despite the efficiency of our kernel predictor in determining a kernel's weights, its predictions are neither differentiable *w.r.t* s_l nor v_l as s_l only determines the number of time step calculations. Thus, naive usage of our kernel predictor does not provide any feedback for the size predictor model during training when using gradient-based optimization schemes.

We propose modulating the predicted kernel size into the predicted weights as a workaround. Inspired by adaptive attention span [140], we multiply a mask m_l parameterized by the soft kernel size v_l to the predicted weights $\hat{w}_l$. We design m_l as:

$$\begin{aligned} m_l(i,j) &= exp(-\frac{(i-c_l)^2 + (j-c_l)^2}{v_l^2}) \\ c_l &= v_l/2, \quad i,j \in \{1,2,\cdots s_l\} \end{aligned} \tag{3.6}$$

Finally, we apply the mask m_l to the predicted weights $\hat{w}_l$ and use the resulting kernel w_l in the classifier for training:

$$w_l = m_l \odot \hat{w}_l \qquad l \in \{1,\cdots,L\} \tag{3.7}$$

where $\odot$ means element-wise product. We use our exponential mask instead of the piece-wise linear one proposed by [140] as it makes all of the weights of the resulting kernel differentiable *w.r.t* v_l, making the gradient flow [141] for v_l easier. In contrast, in the latter, only the weights at the edges have such property, limiting the flow. Our mask attenuates the edges of kernels and almost keeps the middle parts intact, encouraging the model not to predict unnecessarily large kernels.

We train the size predictor, kernel predictor, and the parameters of the CNN to minimize the classification objective (Cross-Entropy loss) while keeping the number of parameters of the CNN at the desired budget. Thus, our final objective is:

$$\min_{\theta_{sp},\theta_{kp},\theta_c} \mathcal{L} = \mathcal{L}_{class}(\theta_{sp},\theta_{kp},\theta_c) + \lambda \mathcal{L}_{param}(\theta_{sp}) \tag{3.8}$$

θ_c represents the parameters of the CNN, and λ is a hyperparameter. We show in our experiments that our method only needs a small fraction of the training epochs of a CNN to train a size predictor for it. After that, we discard the size predictor as well as kernel predictor models and use the learned kernel sizes for the CNN to train it with its default training settings. We summarize our training algorithm in Alg. 1.

3.4 Experiments

In this section, at first, we provide our experimental setup. Then, we evaluate our proposed method against baselines on four different tasks. Finally, we explore the behavior of our model through two ablation studies.

Algorithm 1 Our Kernel Size Learning Algorithm

Input: Training dataset $\mathcal{D} = \{(x_i, y_i)\}_{i=1}^{D}$; CNN $f_c(.;\theta_c)$; size predictor $f_{sp}(\cdot;\theta_{sp})$; size predictor nonlinearity parameters $\{k, b, \tau\}$; kernel predictor $f_{kp}(\cdot;\theta_{kp})$; size learning epochs E_s; training epochs E_T.

Output: CNN with learned kernel sizes and weights.

`/* Learning Kernel Sizes of the CNN */`

for $e := 1$ *to* E_s **do**

1. Sample a mini-batch $(\mathbf{x}, \mathbf{y})$ from $\mathcal{D}$.
2. Calculate kernel sizes $\mathbf{s}$ and soft predictions $\mathbf{v}$ of the size predictor (Eqs. 3.1, 3.2) using $f_{sp}(\cdot;\theta_{sp})$ and $\{k, b, \tau\}$.
3. Calculate $\hat{w}_l$ using the predicted sizes $\mathbf{s}$ and the kernel predictor $f_{kp}(\cdot;\theta_{kp})$ according to Eq. 3.5
4. Compute the modulating masks m_l using soft predictions $\mathbf{v}$ of the size predictor in Eq. 3.6.
5. Apply the masks m_l to $\hat{w}_l$ in Eq. 3.7 and use the resulting w_l as the weights of the CNN.
6. Use the sampled $(\mathbf{x}, \mathbf{y})$ to calculate the classification loss $\mathcal{L}_{class}$ of the CNN.
7. Calculate the parameters budget loss $\mathcal{L}_{param}$ using $\mathbf{s}$ and Eq. 3.4
8. Compute $\mathcal{L}$ in Eq. 3.8, backpropagate the gradients $w.r.t\ \theta_{sp}, \theta_{kp}, \theta_c$, and update them.

end

`/* Train the CNN with new kernel sizes */`

9. Determine the optimal kernel sizes of the CNN using the trained size predictor.
10. Initialize the CNN using the predicted kernel sizes.
11. Train the CNN for E_T epochs with its standard settings.

Return: Trained CNN with new kernel sizes.

3.4.1 Experimental Setup

Datasets: We use four datasets in our experiments to analyze our model: MNIST [142], CIFAR-10 [5], STL-10 [117], and ImageNet-32 [143].

Architectures: We mainly focus on finding kernel sizes for ResNet [39] architectures with depths in $\{20, 26, 32, \cdots, 56\}$ in our experiments. We explore our model's performance for Wide-ResNet [144] for CIFAR-10 experiments as well. For all models, we learn the kernel sizes for the residual layers and keep the first input 3×3 convolution kernel.

Training Details: For all architectures and datasets except ImageNet, we train the size

predictor and kernel predictor models for 10 epochs. We do so for ImageNet-32 experiments for 5 epochs. Remarkably, we found that training our size predictor and kernel predictor models does not need hyperparameter tuning across architectures and datasets. We use the AdamW optimizer [145] for all models for the kernel size learning stage. We use a learning rate of 0.0001 with a weight decay of 0.001 for the size predictor and the CNN's parameters (the ones other than convolution kernels such as Batch Normalization [146] and Fully connected layers). For the kernel predictor, we utilize a learning rate of 0.001 and the same weight decay. Then, we train the resulting CNN with the training settings of the original model following the settings described in FlexConv [116]. For all experiments, we set the size predictor nonlinearity parameters to be $(k, \tau, b) = (3, 3, -0.35)$ so that our possible size range will be $[1, 7]$. In addition, the initial values of predicted soft kernel sizes will be close to 3, which is the original design for ResNet [39] and Wide-ResNet [144] architectures. We also set the parameter $\lambda = 2$ for the parameters budget loss in Eq. 3.8.

Evaluation Metrics: We use the accuracy of the models/training time per epoch to compare the performance/efficiency of the models, following FlexConv [116]. We calculate the time per epoch of our model as the average of times for training epochs of the stage for learning kernel sizes as well as training the resulting CNN from scratch. We call our method EffConv-X/Wide-EffConv-X-Y, representing the ResNet-X/Wide-ResNet-X-Y model with the depth X, widen factor Y, and kernel sizes learned by our method. We use a Lambda machine with 8 NVIDIA RTX A5000 GPUs with 256 CPU cores for our experiments. We provide other details of our experiments in supplementary S3.1.

[1]The number is provided by the authors by correspondence.

Model	Size	Accuracy	Time (Sec / Epoch)
Efficient-CapsNet [147]	0.16M	99.74 ± 0.0002	-
Network in Network [148]	N/A	99.53	-
VGG-5 [149]	3.65M	99.72	-
FlexNet-16 [116]	0.67M	99.7 ± 0.0	∼ 205[1]
EffConv-20 (Ours)	0.66M	99.80 ± 0.01	35.86

Table 3.1: Results on the MNIST dataset.

Model	Size	Accuracy	Time (Sec / Epoch)
N-JetNet-ALLCNN [115]	0.66M	89.91 ± 0.03	-
N-JetNet-CIFARResNet32 [115]	0.52M	92.28 ± 0.26	-
DCN-σ^{ji} [125]	0.47M	89.7 ± 0.3	-
CKCNN-7 [135]	0.70M	71.7	-
CKCNN-16 [135]	0.63M	72.1 ± 0.2	68
$\text{CKCNN}_{\text{MAGNet}}$-16 [135]	0.67M	86.8 ± 0.6	102
FlexNet-3 [116]	0.27M	90.4 ± 0.2	-
FlexNet-5 [116]	0.44M	91.0 ± 0.5	-
$\text{FlexNet}_{\text{Gabor}}$-16 [116]	0.67M	91.9 ± 0.2	161
FlexNet-16 [116]	0.67M	92.2 ± 0.1	127
CIFARResNet-44 [39]	0.66M	92.83	22
CIFARResNet-44 w/ CKConv (k=3)	2.58M	86.1 ± 0.9	-
CIFARResNet-44 w/ FlexConv	2.58M	81.6 ± 0.8	-
EffConv-20 (Ours)	0.20M	91.78 ± 0.16	
EffConv-20 (Ours)	0.50M	92.30 ± 0.11	
EffConv-20 (Ours)	0.66M	92.35 ± 0.05	17.77
EffConv-32 (Ours)	0.66M	92.99 ± 0.02	23.82
EffConv-44 (Ours)	0.66M	93.35 ± 0.17	27.65
Wide-EffConv-28-1	0.66M	93.14 ± 0.23	15.99

Table 3.2: Results on the CIFAR10 dataset.

3.4.2 Results

We report a mean and standard deviation of 3 runs of our model with different random seeds for all experiments.

3.4.3 MNIST

We summarize the results on the MNIST dataset in Tab. 3.1. As can be seen, our method can achieve the highest accuracy. Although MNIST is a saturated benchmark, we can observe that our method can find a performant model in much lower training time ($\sim 6\times$ less) compared to FlexConv.

3.4.4 CIFAR-10

Tab. 3.2 represents the results of experiments on the CIFAR10 dataset. We can observe that our method can perform better than N-JetNet [115] and DCN [125] models with different number of parameters. Both of these methods model a convolution kernel as a truncated Taylor series (called N-Jet [150]) of Gaussian derivative filters to learn kernel sizes. N-JetNet-ALLCNN shows significantly worse accuracy compared to our model's variants with 0.66M parameters. N-JetNet-CIFARResNet32 shows higher accuracy, which is on par with our EffConv-20 model with a similar number of parameters, but our model has much lower depth than the former.

It can be seen that our method significantly outperforms CKCNN [135] models in terms of both accuracy and training time with a similar parameter budget. Our conjecture for the cause of weak results of CKCNN is that it is mainly designed for modeling sequential

data and not images.

FlexConv [116] can obtain models with higher accuracy models compared to DCN and CKCNN, but it does so at the cost of higher training time per epoch. Our method shows higher accuracy with a similar number of parameters while having a much lower training time. In the low parameter regime, FlexConv-3 with 0.27M parameters can achieve an average of 90.4% accuracy. At the same time, our EffConv-20 model with 0.20M parameters shows an average accuracy of 91.78%. In the higher parameter setting, FlexNet-16 with 0.67M parameters shows lower average accuracy compared to EffConv-16 with 0.66M parameters while requiring more than 7× (127 *vs.* 17.77) training time per epoch. This result highlights the critical advantage of our model, *i.e.,* finding performant deeper models with much lower computations than FlexConv. The cost of training FlexConv models drastically increases with the increase in the model's depth, which prevents it from training models deeper than 16 layers. Conversely, we show in our ablation studies that our model can readily find proper kernel sizes for much deeper models like ResNet-56. One can also find that directly plugging CKConv and FlexConv layers into the CIFARResNet-44 model's blocks severely degrades its performance while increasing its number of parameters by about 4×. In contrast, while keeping the number of parameters the same as CIFARResNet-44 (0.66M), our EffConv-44 can find a better configuration for its kernel sizes with a slight increase in its training time without requiring special block designs like CKConv and FlexConv. Finally, we can observe that our Wide-EffConv-28-1 can achieve an average accuracy of 93.14, which is between EffConv-32 and EffConv-44 while requiring about half the training time of EffConv-44. This result is congruent with previous works [144] that observed that Wide-ResNet models perform similarly to deeper ResNets while having lower training/inference

time thanks to their smaller depth.

Finally, we visualize the learned sizes and weights of the kernels for our EffConv-20 model with 0.66M parameters in Fig. 3.2. As can be seen, in contrast with FlexConv [116] that puts larger kernels in the deeper layers, our model does not show such a behavior, and the largest kernel is set in the 5^{th} layer. We provide the optimal kernel sizes for other architectures in supplementary S3.2.

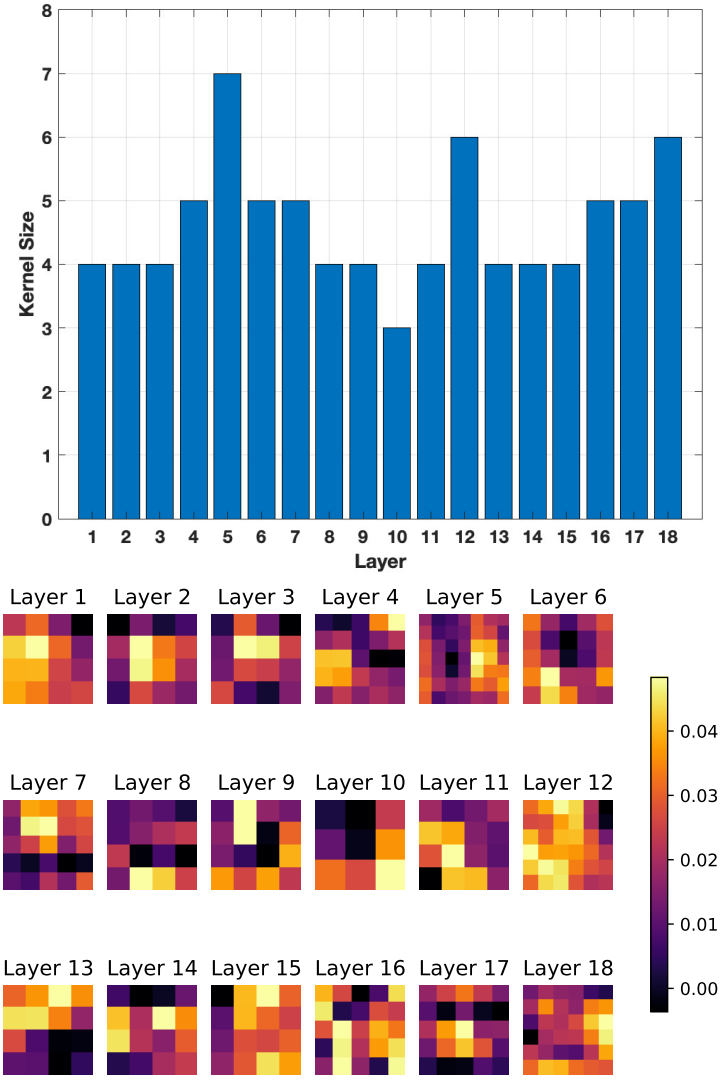


Figure 3.2: Learned sizes (top) and weights (bottom) for the kernels of our EffConv-20 model with 0.66M parameters on CIFAR-10.

[2]The number is provided by the authors by correspondence.

Table 3.3: Results on the STL-10 dataset.

Model	Size	Accuracy	Time (Sec / Epoch)
CIFARResNet-18 [39]	11.7M	81.0	14.2
FlexNet-16 [116]	0.67M	68.6 ± 0.7	∼1800[2]
EffConv-20 (Ours)	0.66M	73.06 ± 0.53	13.2
EffConv-20 (Ours)	0.71M	73.60 ± 0.58	14.41
EffConv-20 (Ours)	0.78M	74.02 ± 0.27	14.45

3.4.5 STL-10

We present the results on the STL-10 dataset in Tab. 3.3. STL-10 [117] is a challenging benchmark as it has a small training dataset (5k samples) and a more extensive test set (8k samples) that can better show the differences between methods compared to CIFAR-10 and MNIST. As can be seen, FlexConv [116] becomes a highly inefficient method on STL-10 even though it has a small training dataset. The reason is that the input images have a size of 96px, *i.e.*, 9× larger than 32px images in CIFAR-10. Thus, FlexConv needs to compute 96^2 forward passes of the MLP for each convolution kernel to compute its weights in each iteration, making it require about 30 minutes to complete each epoch. In contrast, thanks to our adaptive weight predictor model design, our method can successfully find the proper kernel sizes in a significantly lower time showing training time on par with the CIFARResNet-18. In addition, our method can noticeably shrink the gap between low parameter models and the high parameter CIFARResNet-18 model. With a similar 0.66M parameter budget, EffConv-20 can outperform FlexConv with 4.46% accuracy, and at the same time, finding and training such a model in 136× less time. Thus, EffConv is more practical compared to FlexConv for applications with large input sizes.

Table 3.4: Results on the ImageNet-32 dataset.

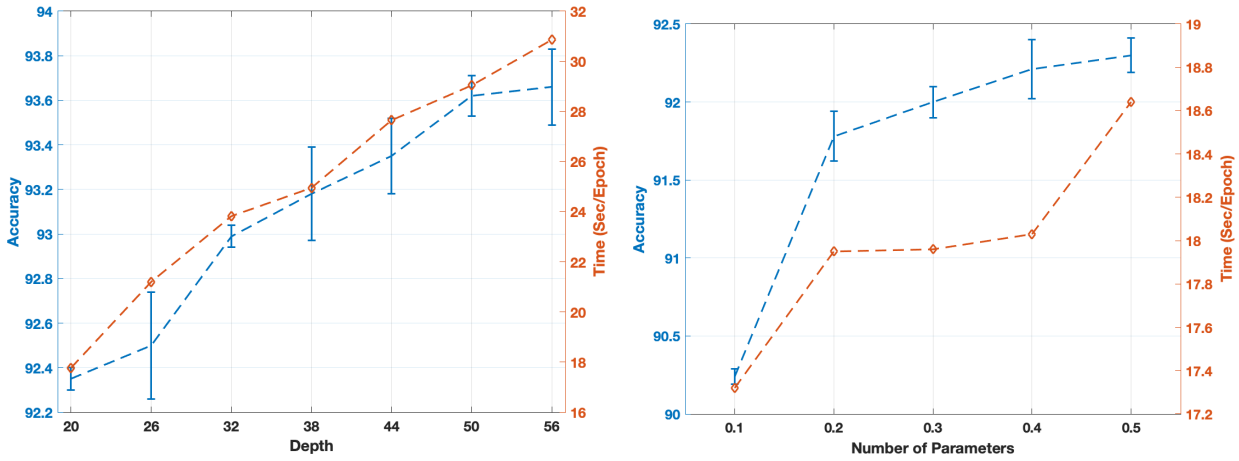
Model	Size	Accuracy		Time
		Top-1	Top-5	(Sec / Epoch)
CIFARResNet-32 [39]	0.53M	26.41 ± 0.13	49.37 ± 0.15	-
WRN-28-1 [144]	0.44M	32.03	57.51	-
FlexNet-5 [116]	0.44M	24.9 ± 0.4	47.7 ± 0.6	~ 740[3]
EffConv-20 (Ours)	0.50M	36.07 ± 0.18	60.96 ± 0.1	215.27

3.4.6 ImageNet-32

We provide the results on the ImageNet-32 dataset in Tab. 3.4. We can find that WRN-28-1 significantly outperforms ResNet-32, similar to the results mentioned in CIFAR-10 experiments above and previously shown [144] in the literature. Moreover, FlexConv cannot achieve a competitive performance compared to baselines. Due to its computational inefficiency bottleneck, training deep models on the ImageNet-32 with 1M samples is very expensive for FlexConv. Thus, it has to resort to training a model with only 5 layers, resulting in a limited capacity and unsatisfactory performance. However, EffConv can train a 4× deeper model (20 layers) in 3.4× less time. With slightly less parameters count, EffConv-20 significantly outperforms CIFARResNet-32 with 9.66%/11.59% average top-1/top-5 accuracy. It also shows about 4%/3.4% higher average top-1/top-5 accuracy compared to WRN-28-1.

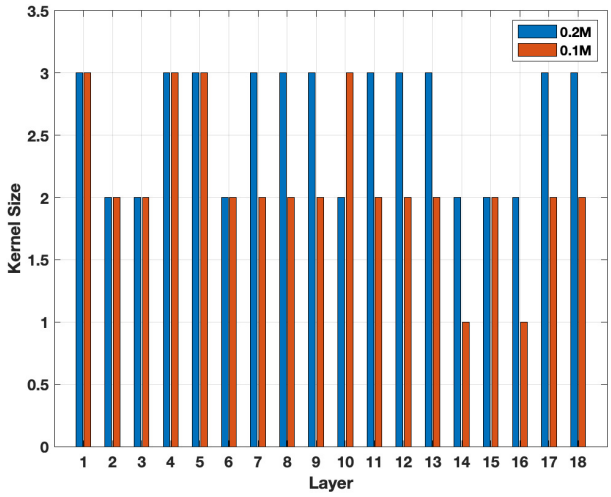
In summary, our experiments on four benchmarks clearly demonstrate that our method, EffConv, can find proper kernel sizes for a classifier more effectively and efficiently than the previous kernel size learning methods. It can provide a practical solution to the problem of determining kernel sizes for a CNN classifier.

[3]The number is provided by the authors by correspondence.



(a)

(b)



(c)

Figure 3.3: Results of our ablation studies.

3.4.7 Ablation Studies

We perform two ablation experiments to explore the behavior of our method.

Fixed Parameter Budget, Varying Depth: In the first experiment, we use EffConv to find proper kernel sizes for ResNet [39] models with depths in $\{20, 26, \cdots, 56\}$ with a fixed parameter budget of 0.66M. Fig. 3.3(a) demonstrates the results. The left vertical axis represents the accuracy of the resulting models, and the right one shows the training time

per epoch. We can observe that, as expected, the accuracy and training time of the learned model increases when we increment its depth. However, the performance almost gets saturated from ResNet-50 to ResNet-56 when these models should decrease their parameters while preserving their depth. Thus, they become thin and deep models, so increasing their depth does not provide noticeable performance returns. We also emphasize that our method can readily find proper kernel sizes for deep models such as ResNet-56 while requiring ~ 31 seconds per epoch. At the same time, FlexConv [116]/CKCNN [135] require about $4\times/3\times$ more training time (see Tab. 3.2) for their 16-layer models ($3.5\times$ fewer layers than ResNet-56) and achieve much worse performance.

Fixed Depth, Varying Parameter Budget: In our second experiment, we use a ResNet-20 model (fixed depth) and change its desired parameter counts when we find its kernel sizes with our model. We set the number of parameters from 0.1M to 0.5M, roughly half and twice the number of parameters of the original ResNet-20. We present the results in Fig. 3.3(b). We can find that the accuracy increases with the number of parameters, and the slope of changes is low after 0.4M parameters. Moreover, with only 0.1M parameters, our model shows higher performance (90.24 ± 0.05) than N-JetNet-ALLCNN/DCN-σ^{ji} (see Tab. 3.2), while these models have much higher parameters. One can also notice a nonlinear change in training time. It increases from 0.1M to 0.2M but then almost remains the same from 0.2M to 0.4M despite doubling the number of parameters. Then, again it increases from 0.4M to 0.5M. In contrast, the training time increases almost linearly with the number of layers (Fig. 3.3(a)) but with a slope of less than one. Finally, Fig. 3.3(c) visualizes the kernel sizes for the EffConv-20 with 0.1M/0.2M parameters. Interestingly, we can find that our model determines the same size configuration for the first layers of two models that are

known to extract low-level general features such as edges, and the models differ in deeper layers.

3.5 Chapter Summary and Conclusion

In this chapter, we introduced a novel kernel size learning method for a classifier that overcomes the computational inefficiencies of the previous methods while outperforming their performance. We developed a size predictor model that learns to predict the optimal kernel sizes for the classifier given a desired budget for its parameters' count. We trained the size predictor using a second model called kernel predictor that adaptively determines the kernel weights of the classifier given the predicted sizes of the size predictor. Both models are guided by the classification and our parameter budget objectives. Our new model can discover the proper configuration for the kernel sizes and train the resulting model with much lower computations compared to the previous methods. Experiments on four prominent benchmarks demonstrate that our method outperforms the baselines in terms of the classifier's accuracy and significantly reduces their training time per epoch given a fixed parameter budget, showing larger margins on more challenging tasks. Ablation studies also revealed that our method is able to find kernel sizes for much deeper architectures that were not feasible for baselines due to their inefficiencies.

Supplementary Materials for Chapter 3

S3.1 Experimental settings

In this section, we provide more details of our experiments.

S3.1.1 Setup

Datasets: We used MNIST [142], CIFAR-10 [5], STL-10 [117], and ImageNet-32 [143].

- *MNIST* contains 28×28 hand-written digits images from 10 classes with 60000 training samples and 10000 test ones.
- *CIFAR-10* contains 60000 natural images with size 32×32 that are split into 50000 training and 10000 testing images.
- *STL-10* is a subset of ImageNet [111] with 10 classes of 96×96 images containing 5000 training samples and 8000 test images.
- *ImageNet-32* is a down-sampled version of the original ImageNet dataset. It contains 1000 classes with 1281167 training images and 50000 test samples.

Architectures: As mentioned, we perform our experiments to learn kernel sizes for ResNet [39] and Wide-ResNet [144] architectures. These models have an input 3×3 convolution layer followed by residual blocks. In all of our experiments, we learn the sizes for the kernels in residual layers and keep the input convolution layer intact.

S3.1.2 Implementation

Size Predictor: We use a Gated Recurrent Unit (GRU) [136] and dense layers to implement our size predictor. The detailed architecture is shown in Tab. 3.5. The input z is randomly generated and fixed after initialization.

Table 3.5: The architecture of our size predictor.

Input z
GRU(128,256), WeightNorm, ReLU Dense$_l$(256, 1), WeightNorm, $l = 1, \cdots, L$
Outputs v_l, $l = 1, \cdots, L$;

Kernel Predictor: We implement the kernel predictor for each convolution layer with a GRU unit, and we denote it with g^l. Similar to the size predictor, the kernel predictor models have a constant input. Their output size is determined by the input and output channels of their corresponding convolution layer, as described in Eq. 3.5. Given a determined size s_l by the size predictor, g^l performs $s_l \times s_l$ forward passes to compute the predicted weights $\hat{w}_l$.

S3.2 Results

We provide the optimal kernel sizes found by our model for different architectures and datasets in the following. The caption of each figure describes the dataset name, architecture, and its number of parameters.

S3.2.1 MNIST

The kernel sizes for our EffConv-20 model with 0.66M parameters on MNIST are shown in Fig. 3.4.

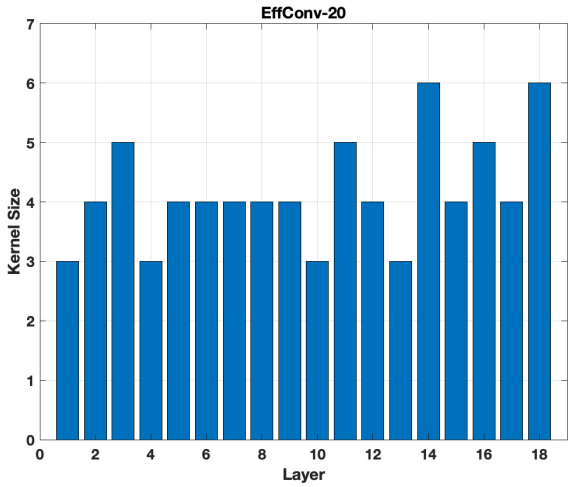


Figure 3.4: MNIST, EffConv-20, 0.66M.

S3.2.2 CIFAR-10

Figs. 3.5-3.11 represent the learned kernel sizes for the EffConvs with depths in $\{26, \cdots, 56\}$ as well as the Wide-EffConv-28-1. We presented the ones for EffConv-20 in Fig. 3.2.

S3.2.3 STL-10

Figs. 3.12-3.14 provide the learned kernel sizes for EffConv-20 with different number of parameters on the STL-10 dataset.

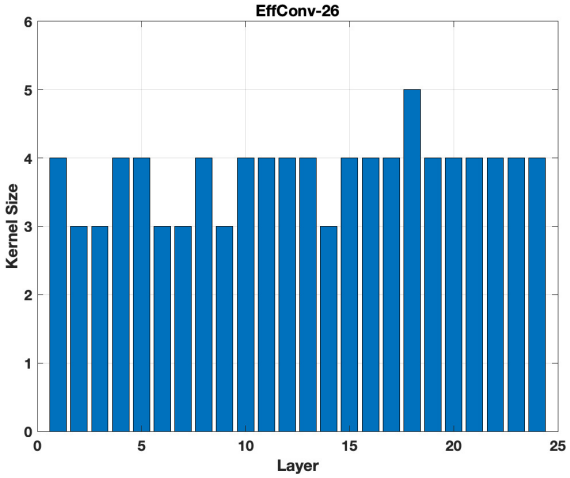


Figure 3.5: CIFAR-10, EffConv-26, 0.66M.

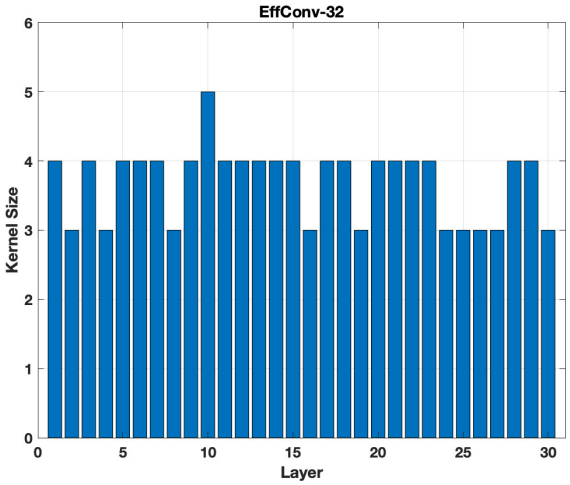


Figure 3.6: CIFAR-10, EffConv-32, 0.66M.

S3.2.4 ImageNet-32

Fig 3.15 shows the learned kernel sizes of EffConv-20 for the ImageNet-32 benchmark.

S3.2.5 Ablation Study

We provide the learned kernel sizes for EffConv-20 with the number of parameters in $\{0.3\text{M}, 0.4\text{M}, 0.5\text{M}\}$ in Figs. 3.16-3.18.

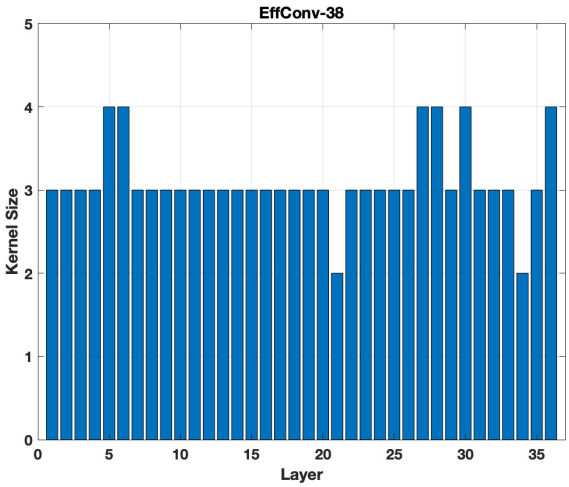


Figure 3.7: CIFAR-10, EffConv-38, 0.66M.

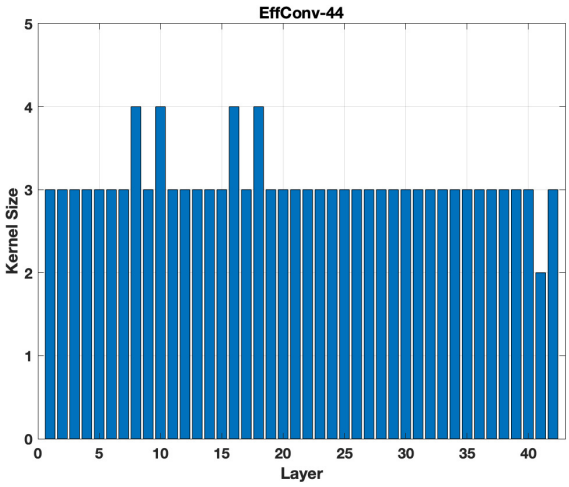


Figure 3.8: CIFAR-10, EffConv-44, 0.66M.

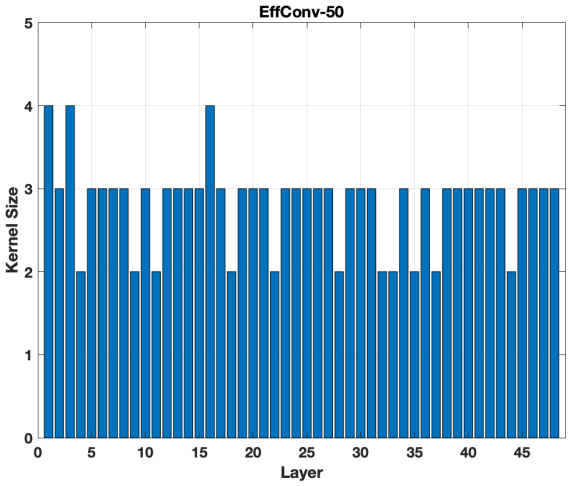


Figure 3.9: CIFAR-10, EffConv-50, 0.66M.

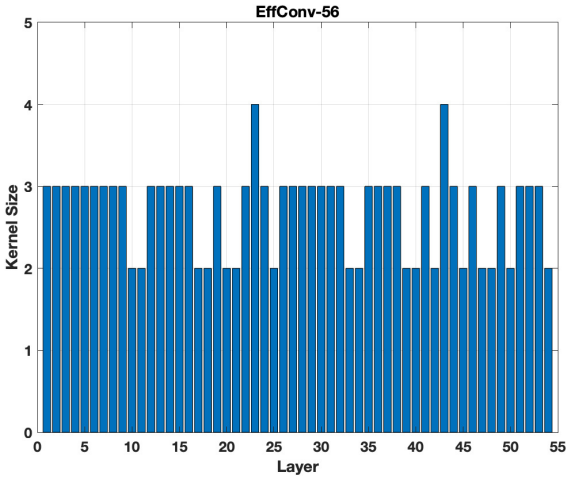


Figure 3.10: CIFAR-10, EffConv-56, 0.66M.

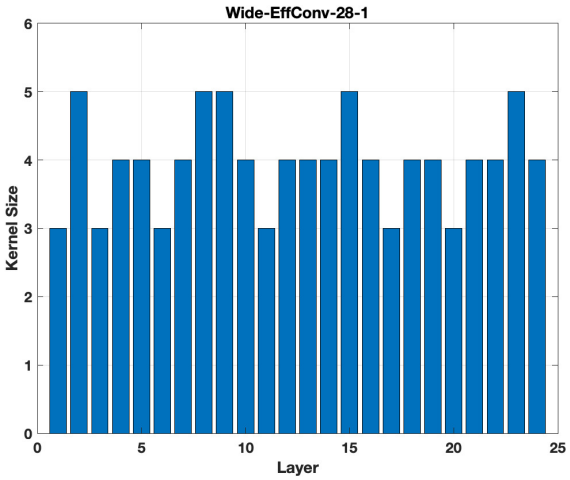


Figure 3.11: CIFAR-10, Wide-EffConv-28-1, 0.66M.

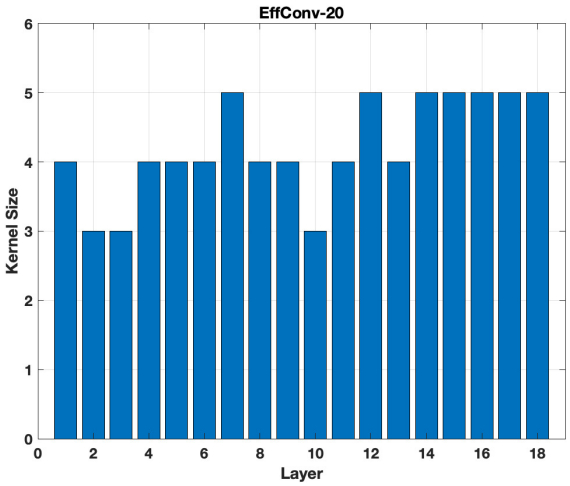


Figure 3.12: STL-10, EffConv-20, 0.66M.

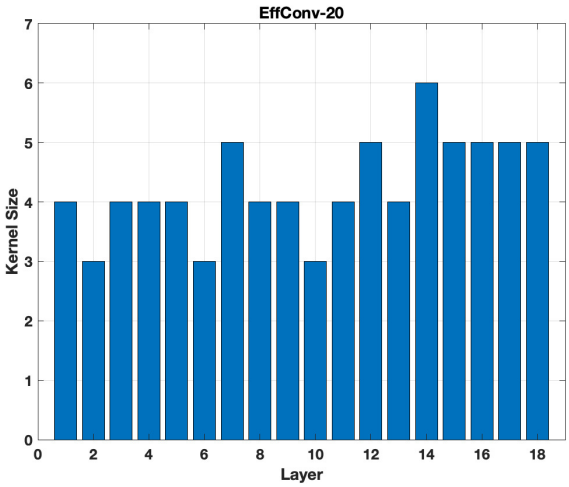


Figure 3.13: STL-10, EffConv-20, 0.71M.

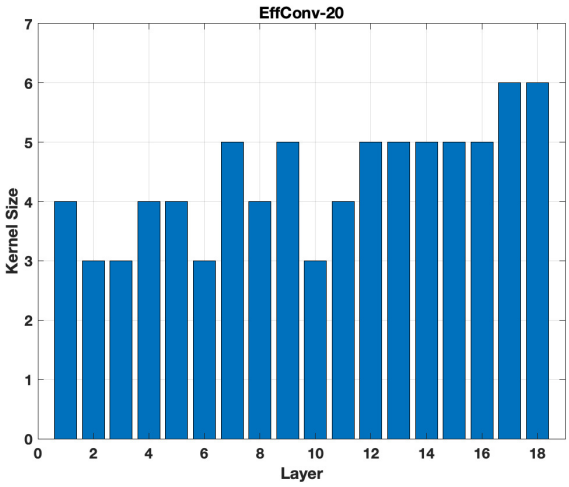


Figure 3.14: STL-10, EffConv-20, 0.78M.

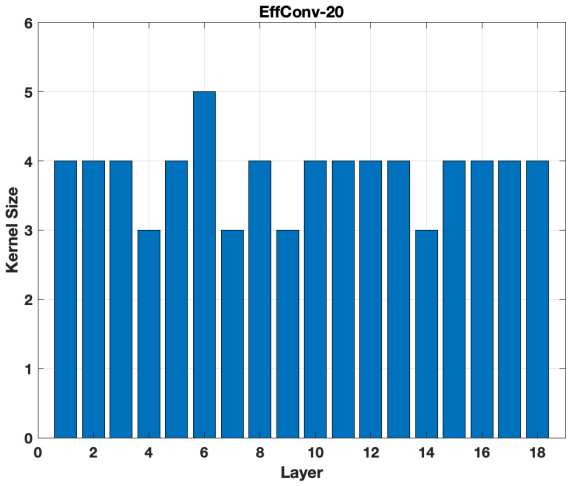


Figure 3.15: ImageNet-32, EffConv-20, 0.50M.

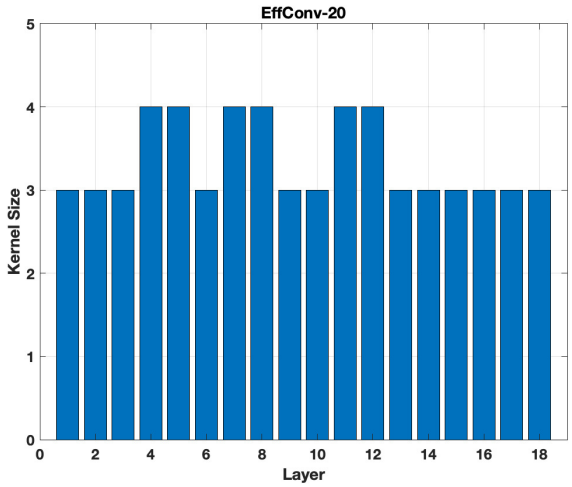


Figure 3.16: CIFAR-10, EffConv-20, 0.3M.

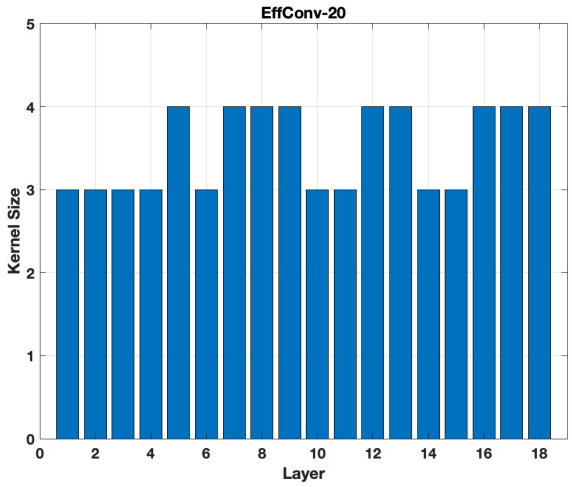


Figure 3.17: CIFAR-10, EffConv-20, 0.4M.

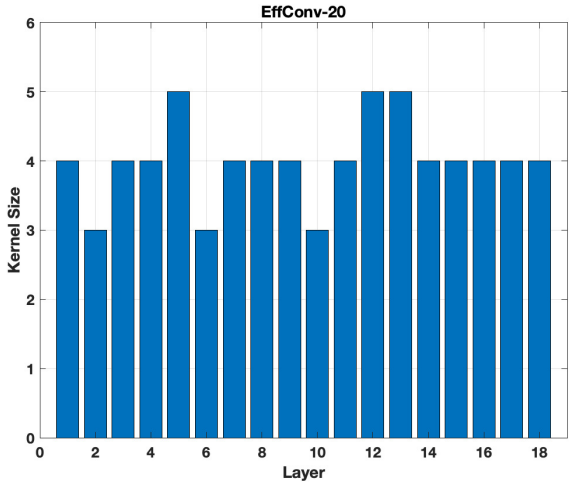


Figure 3.18: CIFAR-10, EffConv-20, 0.5M.

Chapter 4: Jointly Training and Pruning CNNs via Learnable Agent Guidance and Alignment

4.1 Introduction

Convolutional Neural Networks (CNNs) have enabled unprecedented achievements in the last decade in different domains [36, 39, 151, 152]. They have shown a trend for better performance when benefiting from deeper and wider architectures, larger dataset sizes, and longer training times with modern hardware [107, 110, 153, 154]. Despite their accomplishments, the tremendous memory and computational requirements of CNNs prohibit deploying them on edge devices with limited battery and compute resources, making CNN compression a crucial step before their deployment. The goal is to reduce the size and computational burden of CNNs while preserving their performance. Model pruning (removing weights [41] or structures [44] like channels and layers), weight quantization [43], knowledge distillation [45], Neural Architecture Search (NAS) [121, 155], and lightweight architecture designs [98, 156] are common categories of ideas for CNN compression.

Structural pruning which removes redundant channels of a CNN is our main focus in this chapter. It is more practically plausible than weight pruning as it can effectively reduce the inference cost of a model on established hardware like GPUs without requiring

special libraries [26] or post-processing steps. Further, it demands far less design efforts than NAS [122] and architecture design methods [120, 157]. The proposed structural pruning methods determine the importance of each channel using metrics such as resource loss [49], norm [44], and accuracy [50] and prune a model with techniques like greedy search [47] as well as evolutionary algorithms [48]. Thanks to the advances of Reinforcement Learning (RL) methods in complex decision making tasks [158, 159, 160], leveraging RL methods to determine proper sub-networks of a CNN given a desired budget has been explored in recent years [46, 161, 162, 163, 164, 165]. AMC [46] trains a DDPG agent [166] to prune layers of a pretrained CNN. LFP [164] trains an agent to get weights of a CNN's filters and determine keeping or pruning them. N2N [161] uses two agents to perform layer removal and layer shrinkage respectively. Finally, GNNRL [165] utilizes graph neural networks to identify CNN topologies and employ RL to find a proper compression policy. Despite their promising results, these models require a pretrained CNN model for training their RL agent for pruning as the prominent RL algorithms, like DDPG [166] used in AMC [46], cannot perform well in dynamic environments [167] if one trains model's weights along with the agent's policy.

We propose a novel model pruning method to jointly learn a CNN's weights and structurally prune its architecture using an RL agent. As training the weights and pruning them cannot happen simultaneously, we apply a soft regularization term to the model's weights during training to align with the sub-structure chosen by the best agent's policy. We iteratively train the model's weights for one epoch and perform several RL trajectories observations on the most recent model to update the policy of the RL agent. We design our RL agent so that in each of its episodic trajectories, its actions determine the pruning

ratio for each layer of the model. After pruning all layers, we take the resulting model's accuracy as our agent's reward. However, as the model's weights get updated in each epoch, the reward function of the RL agent changes dynamically between epochs. Accordingly, the training episodes of our agent are drawn from a non-stationary distribution. Therefore, one cannot simply employ prominent RL methods like DDPG [166] and Soft Actor-Critic (SAC) [168] in our framework to prune the model because their core assumption which is the environment being stationary [167] is not fulfilled. To overcome this challenge, we design a mechanism to model the evolving dynamics of the agent's environment. We take an embedding for each epoch of the training and employ a recurrent model to determine a representation of the current state of the environment for the agent given the embeddings of the epochs so far. We train the recurrent model along with a decoder model in an unsupervised fashion to reconstruct the reward values observed in the agent's trajectories. Finally, we augment each episodic trajectory of the agent with the representations of the state of the environment (provided by the recurrent model) at the time of the trajectory. By doing so, our RL agent has access to all information regarding the dynamic environment, and we employ SAC [168] to train the agent using the augmented trajectories. In addition, our soft regularization scheme for alignment of weights and the selected structure by the agent enables our pruned model to readily recover its performance in fine-tuning. We summarize our contributions as follows:

- We propose a novel channel pruning method that jointly learns the weights and prunes the architecture of a CNN model using an RL agent. In contrast with previous methods using RL for pruning, our method does not need a pretrained model before

pruning.

- We perform joint training and pruning by iteratively training the model's weights and the agent's policy. We utilize a soft regularization technique to the model's weights during training, encouraging them to align with the structure determined by the agent. By doing so, our method can identify a high-performing base model with weights that closely match the structure selected by the agent. Consequently, the pruned model can readily recover its high performance in fine-tuning.

- We design a mechanism to model the dynamics of our evolving pruning environment. To do so, we use a recurrent model that provides a representation of the state of the environment to the agent. We augment the trajectories observed by the agent using the provided representations to train the agent.

The contents of this chapter are based on our work [169] published in CVPR 2024.

4.2 Related Work

Model Pruning: Model compression ideas [170, 171] can be categorized as structural pruning [44, 73, 75, 76, 77, 172, 173, 174, 175, 176, 177], weight quantization [42, 43, 178], weight pruning [41, 179, 180], Neural Architecture Search (NAS) [119, 155, 181], knowledge distillation [182], and lightweight architecture design [120, 122]. Structural pruning focusing on removing redundant channels (filters) of a CNN is the direction most related to our work. The proposed methods have approached this problem from various directions like pruning filters with smaller norms [44], applying group regularization [183]

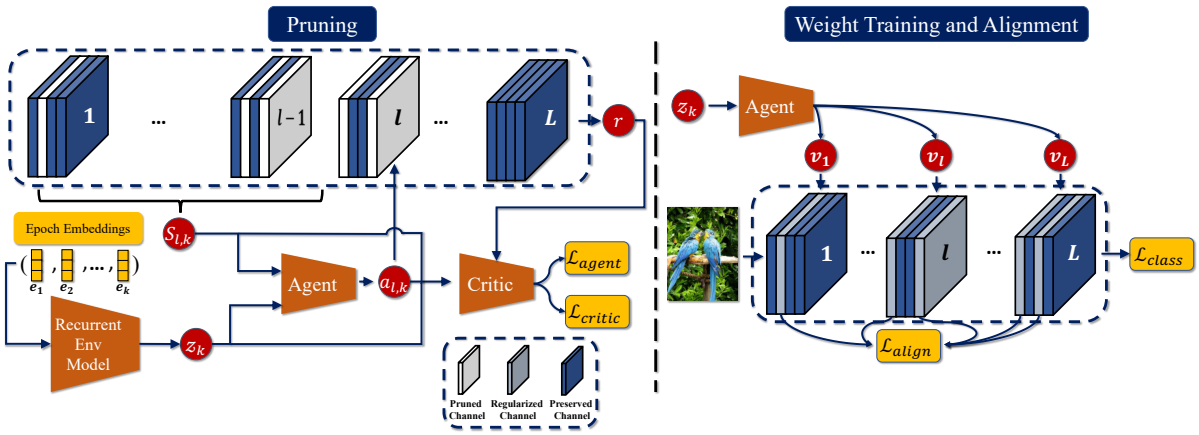


Figure 4.1: **Overview of our method**. We jointly train and prune a CNN model using an RL agent by iteratively training the agent's policy and model's weights. In each iteration, we train the model's weights for one epoch and perform several episodic observations of the agent. **Left:** Each action of our agent prunes one layer of the model, and the procedure of pruning the l-th layer is shown. The agent's actions on the previous layers and the remaining layers' FLOPs determine its state, and we take the resulting model's accuracy as its reward (Sec. 4.3.2). As the model's weights change between iterations, the reward function also changes accordingly. Thus, we map each epoch to an embedding and employ a recurrent model to provide a state of the environment z to the agent. (Sec. 4.3.2.1) **Right:** Given a sub-network selected by the agent, we train the model's weights while softly regularizing them to align with the selected structure (Sec. 4.3.2.3).

like LASSO [184] during training, ranking filters' importance using low-rank decomposition [97], estimating influence of a filter on loss [185] using the Taylor decomposition, and meta-learning [50]. Recently, several ideas have employed Reinforcement Learning (RL) for pruning [46, 161, 162, 163, 164, 165]. LSEDN [163] trains an RL agent to perform layer-wise pruning on DenseNet [40]. N2N [161] proposed a two-stage process that employs two recurrent RL agents in which one agent removes layers of a pretrained CNN, and then, the other agent shrinks each remaining layer. LFP [164] introduces 'try-and-learn' scheme in which RL agents learn to take layers' weights and predict binary masks for pruning or preserving layers' filters. AMC [46] takes a pretrained CNN and trains a DDPG [166] to prune its convolution layers. Yet, these models can only prune pretrained models and cannot jointly train and prune the models. The main reason is that they employ off-the-shelf RL meth-

ods [166, 168] that assume stationary training environments, but if the model's weights change during pruning, the environment will not be stationary. We develop a novel channel pruning method that jointly learns a CNN's weights and prunes its architecture using an RL agent in an iterative manner. We design a procedure to model the changing dynamics of the reward function of our RL agent using a recurrent model that provides a representation of the current state of the environment. In addition, we regularize the model's weights to align with the structure determined by the agent. Thanks to such designs, our method can train a base model and select a perfromant sub-network of it that can easily recover its performance in fine-tuning.

Reinforcement Learning: RL methods [186, 187] have achieved outstanding results in complex tasks [158, 159, 160] using techniques like Q-learning [188] and policy optimization [189]. Yet, it has been shown that they cannot generalize to new variations of their primary task [167, 190]. Continual RL methods [167] are related to our work as our agent's environment is non-stationary. The proposed ideas address different non-stationarity conditions. For example, multi-task learning [191] and meta-learning methods [192, 193, 194] assume that sequential tasks presented to the agent have an unknown stationary distribution. Curriculum learning ideas [195, 196, 197] aim to learn the agent's own curriculum in single or multi-agent dynamic environments. Finally, a group of methods [198, 199, 200] make assumptions on the variation budget of the reward function to improve learning of the agent. We refer to [167] for a comprehensive review on continual RL methods. Different from the mentioned ideas, we employ a recurrent model to model the changing dynamics of the environment of our agent and augment its observations with it to enable using episodic RL methods to train our agent.

4.3 Method

We propose a new channel pruning method for CNNs to jointly learn the weights of a CNN model and prune its filters using an RL agent. To do so, we iteratively train the model's weights and the agent's policy. We design the agent such that its actions determine the compression rate of the model's layers, and we take the accuracy of the pruned model as the reward function for the agent. Nevertheless, as we update the model's weights iteratively, the reward associated with a certain action changes in consecutive iterations, which results in a non-stationary environment for the agent and prevents using episodic RL methods for our purpose as they assume the environment is stationary. We develop a mechanism in which a recurrent network models the changing dynamics of the reward function and provides a representation of the changing state of the environment to the agent to alleviate this challenge. Finally, as we cannot both train and prune the weights simultaneously during training, we propose a regularization to align the model's weights with the selected sub-network by the agent. Fig. 4.1 shows the overall scheme of our method.

4.3.1 Notations

We denote the number of layers in a CNN with L and the weights of its l-th convolution layer with $\mathcal{W}_l \in \mathcal{R}^{C_{l+1} \times C_l \times W_l \times H_l}$. C_{l+1}, C_l represent the number of output and input channels of the layer, and W_l as well as H_l are the spatial dimensions of its kernel. We show the stride of the l-th layer with $stride_l$, and FLOPs[l] is its FLOPs value. Finally, we show the floor function with $\lfloor \cdot \rfloor$

4.3.2 Iterative Weight Training and Compression

We iteratively train a CNN's weights and optimize the policy of an RL agent to jointly train and prune it. Specifically, in each iteration, we first train the model's weights for one epoch on our training dataset while the agent is fixed. Then, we keep the model's weights frozen and our agent observes several episodic trajectories by performing its actions on the CNN's layers. To employ an RL agent to prune our model, we need to define three main components for the agent: States that it visits in the environment, the actions that it can perform, and the reward function given states and actions. We describe our choices for each one in the following, and we denote the desired FLOPs budget for the pruned model with FLOPs_{desire}.

States of the Agent: We design our RL agent to perform actions that determine the pruning rate for consecutive layers of the CNN model. Thus, the agent's states depend on the index of the layer that the agent is currently pruning; the layer's characteristics like its kernel size and FLOPs; and the number of FLOPs that the agent has already pruned as well as the amount it has to prune from the remaining layers. Formally, given that the agent is currently pruning the l-th layer, we define the state of the environment as follows:

$$\begin{aligned} S_l = &[l, C_l, C_{l+1}, stride_l, k_l, \\ &\text{FLOPs}[l], \text{FLOPs}_{1:l-1}, \text{FLOPs}_{l+1:L}, a_{l-1}] \end{aligned} \tag{4.1}$$

where k_l is the layer's kernel size. $\text{FLOPs}_{1:l-1}$ denotes the number of previous layers' FLOPs given the actions that the model has done so far on them. $\text{FLOPs}_{l+1:L}$ shows the next layers' FLOPs that are not pruned yet, and a_{l-1} is the agent's action on the previous layer.

Actions of the Agent: Based on the state S_l for the l-th layer, our agent determines its pruning rate a_l such that $a_l \in [0, 1)$. Given the predicted pruning rate a_l, we remove $\lfloor a_l \times c_l \rfloor$ channels of the layer. In addition, we calculate the minimum and maximum actual feasible pruning rates for the current layer based on $\text{FLOPs}_{1:l-1}$, $\text{FLOPs}[l]$, $\text{FLOPs}_{l+1:L}$, and the desired budget FLOPs_{desire}. Then, we bound the predicted action a_l to lie in the range $[a_{l,min}, a_{l,max}]$. We refer to supplementary S4.1 for more details. In our experiments, we found that ranking the importance of filters using the norm criteria [44] and pruning the ones with the lowest rank works well in our framework, but one may employ more sophisticated approaches [97] as well.

Reward Function: We set our reward function to be the pruned model's accuracy on a small held-out subset of the training dataset as a proxy for its final performance. As our agent prunes one layer of the model at a time, it will be extremely time-consuming to calculate the proxy value after each action of the agent. Thus, we take one pass of the agent on all layers of the model as one episodic trajectory for it. Then, we calculate the final pruned model's accuracy on the subset at the end of the trajectory and take it as the reward value for all state-action pairs seen during the trajectory.

4.3.2.1 Modeling the Dynamic Nature of Rewards

As we iteratively train the model's weights and the agent's policy, the weights of the convolution layers that the agent performs its actions on them are not static in our framework. Thus, our reward function is dynamic in the course of training, which prohibits directly applying prominent RL methods [166, 168] that leverage episodic trajectories to

train the agent's policy in our framework. The reason is that the optimization procedure of these models is biased to only optimize the agent's policy $w.r.t$ the current episode's distribution and disregards the changing dynamics of the environment, resulting in a sub-optimal policy [167].

We design a new mechanism to overcome this challenge by providing a representation of the dynamic environment to the agent. To do so, first, we map the index of each epoch for training the model's weights to an embedding. Then, we employ a recurrent GRU model [201] that takes a sequence of the embeddings corresponding to the epochs that have been passed so far and outputs a representation of the current state of the model's weights. Formally, if we train the model's weights for total T epochs, we denote the epoch embeddings corresponding to epoch indexes $E = [e_1, e_2, \cdots, e_T]$ with $\Psi_{1:T} = [\psi_1, \psi_2, \cdots, \psi_T]$ ($\psi_1 = \text{Emb}(e_1)$, Emb is a learnable embedding layer). We calculate the representation of the state of the model's weights at the epoch e_k as follows:

$$z_k = f_{Env}(\Psi_{1:k}, h_0; \theta_{Env}) \tag{4.2}$$

f_{Env} denotes the recurrent model. $\Psi_{1:k}$ are the embeddings of epochs until the epoch e_k. h_0 is the initial hidden state of the GRU that we set it to a zero vector, and θ_{Env} are the parameters of the GRU. We show in section 4.3.2.2 that we use the representations z provided by the GRU for training our RL agent.

We propose to train the recurrent model using another model that we call it 'decoder.' The decoder model takes 1) the state-action pairs (S, a) and 2) the representation z of the state of the environment when the agent observes (S, a) and predicts the agent's reward

r. Our intuition is that the representation z is informative of the state of the environment when the decoder can use it to accurately predict r. We train both the recurrent model's weights and the ones for the decoder using the following objective:

$$\min_{\theta_{Env},\theta_D} \mathcal{L}_{recons} = \mathbb{E}_{(s,a,r,e)\sim B}[(\hat{r} - r)^2] \tag{4.3}$$

$$\hat{r} = f_D(S, a, z; \theta_D) \tag{4.4}$$

$$z = f_{Env}(\Psi, h_0; \theta_{Env}) \tag{4.5}$$

In practice, we approximate the expectation in Eq. 4.3 using the agent's episodic observations during training. f_D is our decoder model, and θ_D represents its parameters.

4.3.2.2 RL Agent Training

We employ our recurrent model and the Soft Actor-Critic (SAC) [168] method to train our RL agent. We augment the states S with the representations of the environment's state z and design our agent's policy function so that it predicts a distribution over actions conditioned on both of them:

$$a \sim \pi(\cdot|S, z\ ; \theta_A) \tag{4.6}$$

Similarly, we deploy the representations z when calculating the predicted Q-values by the critic networks in SAC. We train them using the mean squared Bellman error objective:

$$\mathcal{L}(\phi_i) = \mathbb{E}_{(s,a,r,s',d,e)\sim B}[(Q_{\phi_i}(s, a, z) - y(r, s', d, z))^2] \tag{4.7}$$

$$
\begin{aligned}
&y(r, s', d, z) = r + \gamma(1-d)[\min_{j=1,2} Q_{\phi_{targ,j}}(s', a', z) - \\
&\alpha \log(\pi(a'|s', z; \theta_A))]; \\
&a' \sim \pi(\cdot|s', z; \theta_A);\ z = f_{Env}(\Psi, h_0)
\end{aligned}
\tag{4.8}
$$

Q_{ϕ_i} represents the critic models, and $Q_{\phi_{targ,i}}$ shows their target models obtained using Polyak averaging. B is a replay buffer containing previous episodic trajectories observed. (s, a) are state-action pairs from B. d indicates whether the state s is a terminal state. s' represents the state that the model gets in after taking the action a when being in the state s. a' is an action chosen using the most recent policy $\pi(\cdot; \theta_A)$ conditioned on the state s' and environment representation z. γ is the discount factor for future rewards. α determines the strength of the entropy regularization term, which is a hyperparameter. Finally, we train the agent's policy using the following objective:

$$
\begin{aligned}
&\max_{\theta_A} \mathbb{E}_{(s,e)\sim B}[\min_{j=1,2} Q_{\phi_j}(s, a) - \alpha \log \pi(a|s, z; \theta_A)] \\
&a \sim \pi(a|s, z; \theta_A)
\end{aligned}
\tag{4.9}
$$

4.3.2.3 Soft Regularization of the Model's Weights

As mentioned in the section 4.3.2, we iteratively train and prune the model's weights in our framework. One approach to do so can be actually pruning the model's architecture by removing the redundant channels selected by the agent and only training the remaining ones in the weight training phase. However, doing so can make the training procedure unstable because it can significantly drop the model's accuracy. Accordingly, we propose

Algorithm 2 Joint Training and Pruning

Input: Training dataset $\mathcal{D} = \{(x_i, y_i)\}$; replay buffer B; CNN model $f_c(\cdot;\ W)$ with L layers; Agent $\pi(\cdot;\theta_A)$; Two Critics $Q_{\phi_i}(\cdot)$ and their target models $Q_{\phi_{targ,i}}$ $(i \in \{1, 2\})$; recurrent model f_{Env} and epoch embeddings Ψ; regularization parameters α, β; discount factor γ; number of iterations T; number of pruning episodes per iteration P; a subset $\mathcal{D}_s$ of $\mathcal{D}$ for calculating the agent's reward.

for $t := 1$ *to* T **do**

`/* Representation of Environment */`

1. Calculate z_t using f_{Env} and embeddings Ψ in Eq. (4.2).

`/* RL Agent Exploration and Training */`

for $p := 1$ *to* P **do**

2. Prune the L layers of the CNN f_c one at a time by calculating states $S_{l,p,t}$ and actions $a_{l,p,t}$ using z_t and Eqs. (4.1,4.6).
3. Calculate the reward $r_{p,k}$ using the final pruned model and $\mathcal{D}_s$.
4. Add the experiences $(S_{l,p,t}, a_{l,p,t}, S_{l+1,p,t}, e_t, r_{p,k})$ to the replay buffer B.

end

5. Sample a batch of previous experiences $\mathcal{B}$ from B and use them to calculate the loss value for the recurrent model, decoder, and epoch embeddings using Eq. (4.3). Then, update their parameters using the Adam optimizer.
6. Use the samples in $\mathcal{B}$ to calculate the loss values of the critics $Q_{\phi_i}(\cdot)$ and the agent $\pi(\cdot;\theta_A)$ using Eqs. (4.7, 4.9).
7. Backpropagate the gradients of the calculated losses and update the parameters of the two critic models and the agent using the Adam optimizer.

`/* Training the CNN's Weights */`

8. Use the policy $\pi(\cdot)$ with the highest reward so far to determine the binary architecture vectors $[\mathbf{v}_1, \mathbf{v}_2, \cdots, \mathbf{v}_L]$.
9. Calculate the loss $\mathcal{L}_w$ for the model's weights using Eqs. (4.10, 4.11). Backpropagate its gradients and update the model's parameters using SGD.

end

Return: Trained CNN model and agent.

an alternative approach to softly regularize the model's weights to align with the selected sub-network by the agent. Given binary architecture vectors $[\mathbf{v}_1, \mathbf{v}_2, \cdots, \mathbf{v}_L]$ denoting the channels selected by the current **best** agent for each layer, we use the following regularization term to train the model's weights:

$$\mathcal{L}_{align} = \sum_{l=1}^{L} ||(1 - \mathbf{v}_l) \odot \mathcal{W}_l||_2 \tag{4.10}$$

Here, $\odot$ means element-wise product and the proposed objective applies the Group Lasso regularization on the channels removed by the agent. Finally we combine the standard Cross Entropy Loss ($\mathcal{L}_{class}$) with proposed $\mathcal{L}_{align}$ to train the model's weights:

$$\mathcal{L}_w = \mathcal{L}_{class} + \beta \mathcal{L}_{align} \tag{4.11}$$

In practice, we apply $\mathcal{L}_{align}$ using the version of the policy with the highest reward until the current training iteration to make the training more stable. We summarize our training algorithm for training the model's weights and optimizing the agent's policy in Alg. 2.

4.4 Experiments

We conduct experiments on ImageNet [31] and CIFAR-10 [5] to analyze the performance of our method. For all experiments, we use fully connected models with two hidden layers of size 300 for the architecture of the actor, two critics, and two target models of the critic models. We train the agent and critic models using the Adam optimizer [102] with learning rate of 1e−4 and 1e−3 respectively. We use exponential de-

Table 4.1: Comparison results on CIFAR-10 for pruning ResNet-56 and MobileNet-V2.

Model	Method	Baseline Acc	Pruned Acc	Δ-Acc	Pruned FLOPs
ResNet-56	DCP-Adapt [76]	93.80%	93.81%	+0.01%	47.0%
	SCP [94]	93.69%	93.23%	−0.46%	51.5%
	FPGM [74]	93.59%	92.93%	−0.66%	52.6%
	SFP [95]	93.59%	92.26%	−1.33%	52.6%
	AMC [46]	92.80%	91.90%	−0.90%	50.0%
	FPC [96]	93.59%	93.24%	−0.25%	52.9%
	HRank [97]	93.26%	92.17%	−0.09%	50.0%
	DTP [202]	93.36%	93.46%	+0.10%	50.0%
	RLAL (ours)	93.41%	**93.86%**	**+ 0.45%**	50.0%
MobileNet-V2	Uniform [76]	94.47%	94.17%	−0.30%	26.0%
	SCOP [203]	94.48%	94.24%	−0.24%	26.0%
	MDP [204]	95.02%	95.14%	+0.12%	28.7%
	DCP [76]	94.47%	94.69%	+0.22%	40.3%
	DDNP [174]	94.58%	94.81%	+ 0.23%	43.0%
	RLAL (ours)	94.48%	**94.85%**	**+ 0.37%**	**49.4%**

cay rates of $(\beta_1, \beta_2) = (0.9, 0.999)$ for all of them. For our recurrent model, we use a GRU [201] model with the input size of 128. We also take embeddings of size 128 for all epochs (Sec. 4.3.2.1). We employ a fully connected model we two hidden layers of size 300 as our decoder model. We train the GRU model, epoch embeddings, and the decoder model using the Adam optimizer with learning rate of 1e−3 and decay parameters of $(\beta_1, \beta_2) = (0.9, 0.999)$. We set the entropy regularization coefficient α to 0.1 and β for soft regularization to 1e−4 for all models. Finally, we choose the number of episodic observations per epoch for our agent to be $P = 10$ (see Alg. 2). In all experiments, as we jointly train and prune our model by using **R**einforcement **L**earning and softly **AL**igning the weights of the model with the selected sub-networks, we call our method **RLAL**. We refer to supplementary S4.2 for more details of our experiments.

Table 4.2: Comparison results on ImageNet for pruning ResNet-18/34 and MobileNet-V2.

Model	Method	Baseline Top-1 Acc	Baseline Top-5 Acc	Δ-Acc Top-1	Δ-Acc Top-5	Pruned FLOPs
ResNet-18	MIL [205]	70.28%	89.63%	−3.18%	−1.85%	41.8%
	SFP [95]	70.28%	89.63%	−3.18%	−1.85%	41.8%
	FPGM [74]	70.28%	89.63%	−1.87%	−1.15%	41.8%
	PFP [206]	69.74%	89.07%	−2.36%	−1.16%	29.3%
	SCOP [203]	69.76%	89.08%	−1.14%	−0.93%	45.0%
	GNNRL [165]	69.76%	-	−1.10%	-	51.0%
	GP [207]	70.28%	89.63%	−1.40%	−0.97%	43.9%
	PGMPF [208]	70.23%	89.51%	−3.56%	−2.15%	53.5%
	FTWT [209]	69.76%	-	−2.27%	-	51.5%
	EEMC [80]	70.28%	89.63%	−2.01%	−1.19%	46.6%
	RLAL (Ours)	69.80%	89.10%	**−0.80%**	**−0.42%**	**50.0%**
ResNet-34	SFP [95]	73.92%	91.62%	−2.09%	−1.29%	56.0%
	FPGM [74]	73.92%	91.62%	−1.29%	−0.54%	41.1%
	Taylor [75]	73.31%	-	−0.48%	-	24.2%
	SCOP [203]	73.31%	91.42%	−0.69%	−0.44%	44.8%
	GP [207]	73.92%	91.62%	−1.14%	−0.69%	51.1%
	DMC [49]	73.30%	91.42%	−0.73%	−0.31%	43.4%
	PGMPF [208]	73.27%	91.43%	−1.68%	−0.98%	52.7%
	FTWT [209]	73.30%	-	−1.59%	-	52.2%
	GFS [47]	73.31%	-	−0.40%	-	43.8%
	ISP [64]	73.31%	91.42%	−0.45%	−0.40%	44.0%
	RLAL (Ours)	73.45%	91.48%	**− 0.14%**	**− 0.23%**	**50.0%**
MobileNet-V2	Uniform [98]	71.80%	91.00%	−2.00%	−1.40%	30.0%
	AMC [46]	71.80%	-	−1.00%	-	30.0%
	Random [210]	71.88%	-	−0.98%	-	28.9%
	CC [99]	71.88%	-	−0.97%	-	28.3%
	MetaPruning [50]	72.00%	-	−0.80%	-	**30.7%**
	RLAL (ours)	71.82%	90.26%	**− 0.50%**	**− 0.33%**	29.4%

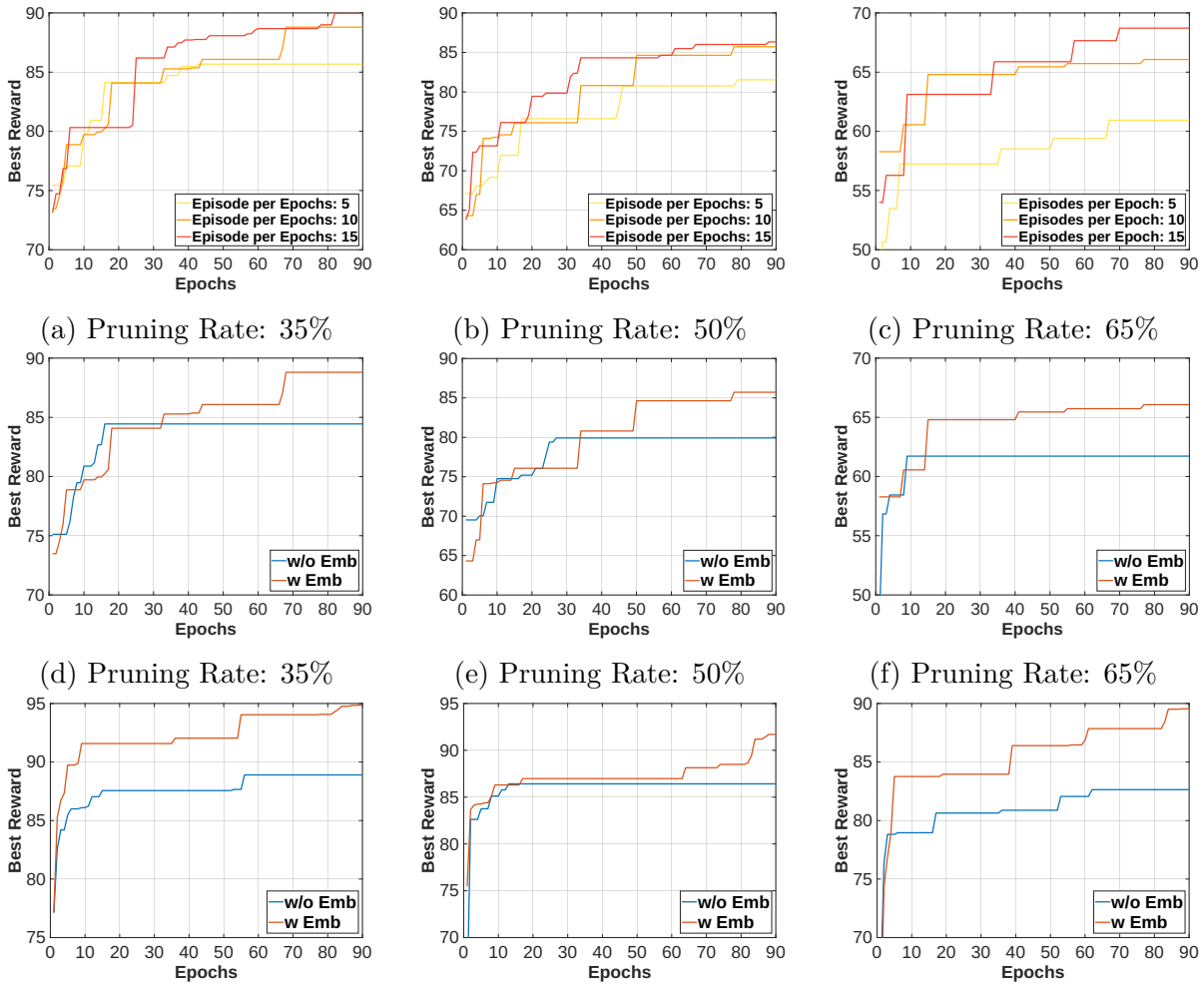
4.4.1 CIFAR-10 Results

Tab. 4.1 summarizes comparison results on the CIFAR-10 dataset. As can be seen, for ResNet-56, RLAL can achieve the best accuracy *vs.* computational efficiency trade-off compared to the baseline methods. One the one hand, it is able to prune FLOPs with a rate comparable to ($< 3\%$ lower) FPC [96] while achieving +0.70 higher Δ-Acc. On the other hand, only RLAL, DTP [202], and DCP-Adapt are able to outperform their baseline methods. RLAL can both prune 3% more FLOPs and accomplish 0.44% better Δ-Acc than DCP-Adapt. It also has 0.44% higher Δ-Acc than DTP with the same FLOPs reduction ratio. Finally, with the same FLOPs pruning rate, RLAL significantly outperforms AMC [46] with 1.35% higher Δ-Acc. For MobileNet-V2, RLAL can attain the highest Δ-

Acc while having the largest pruning rate at the same time. It remarkably prunes 6.4% more FLOPs than DDNP [174] while obtaining 0.14% higher Δ-Acc. In summary, these results demonstrate the effectiveness of our method for finding efficient yet accurate models.

4.4.2 ImageNet Results

We present the experimental results on ImageNet in Tab. 4.2. For ResNet-18, RLAL shows the best Δ-Acc Top-1/5 while showing a competitive pruning rate. It has a similar pruning rate (only 1% lower) to GNNRL [165] and achieves 0.30% higher Δ-Acc Top-1. For pruning ResNet-34, RLAL is able to find a proper balance between accuracy and efficiency of the model. For instance, with a similar computation budget to GP [207] (only 1.1% FLOPs difference), RLAL's pruned model has 1%/0.46% higher Δ-Acc Top-1/5. Moreover, RLAL shows better final accuracies than ISP [64] while significantly pruning more FLOPs with a 6% margin. Pruning MobileNet-V2 is more challenging compared to ResNets because MobileNets [98, 156] are primarily designed for efficient inference. Accordingly, improvements in metrics are more difficult to secure than ResNet cases. We can observe that all methods have close FLOPs pruning rates in a relatively small range from 28.3% to 30.7%. RLAL can achieve 0.5% higher Δ-Acc Top-1 while pruning only 0.6% lower FLOPs compared to AMC, and it has the best Δ-Acc Top-1 among baselines. These results illustrate the capability of our method to effectively prune both large and small size models. Further, we highlight the advantages of our method compared to baseline RL-based pruning methods, GNNRL [165] and AMC [46], as it can obtain more accurate pruned models while not requiring a pretrained model for pruning.



(a) Pruning Rate: 35% (b) Pruning Rate: 50% (c) Pruning Rate: 65%

(d) Pruning Rate: 35% (e) Pruning Rate: 50% (f) Pruning Rate: 65%

(g) Pruning Rate: 35% (h) Pruning Rate: 50% (i) Pruning Rate: 65%

Figure 4.2: Results of ablation experiments on CIFAR-10. **(a-c)** Best reward of our agent when using a different number of episodes per epoch for three pruning rates when pruning ResNet-56. **(d-f)** Best reward with/without using our mechanism to provide representations of the environment to our agent during training for three pruning rates for ResNet-56. **(g-i)** Same results of **(d-f)** for MobileNet-V2.

4.4.3 Ablation Studies

We conduct ablation experiments to explore our method's behavior by studying 1) the effect of changing the number of episodic observations of the agent in each epoch and 2) the advantage of using our soft regularization, epoch embeddings, and the recurrent environment model in our framework. We refer to supplementary S4.2 for details of experimental

Table 4.3: Ablation Results of our method for pruning ResNet-56 on the CIFAR-10 dataset. EE represents the **E**poch **E**mbeddings. SR represents the **S**oft **R**egularization in Eq. 4.10.

Setting	Baseline Acc	Pruned Acc	Δ-Acc	Pruned FLOPs
w/o EE	93.47%	93.44%	−0.03%	50%
w/o EE + w/o SR	93.33%	93.12%	−0.21%	
Ours	93.41%	93.86%	+0.45%	

settings.

Changing the Number of Episodes: We experiment using a ResNet-56 [39] model on CIFAR-10 with three different pruning rates in $\{35\%, 50\%, 65\%\}$, and we set the number of episodic observations for our agent in each epoch from $\{5, 10, 15\}$. For each pruning ratio, we visualize the best reward that the agent achieves during the training *vs.* the epoch numbers. The results are shown in Fig. 4.2 (a-c). We can observe a common trend in all cases that increasing the number of episodes results in a higher final reward, especially, the higher number of episodes benefits more when the desired compression ratio is larger at the cost of longer training time. However, if the number of episodes is large enough, our method can attain a decent final reward value in a reasonable time.

Benefit of the Recurrent Environment Model: In our second experiment, we prune and finetune ResNet-56 and MobileNet-V2 [98] with three pruning rates in $\{35\%, 50\%, 65\%\}$ while using/dropping our mechanism to provide a representation of the environment to the agent using the epoch embeddings and our recurrent model. We visualize the best reward of the agent in the course of training. The results for ResNet-56 and MobileNet-V2 are shown in Fig. 4.2 (d-f) and Fig. 4.2 (g-i), respectively. The cases using/not using our mechanism are shown with 'w Emb'/'w/o Emb.' The results clearly demonstrate the benefit of our

design that provides a representation of the environment to the agent. We can find that 'w/o Emb' cases commonly reach to a relatively high reward but cannot properly deal with the dynamic reward function for their agent to further improve their policy. In contrast, our method can consistently enhance its policy to reach higher reward values during training.

In our third experiment, we prune ResNet-56 on CIFAR-10 with two settings: 1) not using the recurrent model and epoch embeddings to provide representations of the environment to the agent 2) neither using the recurrent model and epoch embeddings nor the soft regularization. We present the results in Tab. 4.3. One can notice that removing each component of our method degrades its performance, especially not using the recurrent model and epoch embeddings severely degrades our method's accuracy, which is inline with the results presented in Fig. 4.2 and discussed above. In summary, our ablation studies illustrate the effectiveness of our design choices in our method for jointly training and pruning a CNN model.

4.5 Chapter Summary and Conclusion

We proposed a method for structural pruning of a CNN model that jointly trains its weights and prunes its channels using an RL agent. Our method iteratively updates the model's weights and allows the agent to observe several episodic pruning trajectories that it performs on the model to update its policy. Our agent's actions determine the pruning ratios for the layers of the model, and we set the resulting model's accuracy to be the agent's reward. Such a design brings about a dynamic reward function for the agent. Thus, we developed a mechanism to model the complex dynamics of the reward function and yield

a representation of it to the agent. To do so, we mapped the index of each epoch of the training to an embedding. Then, we employed a recurrent model that takes the embeddings and provides a representation of the evolving environment's state to the agent. We train the recurrent model and embeddings by utilizing a decoder model that predicts the agent's rewards given observed states, actions, and environment representations predicted by the recurrent model. Finally, we regularized the model's weights to align with the sub-network selected by the agent's policy with the highest reward during training. Our designs enable the agent to effectively leverage the environment representations along with its episodic observations to learn a proper policy for pruning the model while interacting in our non-stationary pruning environment. Our experiments on ImageNet and CIFAR-10 demonstrate that our method can achieve competitive results with prior methods, especially the ones that use RL for pruning, while not requiring a pretrained model before pruning like them.

Supplementary Materials for Chapter 4

S4.1 Bounding our Agent's Actions

As mentioned in Section 4.3.2.2, we calculate the minimum ($a_{l,min}$) and maximum ($a_{l,max}$) feasible pruning rates for the l-th layer before pruning it to ensure that reaching the desired FLOPs budget, FLOPs_{desire}, is still possible after doing so. However, before formally introducing our scheme for calculating $a_{l,min}, a_{l,max}$, we present how we implement our pruning actions in practice.

S4.1.1 Implementation of our Agent's Actions

We describe our implementation for our agent's actions for each architecture. For all models, we take each block of a CNN model as one 'layer' in our framework.

ResNets: for our experiments on ResNet [39] models (ResNet-56 on CIFAR-10 [5] and ResNet-18/34 on ImageNet [31]), we take each residual block as one layer. It contains a structure as `Conv1-BN-ReLU-Conv2-BN` where `Conv1` and `Conv2` are the convolution layers, `BN` represents Batch Normalization [146], and `ReLU` is the ReLU activation function. For each block, given the predicted action a_l for pruning it, we remove $\lfloor a_l \times c \rfloor$ output channels of the `Conv1` layer and the same number of input channels of the `Conv2` layer where c is the number of output/input channels of `Conv1`/`Conv2`.

MobileNet-V2: for experiments using MobileNet-V2, we take each inverted residual block [98] as one layer for pruning. Each block has the structure with `Conv1-BN-ReLU6-DW_Conv-BN-ReLU6-Conv2-BN` form where `DW_Conv` is a depth-wise con-

volution layer. Given the predicted action a_l, we remove $\lfloor a_l \times c \rfloor$ output channels of `Conv1` and the same amount of channels of `DW_Conv` and input channels of `Conv2`.

In summary, our pruning scheme changes the inner number of channels in each block of a CNN and preserves its number of input and output channels.

S4.1.2 Calculating Action Bounds

We calculate $a_{l,min}, a_{l,max}$ for the l-th layer based on the total model's FLOPs that we denote with FLOPs_T the number of FLOPs for the previous pruned layers $\text{FLOPs}_{1:l-1}$; the number of FLOPs for the next remaining layers $\text{FLOPs}_{l+1:L}$; $\text{FLOPs}[l]$; and FLOPs_{desire}. The formulations are as follows:

$$a_{l,min} = 1 - \frac{\text{FLOPs}_{desire} - \text{FLOPs}_{1:l-1}}{\text{FLOPs}[l]} \tag{4.12}$$

$$a_{l,max} = 1 - \frac{\text{FLOPs}_{desire} - \text{FLOPs}_{1:l-1} - \text{FLOPs}_{l+1:L}}{\text{FLOPs}[l]} \tag{4.13}$$

In these equations, $a_{l,max}$ prevents very high pruning rates that even if all the next layers are kept intact, reaching FLOPs_{desire} get infeasible. Similarly, $a_{l,min}$ provides the minimum pruning rate for the current layer given all the next layers are pruned completely. We clip the predicted action a_l to lie in $[a_{l,min}, a_{l,max}]$ when pruning the l-th layer.

S4.2 Experimental Settings

We provide more details of our experimental settings in the following.

CIFAR-10: For CIFAR-10 experiments, we evaluate our method on ResNet-56 [39] and MobileNet-V2 [98]. In our iterative pruning phase, we train both of the CNN models for 200 epochs with the batch size of 128 using SGD with momentum [211] of 0.9, weight decay of 1e−4, and starting learning rate of 0.1. We decay the learning rate by 0.1 on epochs 100 and 150. We take 5000 samples of the training dataset as a subset for calculating the agent's reward. For all cases, we start to train the RL agent after 10 warmup epochs of the model's weights. Specifically, we collect initial data for the replay buffer of the RL agent from epochs 10 to 20. Then, for both ResNet-56 and MobileNet-V2, we update the agent from epoch 20 until the epoch 90, and we train only the model's weights from epoch 90 to 200. After the iterative stage, we prune the model's architecture and finetune it with the same settings for the base model.

ImageNet: We use ResNet-18, ResNet-34, and MobileNet-V2 for ImageNet experiments. For the iterative training stage of ResNets, we use SGD as the optimizer with the momentum of 0.9, weight decay of 1e−4, and the start learning rate of 0.1. We train ResNet models for 90 epochs, and we decay the learning rate to 0.01 and 0.001 at epochs 30 and 60. For MobileNet-V2, we do so for 155 epochs with a batch size of 256. We train the model's weights using SGD with the momentum of 0.9, weight decay of 1e−4, and starting learning rate of 0.05 decayed using cosine scheduling [212]. For all cases, we use 50000 samples of the training dataset to evaluate rewards of the agent. Similar to CIFAR-10 experiments, we train the model's weights for 10 warmup epochs followed by 10 epochs for filling the replay buffer of the RL agent. Then, for MobileNet-V2, we train the agent's policy from epochs 20 to 90, and we only train the model's weights from epoch 90 to 155. After the pruning stage, we fine-tune the pruned model with the same training parameters as the

base model. For ResNet models, we train the agent's policy from epochs 20 to 70, and the model's weights are trained from epochs 70 to 90.

Ablation Experiments: We follow the same settings as mentioned above for our ablation experiments in Tab. 4.3. For the visualizations in Fig. 4.2, we use the same settings except that we perform our iterative pruning scheme for 90 epochs.

Part II

Pruning Methods for Efficient Inference of Deep Generative Models

Chapter 5: Compressing Image-to-Image Translation GANs Using Local Density Structures on Their Learned Manifold

5.1 Introduction

Image-to-Image translation [213, 214] Generative Adversarial Networks (I2IGANs) [215] have shown excellent performance in many real-world computer vision applications: style transfer [216], converting a user's sketch to a real image [217], super resolution [218, 219], and pose transfer [220, 221]. Yet, I2IGANs require excessive compute and memory resources. Moreover, the mentioned tasks require real-time user interaction, making it infeasible to deploy I2IGANs on mobile and edge devices in Artificial Intelligence of Things (AIoT) with limited resources. Thus, developing compression schemes for GANs to preserve their performance and reduce their computational burden is highly desirable. As the training dynamics of GAN models are notoriously unstable, GAN compression is much more challenging than pruning other deep models like Convolutional Neural Network (CNN) classifiers.

Despite that notable efforts have been made to compress CNNs [41, 43, 44, 47, 98, 222], GAN compression has only been explored in recent years. Early works have proposed a combination of prominent CNN pruning techniques like Neural Architecture Search (NAS) [223,

224, 225, 226, 227], Knowledge Distillation [228, 229, 230, 231, 232, 233, 234, 235], and channel pruning [44, 227] to prune GANs. However, GAN Slimming [229] demonstrated that heuristically stacking several CNN pruning methods for GAN compression can degrade the final performance mainly due to the instabilities of GAN training. GCC [236] empirically showed the importance of restricting the discriminator's capacity during compression. It demonstrated that the previous methods' unsatisfactory performance might be because they only pruned the generator's architecture while using the full-capacity discriminator. By doing so, the adversarial game cannot maintain the Nash Equilibrium state, and the pruning process fails to converge appropriately. Although some of these methods have shown competitive results [225, 236], they do not explicitly consider an essential characteristic of GANs as generative models during pruning, which is their *density structure over their learned manifold.*

In this chapter, we propose a novel GAN Compression method by enforcing the similarity of the density structure of the original parameter-heavy model and the pruned model over the learned manifold of the original model. Our intuition is that the difference in density structures can serve as the supervision signal for pruning. Specifically, at first, we partition the learned manifold of the original model into local neighborhoods. We approximate each neighborhood with a generated sample and its nearest neighbors on the original model's manifold. We leverage a pretrained self-supervised model fine-tuned on the training dataset to find the neighborhoods. Then, we introduce an adversarial pruning objective to encourage the pruned model to have a similar local density structure to the original model on each neighborhood. By doing so, we break down the task of preserving the whole density structure of the original model on its learned manifold into maintaining

local density structures on neighborhoods of its manifold, which resembles kernel density estimation [237]. In addition, we design a new adversarial GAN compression scheme in which two pruning agents (we call them gen_G and gen_D, which determine the structure of the generator G and discriminator D) collaboratively play our proposed adversarial game. Specifically, each agent takes the architecture embedding of its colleague as its input to determine the structure of its corresponding model in each iteration. By doing so, gen_G and gen_D will be able to effectively preserve the balance between the capacities of G and D and keep the adversarial game close to the Nash Equilibrium state during the pruning process. We summarize our contributions as follows:

- We propose a novel GAN compression method that encourages the pruned model to have a similar local density structure as the original model on neighborhoods of the original model's learned manifold.
- We design two pruning agents that collaboratively play our adversarial pruning game to compress both the generator and discriminator together. By doing so, our method can effectively preserve the balance between the capacities of the generator and discriminator and show more stable pruning dynamics while outperforming baselines.
- Our extensive experiments on Pix2Pix [213] and CycleGAN [214] on various datasets demonstrate our method's effectiveness.

The contents of this chapter are based on our work [238] published in AAAI 2024.

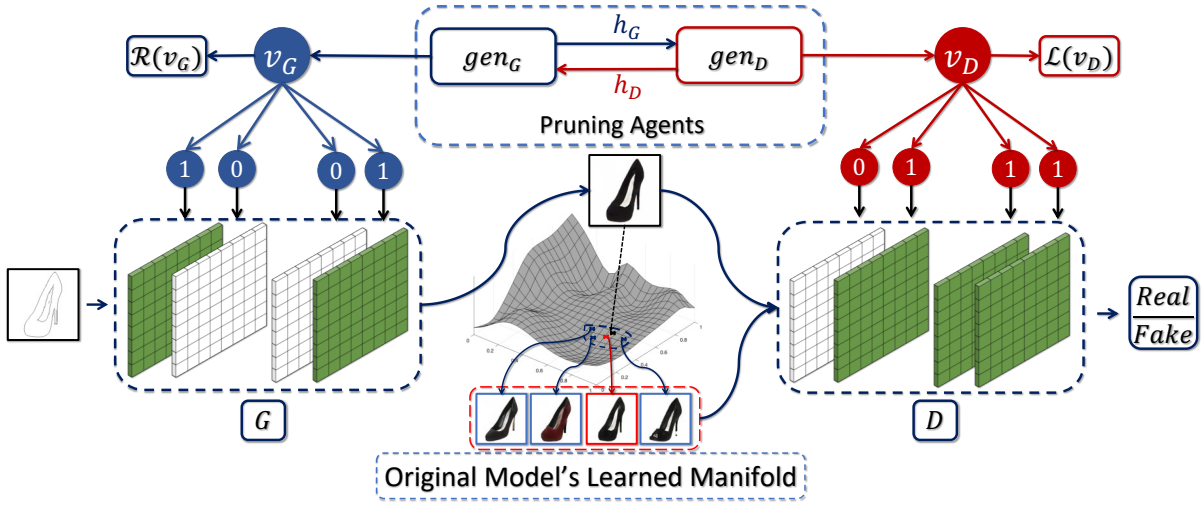


Figure 5.1: Our GAN pruning method. We encourage the pruned generator to preserve the density structure of the original model over its learned manifold during pruning. To do so, we partition the manifold into local neighborhoods around the samples generated by the original generator (Fig. 5.2) and represent each local neighborhood with a 'Center' sample (shown with a red frame) and its neighbors (blue frames). We use these samples as 'real' samples and the one generated by the pruned generator as a 'fake' one in our adversarial pruning objective. We implement our adversarial game with two pruning agents, gen_G and gen_D, that collaboratively learn to prune the original pretrained G and D. gen_G (gen_D) takes the architecture embedding of their colleague gen_D (gen_G) when determining the architecture of G (D). By doing so, gen_G and gen_D can maintain the balance between the capacity of G and D during pruning and make the process stable. (Fig. 5.4)

5.2 Related Work

GAN Compression: GANs require two orders of magnitude more computation than CNNs [224]. Hence, GAN compression is crucial prior to deploying them on edge devices. Search-based methods [224, 239, 240, 241] search for a lightweight architecture for the generator but are extremely costly due to their vast search space. Pruning methods [224, 225, 229, 242] prune the redundant weights of the generator's architecture but neglect the discriminator. They result in an unbalanced generator and discriminator capacities, leading to mode collapse [236]. To address this problem, discriminator-free methods [232, 243] distill the generator into a compressed model without using the discrimina-

tor. In another direction, GCC [236] and Slimmable GAN [233] prune both the generator and discriminator together. Slimmable GAN sets the discriminator's layers' width proportional to the ones for the generator during pruning. Yet, GCC empirically showed there is no linear relation between the number of channels of the generator and optimal discriminator, and Slimmable GAN's approach is sub-optimal. Inspired by GCC, we use two pruning agents that learn to determine the architectures of the generator and discriminator in our proposed adversarial game. Each agent gets feedback from its peer when determining its corresponding model's architecture. Thus, they can effectively preserve the balance between the generator and discriminator and stabilize the pruning process.

Manifold Learning for GANs: The manifold hypothesis indicates that high-dimensional data like natural images lie on a nonlinear manifold with much smaller intrinsic dimensionality [244]. Accordingly, a group of methods alter the training [245] and/or architecture of GANs by using several generators [246], including a manifold learning step in the discriminator [247], and local coordinate coding based sampling [248]. Yet, one cannot use these methods for pruning GANs that are pretrained with other methods. Another group of ideas observed that semantically meaningful paths and neighborhoods exist in the latent space of trained GANs. [249, 250]. Inspired by these methods, we propose to partition the learned manifold of a pretrained GAN into overlapping neighborhoods. Then, we develop an adversarial pruning scheme that encourages the pruned model to have a similar density structure to the original one over each neighborhood.

Network Pruning: Model compression [170] is a well-studied topic, and proposed methods have primarily focused on compressing CNN classifiers. They use various techniques like weight pruning [41], light architecture design [120, 156], weight quantization [43], structural

pruning [44, 46, 47, 64], knowledge distillation [251], and NAS [119, 252]. We focus on developing a compression method for GANs, which is a more challenging task due to the instability and complexity of their training [229].

5.3 Proposed Method

We develop a new GAN compression method that explicitly regularizes the pruned model not to forget the density structure of the original one over its learned manifold. However, directly applying such regularization along with compression objectives can make the pruning process unstable because of the complex nature of training GANs. Thus, we simplify this objective for the pruned generator in two steps. At first, we propose to partition the learned manifold of the original model into local neighborhoods, each consisting of a sample and its nearest neighbors on the manifold. We employ a self-supervised model fine-tuned on the training dataset to approximately find such neighborhoods. Then, we propose an adversarial pruning objective to enforce the pruned model to have a density structure similar to the original model over each local neighborhood. Finally, we introduce a novel GAN compression scheme in which we use two pruning agents - called gen_G and gen_D - that determine the architectures of the generator G and discriminator D, respectively. gen_G and gen_D play the adversarial game introduced in the previous step in a collaborative manner to find the optimal structure of G and D. In each step, gen_G (gen_D) gets feedback from its peer gen_D (gen_G) about the architecture of D (G) when determining the architecture of G (D). By doing so, they can maintain the balance between the generator and discriminator during pruning and stabilize the pruning process. We show our pruning scheme in Fig. 5.1.

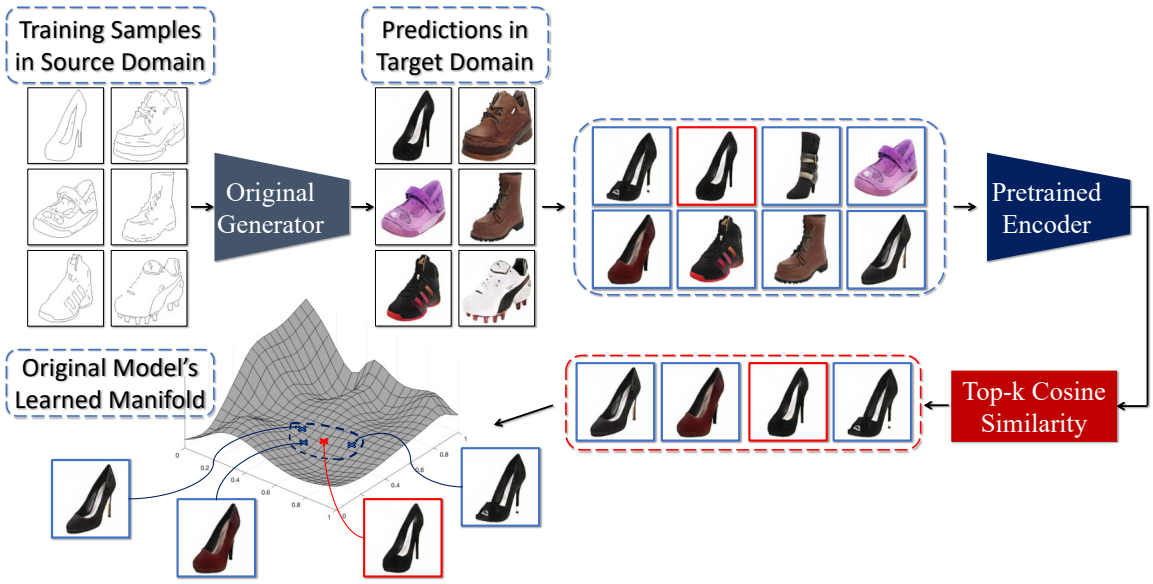


Figure 5.2: Our method to find local neighborhoods on the learned manifold of the original generator. **(Top Left):** First, we obtain the original model's predictions in the target domain for training samples in the source domain. **(Right and Down):** We call the sample that we want to find its local neighborhood on the manifold 'Center' sample (shown with Red solid frame). We pass the predicted samples in the previous step to a pretrained self-supervised encoder [6] that is fine-tuned on the target images in the training dataset. Then, we take the samples whose representations have the highest cosine similarity with the representation of the 'Center' sample as its approximate neighbors on the manifold. Neighbor samples and the approximate neighborhood on the manifold are shown with blue crosses and a dashed line.

5.3.1 Notations

We denote an I2IGAN model's source and target domains with $\mathcal{X}$ and $\mathcal{Y}$, respectively. We show the training dataset as $\mathcal{D} = \{\{(x_i)\}_{i=1}^{N}, \{(y_j)\}_{j=1}^{M}\}$ such that $(x, y) \sim \mathcal{P}(\mathbf{x}, \mathbf{y})$ where $\mathcal{P}$ is the underlying joint distribution over the source and target domains. N and M can be equal for the paired datasets [213] or be unequal for unpaired models [214]. We represent the generator and discriminator with G and D. Also, we denote pruning agents determining the architectures of G and D during pruning with $gen_G(\cdot)$ and $gen_D(\cdot)$. The goal of the generator is to learn the distribution of corresponding $y \in \mathcal{Y}$ in the target domain conditioned on a sample $x \in \mathcal{X}$ from the source domain. We denote the learned

manifold of the original generator in the domain $\mathcal{Y}$ with $\mathcal{M}_y$.

5.3.2 Finding Local Neighborhoods on the Learned Data Manifold of the Original Generator

As mentioned above, our idea is to guide the pruning process of a GAN model by regularizing the pruned generator to have a similar density structure to the original model over its learned manifold, $\mathcal{M}_y$. To simplify this objective for the pruned model, we partition $\mathcal{M}_y$ into local neighborhoods $\mathcal{N}_{y_i}$ containing a sample y_i and its nearest neighbors on the manifold $\mathcal{M}_y$. The intuition is that separately modeling the density structure over each neighborhood is easier than modeling all of them simultaneously, which resembles the kernel density estimation method [237].

To find the local neighborhoods, on the one hand, we get inspirations from recent works that observed that latent spaces of GAN models have semantically meaningful paths and neighborhoods [249, 253, 254, 255, 256]. On the other hand, self-supervised pretrained encoders [6, 257, 258, 259] have shown significant results in unsupervised clustering and finding semantically similar samples without using labels on the manifold of their input data. Accordingly, at first, we fine-tune an encoder pretrained by SwAV [6] on the samples $\{(y_j)\}_{j=1}^{M}$ in the training dataset. Then, we use the training dataset and pass the training samples x_i into the original generator to obtain its predicted samples $\mathcal{D}'_y = \{y'_i\}_{i=1}^{N}$ on $\mathcal{M}_y$. Finally, we approximately model the neighborhood of y'_i over $\mathcal{M}_y$ by finding its k nearest neighbor samples in $\mathcal{D}'_y$ using our fine-tuned encoder. Formally, given a fine-tuned self-supervised encoder E, we find k nearest neighbors of y'_i in $\mathcal{D}'_y$, denoted by $\mathcal{N}_{y_i,k}$, as

follows: (Fig. 5.2)

$$\begin{aligned} \mathcal{E}' &= \{e'_j\}_{j=1}^{N},\ e'_j = E(y'_j) \\ Sim_i &= \{Cosine(e'_i, e'_j)\}_{j=1,\ j\neq i}^{N} \\ Cosine(e'_i, e'_j) &= ||e_i'^{T} e'_j||/(||e'_i||||e'_j||) \\ \mathcal{N}_{y'_i,k} &= \{y'_j | e'_j \in \text{Top-}k(Sim_i)\} \end{aligned} \tag{5.1}$$

i.e., we take samples in $\mathcal{D}'_y$ that their encoded representations have the highest cosine similarity with the one for y'_i as its neighbors on $\mathcal{M}_y$. We use the cosine similarity metric as it has been widely used in the nearest neighbor classifiers [260, 261] and self-supervised learning [6].

5.3.3 Pruning

In this section, first, we elaborate on our pruning objective. Then, we introduce our pruning agents.

Pruning Objective: We regularize the pruned generator to have a similar density structure to the original one over each local neighborhood of the learned manifold of the original model ($\mathcal{M}_y$). Formally, we approximately represent the neighborhood around each sample y'_i with samples in $\mathcal{N}_{y'_i,k}$ in Eq. 5.1. Then, we define our adversarial training objective to regularize the pruned model to preserve the local density structure of the original model in each neighborhood:

$$\begin{aligned} &\min_{\theta_G}\max_{\theta_D} \mathbb{E}_{(x,y')\sim\mathcal{P}'(\mathbf{x},\ \mathbf{y})}[\mathbb{E}_{y''\sim\mathcal{N}_{y'}}[f_D(D(x,y'';v_D))]]+ \\ &\mathbb{E}_{x\sim\mathcal{P}(\mathbf{x})}\mathbb{E}_{\xi}[f_G(D(x,G(x,\xi;v_G);v_D))] + \lambda_1\mathcal{R}(v_G) - \lambda_2\mathcal{L}(v_D) \end{aligned} \tag{5.2}$$

$\mathcal{P}'(y)$ is the marginal distribution of the original generator model on $\mathcal{Y}$. G and D are the pretrained generator and discriminator. θ_G and θ_D are parameters of gen_G and gen_D. v_G and v_D are architecture vectors that determine the structures of G and D. They are functions of θ_G and θ_D. f_D and f_G are GAN objectives that are least squares for CycleGAN [214] and hinge loss for Pix2Pix [213]. ξ represents the randomness in the generator implemented with Dropout [262] for Pix2Pix and Cycle GAN. $\mathcal{R}$ is the regularization term to enforce the desired compression ratio on the generator, and $\mathcal{L}$ imposes sparsity on the architecture of the discriminator. Penalizing unimportant components for discriminator is crucial to maintain the capacity balance between G and D during pruning and keeping them close to the Nash Equilibrium state, as pointed out by GCC [236]. Moreover, our objective is similar to the Kernel Density Estimation (KDE) [237] method, which aims to model the local density around each data point. Yet, in contrast with KDE, we use an implicit objective to encourage the pruned model to have a similar density structure to the original one in our method. In addition, in Obj. 5.2, the parameters of G and D are inherited from the original pretrained models, and we do not train parameters of G/D along with θ_G/θ_D to prevent instability. We note that our pruning objective does not require paired images during pruning, even for GAN models such that use a paired dataset for training. The reason is that our objective employs the samples generated by the original generator to approximate its density structure over each local neighborhood. Thus, it can readily prune both paired [213] and unpaired conditional GANs [214].

Pruning Agents: Inspired by GCC [236] that demonstrated that preserving the balance between the capacity of G and D is crucial for pruning stability, we prune both G and D during the pruning process. To do so, we introduce a novel GAN compression scheme in

which we use two pruning agents, gen_G and gen_D, to predict 1) architecture vectors and 2) architecture embeddings for G and D. The former is a binary vector determining the architecture of the corresponding model (G/D), and the latter is a compact representation describing its state (architecture). To preserve the balance between the capacity of G and D, we design our pruning scheme such that each pruning agent gen_G/gen_D considers the architecture embedding of D/G when determining the architecture of G/D. We implement pruning agents with a GRU [263] model and dense layers (more details in supplementary S5.2) and take the outputs of the GRU units of the pruning agents gen_G/gen_D as their corresponding model's (G/D) architecture embedding, summarizing the information about its architecture.

In each step of the adversarial game, gen_G and gen_D exchange their architecture embeddings. Then, each of them determines its corresponding model's architecture while knowing the other model's structure (architecture embedding), making the pruning process stable and efficient (Fig. 5.1). We denote the architecture vector determining the architecture of G/D with $v_G/v_D \in \{0, 1\}$. As these vectors have discrete values, their gradients $w.r.t$ agents' parameters cannot be calculated directly. Thus, we use Straight-through Estimator (STE) [139] and Gumbel-Sigmoid reparametrization trick [91] to calculate the gradients. The sub-network vector v_G is calculated as:

$$\begin{aligned} v_G &= \text{round}(\text{sigmoid}(-(o_G + g)/\tau)), \\ o_G, h_G &= gen_G(h_D; \theta_G) \end{aligned} \tag{5.3}$$

where round$(\cdot)$ rounds input to its nearest integer, sigmoid$(\cdot)$ is the sigmoid function, τ is

the temperature parameter to control the smoothness, and $g \in \text{Gumbel}(0, 1)$ is a random vector sampled from the Gumbel distribution [264]. h_D is the architecture embedding for D, which is the input for gen_G. o_G and h_G are the output of gen_G and its architecture embedding for G. Similarly, v_D is calculated as:

$$\begin{aligned} v_D &= \text{round}(\text{sigmoid}(-(o_D + g)/\tau)), \\ o_D, h_D &= gen_D(h_G; \theta_D) \end{aligned} \tag{5.4}$$

The calculation from o_* to v_* ($* \in \{G, D\}$) can be seen as using straight-through Gumbel-Sigmoid [91] to approximate sampling from the Bernoulli distribution. We provide our pruning algorithm and details of the calculation of o_* and h_* in the supplementary S5.2.2.

Predicted architecture vectors v_G and v_D determine the architectures of G and D. For the generator G, we aim to reduce MACs to reach a given budget. To do so, we use the following regularization objective:

$$\mathcal{R}(v_G) = \log(\max(T(v_G), pT_{\text{total}})/pT_{\text{total}}), \tag{5.5}$$

where p is the predefined threshold for pruning, $T(v_G)$ is the MACs of the current sub-network chosen by v_G, and T_{total} is the total prunable MACs.

Different from G, it is not obvious how to set a specific computation budget for D, as shown in GCC [236]. Instead of using a predefined threshold, we encourage gen_D to automatically identify the sub-network that can keep the Nash Equilibrium given the v_G. Inspired by the functional modularity [265], we add a penalty to v_D to discover the suitable

sub-module (sub-network) to keep the Nash Equilibrium:

$$\mathcal{L}(v_D) = \sum v_D/|v_D|, \tag{5.6}$$

where $|v_D|$ is the number of elements in v_D. The goal of Eq. 5.6 is to penalize unimportant elements in v_D, so the remaining elements can maintain Nash Equilibrium given v_G.

Finetuning: After the pruning stage, we employ the trained gen_G/gen_D from the pruning process to prune G/D according to their predictions, v_G/v_D. Then, we finetune G and D together with the original objectives of their GAN methods [213, 214]. Similar to GCC [236], we also apply knowledge distillation (KD) for finetuning, but we show in our ablation experiments that our model can achieve high performance even without KD.

5.4 Experiments

We perform our experiments with Pix2Pix and Cycle-GAN models. For all experiments, we set $\lambda_1 = 3.0$, $\lambda_2 = 0.1$, and the number of neighbor samples $k = 5$ during pruning. We also find that our model is not very sensitive to the choice of λ_1 and λ_2. (more details in ablation experiments) We set the number of pruning epochs to 10% of the original model's training epochs. We refer to supplementary S5.3 for more details of our experimental setup.

5.4.1 Results

Comparison with State of the Art Methods: We summarize the quantitative results of our method and baselines in Tab. 5.1. As can be seen, our method, **MGGC**

Table 5.1: Quantitative comparison of our proposed method with state-of-the-art GAN compression methods.

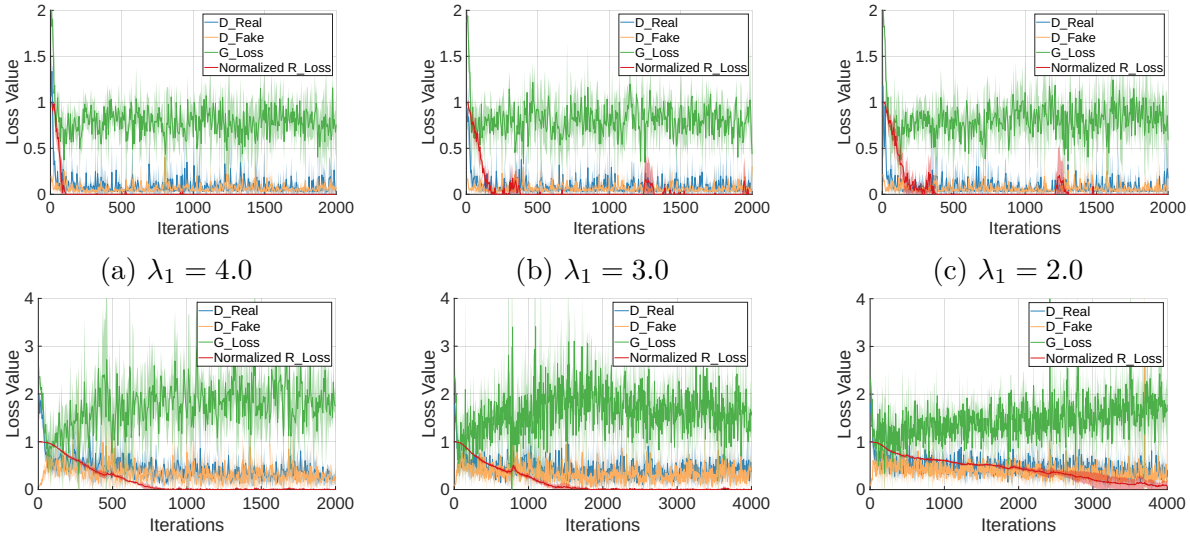
Model	Dataset	Method	MACs	Compression Ratio	Metric	
					FID (↓)	mIoU (↑)
Pix2Pix	Cityscapes	Original [213]	18.60 G	-	-	42.71
		GAN Compression [224]	5.66 G	69.57%	-	40.77
		CF-GAN [241]	5.62 G	69.78%	-	42.24
		CAT [225]	5.57 G	70.05%	-	42.53
		DMAD [227]	3.96 G	78.71%	-	40.53
		WKD [234]	3.88 G	79.13%	-	42.93
		RAKD [235]	3.88 G	79.13%	-	42.81
		Norm Pruning [44]	3.09 G	83.39%	-	38.12
		GCC [236]	3.09 G	83.39%	-	42.88
		MGGC (Ours)	3.05 G	83.60%	-	**44.63**
	Edges2Shoes	Original [213]	18.60 G	-	34.31	-
		Pix2Pix 0.5× [213]	4.65 G	75.00%	52.02	-
		CIL [266]	4.57 G	75.43%	44.40	-
		DMAD [227]	2.99 G	83.92%	46.95	-
		WKD [234]	1.56 G	91.61%	80.13	-
		RAKD [235]	1.56 G	91.61%	77.69	-
		MGGC (Ours)	2.94 G	84.19%	**42.02**	-
CycleGAN	Horse2Zebra	Original [214]	56.80 G	-	61.53	-
		Co-Evolution [239]	13.40 G	76.41%	96.15	-
		GAN Slimming [229]	11.25 G	80.19%	86.09	-
		AutoGAN-Distiller [232]	6.39 G	88.75%	83.60	-
		WKD [234]	3.35 G	94.10%	77.04	-
		RAKD [235]	3.35 G	94.10%	71.21	-
		GAN Compression [224]	2.67 G	95.30%	64.95	-
		CF-GAN [241]	2.65 G	95.33%	62.31	-
		CAT [225]	2.55 G	95.51%	60.18	-
		DMAD [227]	2.41 G	95.76%	62.96	-
		Norm Prune [44]	2.40 G	95.77%	145.1	-
		GCC [236]	2.40 G	95.77%	59.31	-
		MGGC (Ours)	2.50 G	95.60%	**55.06**	-
	Summer2Winter	Original [214]	56.80 G	-	79.12	-
		Co-Evolution [239]	11.06 G	80.53%	78.58	-
		AutoGAN-Distiller [232]	4.34 G	92.36%	78.33	-
		DMAD [227]	3.18 G	94.40%	78.24	-
		MGGC (Ours)	1.69 G	97.02%	**77.33**	-
		MGGC (Ours)	2.97 G	94.77%	**75.85**	-

(**M**anifold **G**uided **GAN** **C**ompression), can achieve the best performance *vs.* computation rate trade-off compared to baselines in all experiments. For Pix2Pix on Cityscapes, MGGC reduces MACs by 83.60%, achieving the highest MACs compression rate, and improves mIoU by 1.92 compared to the original model, significantly outperforming baselines. On Edges2Shoes, MGGC prunes 0.27% more MACs than DMAD [227] while outperforming it

with a large margin of 4.93 FID. Although WKD [234] and RAKD [235] achieve higher compression ratio, their final model has significantly worse FID. For CycleGAN on Horse2Zebra, MGGC shows a pruning ratio very close (95.60% *vs.* 95.77%) to GCC [236] and accomplishes 55.06 FID, significantly outperforming GCC by 4.25 FID. On Summer2Winter, MGGC attains 94.77% MACs compression rate, slightly more than DMAD with 94.40%, and shows 2.39 less FID. Remarkably, it can outperform other baselines even with an extreme MACs compression rate of 97.02%, and yet, reaching 77.33 FID. In summary, our results demonstrate the effectiveness of our method that explicitly focuses on the differences between the density structure of the pruned model and the original one over its learned manifold during pruning.

Figure 5.3: Qualitative results for **1) Pix2Pix:** Cityscapes (top left), Edges2Shoes (bottom left), and **2) CycleGAN:** Horse2Zebra (top right) and Summer2Winter (bottom right).

Qualitative Results: We visualize predictions of our pruned models and the original ones in Fig. 5.3. As can be seen, our pruned model can preserve the fidelity of images with a much lower computational burden. Also, its superior performance compared to baselines is observable in the samples for Cityscapes (better preserving street structure) and Horse2Zebra (more faithful background color).



(a) $\lambda_1 = 4.0$ (b) $\lambda_1 = 3.0$ (c) $\lambda_1 = 2.0$

(d) $\lambda_1 = 4.0$ (e) $\lambda_1 = 3.0$ (f) $\lambda_1 = 2.0$

Figure 5.4: Different losses given different λ_1 during the pruning process. (a)-(c) Loss values for CycleGAN on Horse2Zebra dataset. (d)-(f) Loss values for Pix2Pix on Cityscapes dataset. We normalize $\mathcal{R}$ to the range $[0, 1]$ for better visualization.

5.4.2 Stability of Our Pruning Method

We explore our method's stability by visualizing different loss values and the resource loss $\mathcal{R}$ during pruning. Loss values for CycleGAN on Horse2Zebra are shown in Fig. 5.4 (a)-(c), and the ones for Pix2Pix on Cityscapes are presented in Fig. 5.4 (d)-(f). We found in our experiments that λ_2 has less impact on pruning dynamics than λ_1, which is expected as the generator's capacity is more crucial to GANs' performance than the discriminator's

capacity. Thus, we alter λ_1 to explore how it impacts the pruning process. We can observe that if λ_1 be large enough (Fig. 5.4 (a)-(e)), the loss values for G and D will get stable and remain close to each other when $\mathcal{R}$ converges to zero. Further, the difference between loss values for G and D in Fig. 5.4 (d)-(f) for our model is much smaller than the same value for GCC (shown in GCC [236] Fig. 5(a)), which demonstrates that our method can preserve the balance between G and D more effectively than GCC during pruning. In summary, visualizations in Fig. 5.4 show that our method can successfully preserve the balance between G, D after achieving the desired computation budget, leading to attain competitive final performance metrics.

5.4.3 Ablation Study

We present ablation results of our method with different settings in Tab. 5.2. We construct a Baseline by only pruning the generator G with a naive parameterization $v_G = \text{round}(\text{sigmoid}(-(\theta + g)/\tau))$ (Eq. 5.3) when using the full capacity discriminator. The results demonstrate that using pruning agents, pruning the discriminator, and establishing a feedback connection between gen_G and gen_D provide substantial performance gain compared to the baseline on Pix2Pix and CycleGAN models. This observation suggests that sophisticated designs of pruning agents are beneficial for pruning conditional GAN models. Further, considering local density structures over neighborhoods of the learned manifold ($k > 0$) boosts the performance of our method significantly than not leveraging them ($k = 0$). Remarkably, our method can almost recover the original model's performance for Pix2Pix (42.53 *vs.* 42.71) and even outperform it for CycleGAN (58.64 *vs.* 61.53) with-

Table 5.2: Ablation results of our proposed method.

Settings	Pix2Pix - Cityscapes		Cycle GAN - Horse2Zebra	
	mIoU (↑)	MACs	FID (↓)	MACs
Baseline	39.37	3.05 G	73.28	2.50 G
+ D pruning	39.84		70.41	
+ Pruning Agents	40.68		67.60	
+ G↔D Feedback	40.99		66.72	
+ Manifold Pruning	42.53		58.64	
+ Knowledge Distillation	44.63		55.06	
Original	42.71	18.60 G	61.53	56.80 G

out using Knowledge Distillation (KD) in the finetuning process. These results show that the density structure over the learned manifold contains valuable information for pruning GAN models. Utilizing KD can further improve our model. Compared to GCC [236], we do not use online distillation for KD. Also, we do not learn the discriminator's architecture during finetuning, which saves computational costs.

5.4.4 Visualization of Local Neighborhoods

We explore the difference between the learned neighborhoods of our model *vs.* the original one in Fig. 5.5. Samples with red frames are the predictions, and their neighbors on the right are obtained with the method in Eq. 5.1. In each column, green frames show the samples having the same source domain ('Edge') image, and the blue ones mean different source domain inputs. As can be seen, most of the neighbor samples of our pruned model are identical to those for the pruned model. In addition, the samples with different source images still have a semantically meaningful connection to the predicted image. These results demonstrate that our pruning objective, Eq. 5.2, can effectively regularize the pruned model to have a similar density structure over the neighborhoods of the original model's manifold.

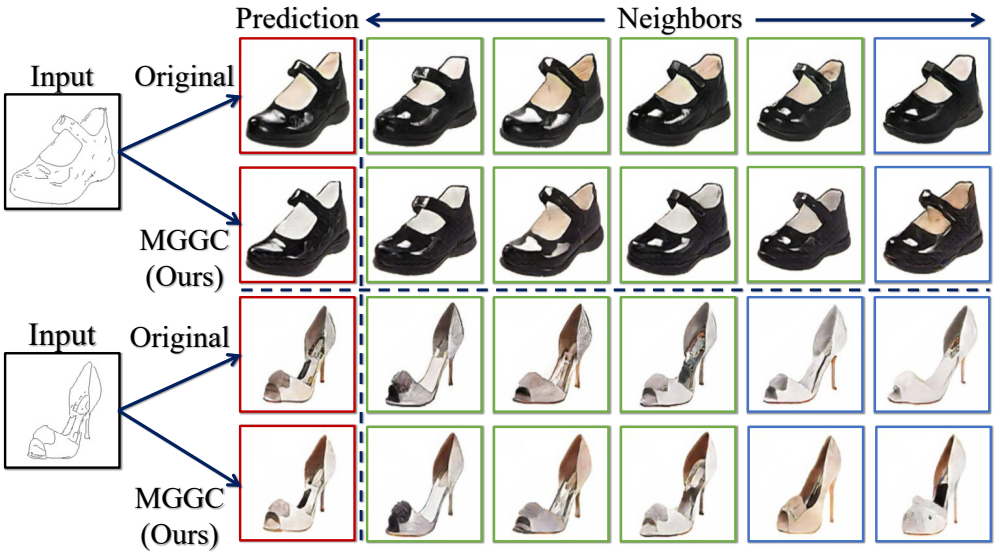


Figure 5.5: Visualization of approximate neighborhoods on the learned manifold of our pruned model *vs.* the original model.

5.5 Chapter Summary and Conclusion

We introduced a new compression method for image translation GANs that explicitly regularizes the pruned model to have a similar density structure to the original one on the original model's learned data manifold. We simplify this objective for the pruned model by leveraging a pretrained self-supervised encoder fine-tuned on the target dataset to find approximate local neighborhoods on the manifold. Then, we proposed our adversarial pruning objective that motivates the pruned generator to have a similar density structure to the original model over each local neighborhood. In addition, we proposed a novel pruning scheme that uses two pruning agents to determine the architectures of the generator and discriminator. They collaboratively play our adversarial pruning game such that in each step, each pruning agent takes the architecture embedding of its colleague as its input and determines its corresponding model's architecture. By doing so, our agents can effec-

tively maintain the balance between the generator and discriminator, thereby stabilizing the pruning process. Our experimental results clearly illustrate the added value of using the learned density structure of a GAN model for pruning it compared to the baselines that directly combine CNN classifiers' compression techniques.

Supplementary Materials for Chapter 5

S5.1 Details of Our Experiments

We mainly follow the experimental settings of the previous works in the literature [224, 227, 236]. We perform our experiments on the prominent image-to-image translation methods, Pix2Pix [213] and CycleGAN [214]. The generator model has a U-Net [93] style architecture for the Pix2Pix experiments [213, 236] and a ResNet [39] style for CycleGAN experiments [214, 224, 236]. For Pix2Pix experiments, we evaluate our model on the Edges2Shoes [267] and Cityscapes [268] datasets. For CycleGAN experiments, we use Horse2Zebra [214] and Summer2Winter [214] datasets. We follow the evaluation metrics in the literature [224, 227, 236] to evaluate our model, *i.e.*, we use mean Intersection over Union (mIoU) for the experiment for Pix2Pix on Cityscapes. We use Fréchet Inception Distance (FID) [269] as our evaluation metric for other experiments. Higher mIoU and Lower FID values mean superior generative capability. For all datasets, we set $\lambda_1 = 3.0$, $\lambda_2 = 0.1$, and the number of neighbor samples $k = 5$ during pruning. We also found that the setting $k \in \{3, 4, 5\}$ results in close final performance. Thus, we set $k = 5$ in all experiments. We implemented our experiments with PyTorch and ran them on a server with 2 NVIDIA P40 GPUs.

S5.2 Our Pruning Agents

We provide implementation details of our proposed pruning agents in this section.

S5.2.1 Architectural Design

We use a Gated Recurrent Unit (GRU) [263] and dense layers to implement our pruning agents. The detailed architecture is shown in Tab. 5.3. The inputs x_*^l are randomly generated, and they are fixed after initialization during training and inference.

Table 5.3: The architecture of gen_* ($* \in \{G, D\}$) used in our method.

Inputs x_*^l, $l = 1, \cdots, L_*$
GRU(128, 256), WeightNorm, ReLU Dense$_l$(256, C_*^l), WeightNorm, $l = 1, \cdots, L_*$
Outputs o_*^l, $l = 1, \cdots, L_*$; $h_*^{L_*}$

S5.2.2 Architecture Vectors and Architecture Embeddings

We calculate architecture vectors v_* and architecture embedding h_* ($* \in \{G, D\}$) for our pruning agents as follows:

$$\begin{aligned} y_*^l, h_*^l &= \text{GRU}(x_*^l, h_*^{l-1}), \\ o_*^l &= \text{Dense}_l(y_*^l), \\ l &= 1, \cdots, L_* \; * \in \{G, D\}, \end{aligned} \tag{5.7}$$

Take the discriminator D as an example, we use let the last layer hidden outputs $h_D^{L_D}$ as the corresponding architecture embedding h_D. We let the initial hidden state $h_D^0 = h_G$. We concatenate all o_*^l to get the final $o_D = [o_D^1, \cdots, o_D^{L_D}]$ used in Eq. 5.4. Similar procedures are applied for the generator G in Eq. 5.3.

Algorithm 3 Our Proposed Pruning Scheme

Input: A pruning dataset of source domain images, their corresponding predictions in the target domain (by the original generator), and the set of neighbors for the predicted images on the original model's learned manifold (Section 3.3) $\mathcal{D}_p = \{(x_i), (y'_i, \mathcal{N}_{y'_i,k})\}_{i=1}^{N}$; Pruning agents $gen_G(\theta_G)$ and $gen_D(\theta_D)$; original generator and discriminator G and D; Number of iterations T

Output: Pruning agents $gen_G(\theta_G)$ and $gen_D(\theta_D)$.

Initialization: Freeze the pretrained weights of G and D and disable gradient calculation for them.

for $t := 1$ *to* T **do**

1. Sample a pruning pair $d = \{(x_i, (y'_i, \mathcal{N}_{y'_i,k}))\}$

`/* ****** Forward Pass of the Adversarial Game ***** */`

2. $h_D \leftarrow gen_D$; `// ` h_D `gets architecture embedding of` gen_D `for` D`.`

3. $o_G, h_G \leftarrow gen_G(h_D.detach(); \theta_G)$
4. $v_G \leftarrow Gumbel\text{-}Sigmoid(o_G)$
5. Determine the architecture of G using v_G.
6. $y^{fake} \leftarrow G(x_i; v_G)$

`/* ********** Updating` gen_D `********** */`

7. $h_G \leftarrow gen_G$
8. $o_D, h_D \leftarrow gen_D(h_G.detach(); \theta_D)$; `// Do not backprop gradients for` gen_G `when updating` gen_D`.`

9. $v_D \leftarrow Gumbel\text{-}Sigmoid(o_D)$
10. Determine the architecture of D using v_D.
11. $p_D^{fake} \leftarrow D(x_i, y^{fake}.detach(); v_D)$, $p_D^{real} \leftarrow [D(x_i, y'_j; v_D)$ for y'_j in $\mathcal{N}_{y'_i,k}]$
12. Calculate Objective 5.2 using p_D^{fake} and p_D^{real} and backpropagate the loss gradients for the parameters of $gen_D(\theta_D)$ using STE [139].
13. Update θ_D using Adam optimizer.

`/* ********** Updating` gen_G `********** */`

14. $h_D \leftarrow gen_D$; `// ` h_D `gets architecture embedding of` gen_D `for` D`.`

15. $o_G, h_G \leftarrow gen_G(h_D.detach(); \theta_G)$; `// Do not backprop gradients for` gen_D `when updating` gen_G`.`

16. $v_G \leftarrow Gumbel\text{-}Sigmoid(o_G)$
17. Determine the architecture of G using v_G.
18. $p_G^{fake} \leftarrow D(x_i, y^{fake}; v_D)$
19. Calculate objective 5.2 using p_G^{fake} and backpropagate the loss gradients for the parameters of $gen_G(\theta_G)$ using STE [139].
20. Update θ_G using Adam optimizer.

end

return $gen_G(\cdot)$, $gen_D(\cdot)$.

S5.3 Experimental Details

As mentioned before, we mainly follow the setup of the previous works [224, 227, 234, 236] in our experiments. We elaborate on our hyperparameter setting for original models' training, pruning, and fine-tuning of them in the following.

S5.3.1 Original Models' Training

We use the original repository[1] for Pix2Pix [213] and CycleGAN [214] to train the original models. We use Adam optimizer [102] with parameters $(\beta_1, \beta_2) = (0.5, 0.999)$ to train the generator and discriminator for all models. The hyperparameter settings for each experiments is summarized in Tab. 5.4. Similar to the original training schemes [213, 214], we use constant learning rate for half of the training and linearly decay it to zero in the rest of it except for Pix2Pix on Edges2Shoes. We also utilize Batch Normalization [146] and Instance Normalization [270] in our experiments for the architectures of Pix2Pix and CycleGAN respectively.

Table 5.4: Hyperparameter settings for training original models.

Model	Dataset	GAN Loss	Batch Size	Training Epochs		Optimization Params		Normalization
				Constant LR	Decay	LR	Weight Decay	
Pix2Pix	Cityscapes	hinge	1	100	100	0.0002	0	Batch
Pix2Pix	Edges2Shoes	hinge	4	5	25	0.0002	0	Batch
CycleGAN	Horse2Zebra	Least Squares	1	100	100	0.0002	0	Instance
CycleGAN	Summer2Winter	Least Squares	1	100	100	0.0002	0	Instance

[1] https://github.com/junyanz/pytorch-CycleGAN-and-pix2pix

S5.3.2 Pruning

We prune the original pretrained GAN models using our proposed Obj. 5.2 and two pruning agents with the architectures described in section S5.2. We set $\lambda_1 = 3.0$, $\lambda_2 = 0.1$, and the number of neighbor samples $k = 5$ during pruning for all the datasets. We also use Adam [102] with parameters $(\beta_1, \beta_2) = (0.9, 0.999)$ for pruning. Other pruning hyperparameters are summarized in Tab. 5.5. For the self-supervised SwAV [6] model, we take the provided checkpoint for ResNet-50, trained on ImageNet for 800 epochs, in their original[2] repository. Then, we fine-tuned the models with batch size of 256 for 10 epochs. We set the number of clusters to 100 for Cityscapes, 200 for Edges2Shoes, and 50 for Horse2Zebra as well as Summer2Winter. We set the other parameters the same as the default ones used in the SwAV [6] training from scratch. We provide our pruning algorithm in alg. 3.

Table 5.5: Hyperparameter settings for pruning agents.

Model	Dataset	Manifold Loss	Batch Size	Pruning Epochs	Optimization Params	
					LR	Weight Decay
Pix2Pix	Cityscapes	hinge	1	20	0.001	0.0001
Pix2Pix	Edges2Shoes	hinge	4	3	0.001	0.0001
CycleGAN	Horse2Zebra	Least Squares	1	20	0.001	0.0001
CycleGAN	Summer2Winter	Least Squares	1	20	0.001	0.0001

S5.3.3 Fine-tuning

After pruning stage, we employ the trained pruning agents to determine the optimal sub-structure of the generator and discriminator. Then, we use the original loss functions for Pix2Pix [213] and CycleGAN [214] to fine-tune the models with Adam optimizer [102].

[2] https://github.com/facebookresearch/swav

Thus, most of the hyperparameters are similar to section S5.3.1 except a few changes. We use knowledge distillation [45, 236] between the original generator and the pruned one. However, as mentioned in Sec. 5.4.3, we do not use online distillation in our model, which saves computational costs. We use the Perceptual loss [271] for distillation that consists of two components: 1) Content loss that is the squared norm of the difference between two feature maps of the original and pruned models. 2) Texture loss that is the Frobenius norm of the difference between their Gram matrices. These two losses are applied on the same layers as GCC [236]. We represent their coefficients in our loss function with $\lambda_{content}$ and $\lambda_{texture}$ respectively. Tab. 5.6 shows our hyperparameter setting for fine-tuning the models in each experiment.

Table 5.6: Hyperparameter settings for Fine-tuning.

Model	Dataset	GAN Loss	Batch Size	Fine-tuning Epochs	Optimization Params		$\lambda_{content}$	$\lambda_{texture}$
					LR	Weight Decay		
Pix2Pix	Cityscapes	hinge	1	50	0.0002	0	50	10000
Pix2Pix	Edges2Shoes	hinge	4	50	0.0002	0	50	10000
CycleGAN	Horse2Zebra	Least Squares	1	50	0.0002	0	0.01	0
CycleGAN	Summer2Winter	Least Squares	1	50	0.0002	0	0.01	0

Chapter 6: Mixture of Efficient Diffusion Experts Through Automatic Interval and Sub-Network Selection

6.1 Introduction

Diffusion Probabilistic Models (DPMs) [11, 272, 273] have become the de facto models for generative modeling applications like image synthesis [11, 274], image editing [275, 276], super-resolution [277, 278], and video generation [279]. They train a denoising model that learns to generate samples from an input noise in an iterative denoising scheme. DPMs have achieved better mode coverage and training stability [274] than GANs [280] and show higher sample quality than VAEs [281]. Yet, the main drawback of DPMs is their slow and computationally intensive sampling process, making their cloud deployment costly and hindering usage on resource-constrained edge devices.

Two important factors contribute to slow sampling in DPMs: the models use 1) a large number of denoising steps and 2) a large number of parameters in each denoising step. Methods to speed up DPMs have primarily focused on reducing the sampling steps, using techniques like faster solvers [282, 283, 284, 285], better noise schedules [286, 287], and distillation [288, 289, 290]. In an orthogonal direction, a group of methods address the second factor and develop more efficient architectures for DPMs. Latent Diffusion Mod-

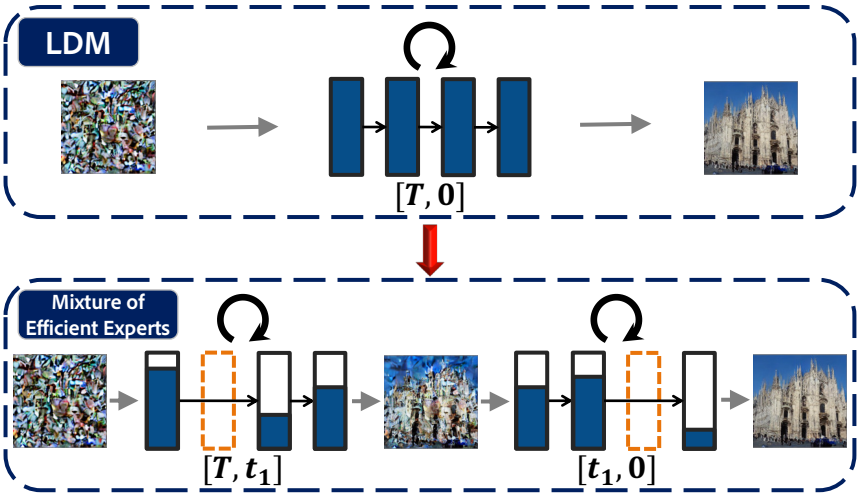


Figure 6.1: **Overview of DiffPruning.** We prune a pre-trained LDM model [1] (top) into a mixture of efficient experts (bottom). Each expert handles an interval, which allows their architectures to be separately specialized by removing layers or channels.

els (LDMs) [1] perform the diffusion process in a latent space with lower dimensions than pixel space, thereby significantly speeding up the sampling process while retaining a competitive performance. Accordingly, LDMs have been deployed in modern generative models like DALL-E 3 [17] and Stable Diffusion [1]. Thus, compressing LDMs is of significant interest. As LDMs do not have redundancies of the pixel-space DPMs by design, pruning them is much more challenging than pruning pixel-space DPMs.

Recently, several works [10, 291, 292] have explored architectural efficiency for LDMs. They divide the denoising path of an LDM into several intervals and use a distinct model for each one. These methods [10, 291, 292] are mainly inspired by studies [7, 293] showing different timesteps have distinct roles in the denoising process, and employing a single denoising model for all timesteps is sub-optimal [7, 294]. Thus, the key design choices here are the clustering scheme of the denoising timesteps and the method for allocating the resource budget between the selected clusters. MEME [291] uses uniform clustering, and TMDA [292] clusters the denoising timesteps by their loss values' similarities. Both MEME [291] and

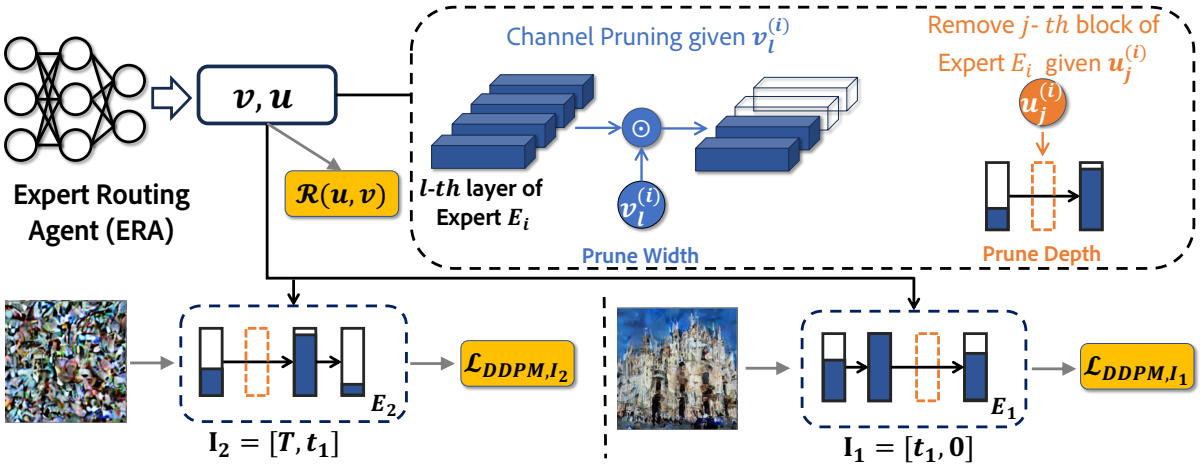


Figure 6.2: **Our Pruning Scheme:** We train our Expert Routing Agent (ERA) to prune the experts into a mixture of efficient experts (Sec. S6.2.3). The ERA predicts the architecture vectors (v, u) to prune experts' width and depth. Then, we calculate the denoising objectives of selected sub-networks of experts, $\mathcal{L}_{\text{DDPM},\mathcal{I}_i}$, as well as our Resource regularization term, $\mathcal{R}$, that encourages the ERA to provide a mixture of efficient experts with a desired compute budget (MACs). We train ERA's parameters to minimize the objective functions. Thus, it learns to automatically allocate the compute budget (MACs) between experts in an end-to-end manner.

TDMA [292] manually design a distinct U-Net model [295] for each cluster, thereby heuristically allocating the resource budget between the denoising intervals. However, by doing so, these methods need to re-design intervals' models for a new distinct budget, which is a complex, time-consuming, and labor-intensive task. OMS-DPM [10] avoids manual designing intervals' models as it trains a model zoo with different sizes and searches for an optimal mixture of denoising models, given a desired computational budget. Still, training a model zoo of various LDMs is extremely costly, even for medium-sized datasets, making OMS-DPM expensive to deploy in practice.

In this chapter, we propose a novel approach to make LDMs more efficient by pruning a large pretrained LDM into a mixture of efficient experts (Fig. 6.1) in four steps. First, we find an optimal division of time intervals by studying how aligned pairs of denoising

steps are to each other in a pretrained LDM. Interestingly, while different datasets all show natural clustering, the exact time intervals differ slightly between them. Thus, we adapt our clustering depending on the behavior of the dataset rather than using a static approach across datasets as in previous work [291]. Second, we fine-tune the pretrained model with elastic depth and width on each interval so that the sub-networks of the resulting model have a strong performance on that interval. We denote the elastically fine-tuned models as *experts* for the intervals. Our elastic fine-tuning provides an 'implicit' model zoo within each expert for its corresponding interval with fewer training iterations than training multiple models from scratch like OMS-DPM [10]. Third, we develop an Expert Routing Agent (ERA) that learns to select proper network configurations for the experts simultaneously, guided by the sub-networks' denoising objectives and allocated compute resource (*e.g.,* MACs). As we train our ERA in an end-to-end manner, it can automatically allocate computing resources between the experts without the need for complex heuristics [291, 292]. We summarize our contributions as follows:

- We introduce a method for pruning LDMs into a mixture of efficient experts.
- We propose to cluster denoising timesteps of a pretrained LDM into several intervals based on their pairwise alignment scores, showing that the optimal clustering intervals are distinct for different datasets. We employ a specialized efficient model for each interval.
- We fine-tune the pretrained LDM on selected intervals with elastic dimensions so that resulting expert models will have strong sub-networks to choose from. Thus, we can readily prune the experts for different computational budgets, and the pruned

experts can properly recover their performance without long fine-tuning iterations.

- We develop a new pruning scheme in which our expert routing agent learns to select optimal layouts of the experts together in an end-to-end manner, thereby allocating the compute budget between experts automatically.

The contents of this chapter are based on our work [296] published in ECCV 2024.

6.2 Related Work

Mixture of Experts (MoE) Diffusion Models: MoE methods cluster denoising time-steps of DPMs into intervals and train a separate *expert* model for each. eDiff-I [7] supports developing MoE for DPMs by showing that different denoising time-steps have distinct roles. Yet, how to cluster time-steps is non-trivial. eDiff-I employs a complex tree-based-branching scheme to divide the denoising path into two intervals sequentially and initializes a child model by its parent. ERNIE-ViLG [8] and MEME [291] uniformly cluster the denoising time-steps. Yet, these heuristic schemes do not necessarily transfer to other tasks. Different from these methods, we propose to cluster the denoising time-steps by measuring the alignment between their training objectives. We emphasize that although a recent work [294] has explored the time-steps' alignment scores *in the course of training*, our work is the first one to leverage them to cluster the time-steps for MoE DPMs.

Efficient DPMs. The majority of ideas for improving DPMs' efficiency reduce their denoising steps by faster samplers [285, 297, 298], distillation [288, 289, 290], better noise schedules [283, 286, 287, 299, 300, 301], learning denoising timesteps to use [302, 303], and caching [304]. We explore an orthogonal direction, compressing the architecture of

DPMs. LSGM [305] and LDM [1] perform the diffusion process in a lower dimensional latent space of an encoder-decoder pair [301, 306], thereby enjoying significantly faster sampling than pixel-space DPMs.

A few ideas have recently addressed compressing DPMs' architectures having two main categories. **Single-model** methods develop a single efficient model for all denoising timesteps. Structural Pruning (SP) [9] approximates weights' importance using the Taylor expansion and removes structures with low scores. Yet, SP's performance has been mainly verified on pixel space DPMs, and its pruned models on datasets like LSUN-Church [307] still have more than 6× MACs than the full-size LDM [1]. MobileDiffusion [308] introduces heuristics to enhance DPMs' efficiency and develops two efficient architectures. Nevertheless, it is highly non-trivial how to generalize the heuristics for different compute budgets. Spectral Diffusion (SD) [293] introduces a wavelet gating operator and performs frequency domain distillation from a teacher model into a small LDM. However, the main weakness of single-model methods is that they use the same model for all denoising steps, which is shown to be sub-optimal [7, 294]. **Mixture of expert** methods employ a separate model for different stages of the denoising process. OMS-DPM [10] trains a model zoo with various sizes and searches for a proper model schedule given a desired compute budget. Yet, gathering a model zoo is very costly on large-scale datasets, making OMS-DPM impractical for them. T-Stich [309] stitches several models with different sizes, each performing a part of the denoising process. But, similar to OMS-DPM [10], it requires several pre-trained models of various sizes, making it costly for practical scenarios. MEME [291] and TMDA [292] cluster denoising timesteps and design a distinct expert for each. However, they need to manually allocate the compute budget between experts and re-design the ex-

perts for a new budget, which makes them cumbersome in practice. We propose to prune an LDM into a mixture of efficient experts. We cluster denoising timesteps into intervals using their alignment scores. Then, we fine-tune the pre-trained model with elastic dimensions on each interval to obtain our experts. Thus, our method gathers an *implicit* model zoo *within* each expert with much lower training iterations than OMS-DPM. Finally, we prune all experts simultaneously using our expert routing agent to obtain our mixture of efficient experts. By doing so, in contrast with MEME [291] and TMDA [292], our method automatically allocates the compute resource (*e.g.,* MACs) between experts.

Model pruning and architecture search. Our work is also related to model pruning [169, 238, 310, 311, 312, 313, 314] and Neural Architecture Search (NAS) [119, 315, 316, 317, 318, 319] methods that prune the pretrained models and design novel architectures given a set of computational constraints. These ideas mainly focus on developing new architectures for image classification tasks, while we aim to design a novel pruning method for latent diffusion models [1]. We refer to recent surveys [320, 321, 322] for a detailed reviewing of pruning and NAS methods.

6.3 Method

We introduce a framework to prune an LDM [1] model into a mixture of efficient experts in four steps. First, we cluster denoising timesteps of the model into several intervals based on their objectives' alignment scores. Second, we fine-tune the pre-trained model on the selected intervals with elastic dimensions to obtain our interval experts. Third, we prune the experts together using our expert routing agent in an end-to-end manner

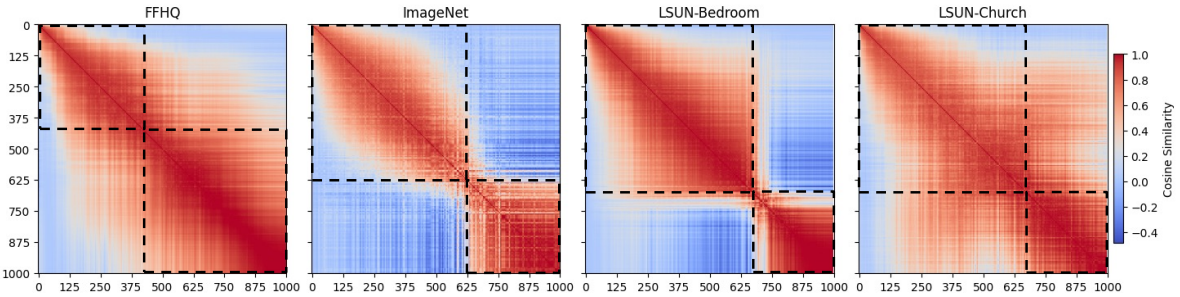


Figure 6.3: **Our Interval Selection Scheme:** We calculate gradients of denoising timesteps' objectives *w.r.t* the pre-trained LDM's parameters and take the cosine similarity value of two timesteps' gradients as their alignment score. The dashed lines show our selected cluster intervals for the experts. One can observe the optimal cluster assignments are different for distinct datasets, and employing a deterministic clustering strategy [7] like uniform clustering [8] for all datasets is sub-optimal.

(Fig. 6.2). Finally, we fine-tune the pruned experts to obtain our mixture of efficient experts.

6.3.1 Background

Given a random variable $\mathbf{x_0} \sim \mathcal{P}$, the goal of DPMs [11, 272] is to model the underlying distribution $\mathcal{P}$ using a training set $\mathcal{D} = \{x_0\}$ of samples. To do so, first, DPMs define a forward process parameterized by t in which they gradually perturb each sample x_0 with Gaussian noise with the variance schedule of β_t:

$$q(x_t|x_{t-1}) = \mathcal{N}(x_t; \sqrt{1-\beta_t}x_{t-1}, \beta_t I) \tag{6.1}$$

where $t \in [1, T]$. Thus, $q(x_t|x_0)$ has a Gaussian form:

$$q(x_t|x_0) = \mathcal{N}(x_t; \sqrt{\bar{\alpha}_t}x_0, (1-\bar{\alpha}_t)I) \tag{6.2}$$

where $\alpha_t = 1 - \beta_t$ and $\bar{\alpha}_t = \prod_{i=1}^{t} \alpha_i$. The noise schedule β_t is usually selected [11] such that $q(x_T) \to \mathcal{N}(0, I)$. Assuming β_t is small, DPMs approximate the denoising distribu-

tion $q(x_{t-1}|x_t)$ by a parameterized Gaussian distribution $p_\theta(x_{t-1}|x_t) = \mathcal{N}(x_{t-1}; \frac{1}{\sqrt{\alpha_t}}(x_t - \frac{\beta_t}{\sqrt{1-\bar{\alpha}_t}}\epsilon_\theta(x_t, t)), \sigma_t^2 I)$, and σ_t^2 is often set to β_t. DPMs implement $\epsilon_\theta(.)$ with a neural network called the denoising model and train it with the variational evidence lower bound (ELBO) objective [11]:

$$\begin{aligned} \mathcal{L}_{\text{DDPM}} &= \mathbb{E}_{t\sim[1,T]}\mathcal{L}_t \\ &= \mathbb{E}_{t\sim[1,T],\epsilon\sim\mathcal{N}(0,I),x_t\sim q(x_t|x_0)}||\epsilon_\theta(x_t, t) - \epsilon||^2 \end{aligned} \tag{6.3}$$

DPMs generate a new sample by sampling an initial noise from $x_T \sim p(x_T) = \mathcal{N}(0, I)$ and iteratively denoising it using the denoising model by sampling from $p_\theta(x_{t-1}|x_t)$. Thus, the sampling process requires T sequential forward passes to the large denoising model, making it a slow and costly process.

6.3.2 Notations

Fig. 6.4 shows the U-Net [295] architecture used in LDM [1] models. The encoder and decoder branches have several *stages* (each row in Fig. 6.4). Each stage has one or several *layers*. We represent the layers' functions and feature maps with $f_l(.)$ and $\mathcal{F}_l$, respectively, where $l \in [1, L]$, and L is the total number of layers. Each layer consists of one or several *blocks*. For instance, the green-colored *layers* in Fig. 6.4 are in the third *stage* of its encoder and decoder and consist of a Residual *block* [323] and an Attention *block* [324].

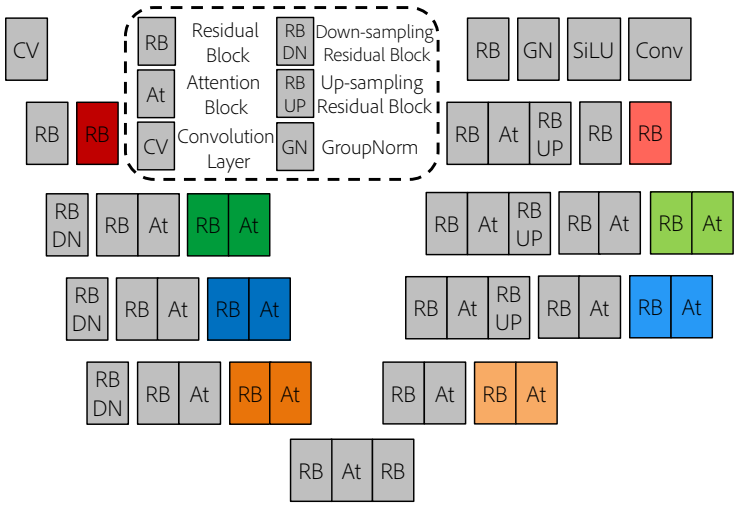


Figure 6.4: U-Net architecture of the LDM [1]. We randomly drop/preserve each colored layer in our elastic depth fine-tuning.

6.3.3 Clustering Denoising Timesteps into Intervals

We propose to cluster denoising timesteps $\mathcal{T} = [1, T]$ of an LDM into N intervals $\{\mathcal{I}_1, \mathcal{I}_2, \cdots, \mathcal{I}_N\}$ such that $\mathcal{T} = \mathcal{I}_1 \cup \mathcal{I}_2 \cup \cdots \cup \mathcal{I}_N$. The intuition is that different intervals have separate roles [7]. For example, it has been empirically shown [325] that an LDM first generates the layout of an image in high-noise timesteps and then fills in the details in low-noise ones. Thus, using the same denoising model for all timesteps is sub-optimal. We employ alignment scores of training objectives $\mathcal{L}_t$ (Eq. 6.3) for denoising timesteps of a pre-trained LDM to cluster them. We estimate the gradient of each $\mathcal{L}_t$ *w.r.t* the denoising model's parameters (θ) using a random batch of samples in the training data and take the cosine similarity between the gradients of $\mathcal{L}_t$ and $\mathcal{L}_s$ as the alignment score of timesteps t and s.

We visualize pairwise alignment scores of denoising time-steps for pre-trained LDMs [1] of different datasets in Fig. 6.3. We select two distinct clusters ($N = 2$) for all datasets

(shown by dashed lines in Fig. 6.3) in our experiments. We choose the cut-off point between the clusters to be the one that maximizes the weighted mean of the average scores of the clusters, and we refer to the supplementary S6.2.1 for the formulation. We do not use more than two experts in our experiments for computational efficiency. However, our formulation can find cut-off points for more than two clusters, as we elaborate in the supplementary S6.2.1. One can observe that intra-cluster alignment scores are high, and inter-cluster scores are small and even negative for LSUN-Bedroom [307], ImageNet [326], and FFHQ [327]. Accordingly, further training the denoising model on one of the clusters degrades its performance on the other. This observation supports our decision to employ a specialized model for each interval, that using a single model for all intervals is sub-optimal. Further, the optimal cluster assignment is different for distinct datasets. Thus, our clustering method is more robust than deterministic ones [7, 8]. In summary, we select clusters $(\mathcal{I}_1, \mathcal{I}_2)$ to be $([0, 700], [701, 1000])$ for LSUN-Church as well as LSUN-Bedroom, $([0, 400], [401, 1000])$ for FFHQ, and $([0, 625], [626, 1000])$ for ImageNet.

6.3.4 Fine-tuning with Elastic Dimensions

We fine-tune the pre-trained LDM with elastic dimensions (depth and width) on each denoising interval $\mathcal{I}_i$ ($i \in [1, N]$) after clustering the denoising timesteps. We call the resulting models *experts* and denote them with E_i, corresponding to $\mathcal{I}_i$. Our main inspiration is that by doing so, each sub-network of an expert E_i has a decent performance on the denoising interval $\mathcal{I}_i$, which brings in several key benefits: First, the loss value of each sub-network of E_i will be a proper proxy for its actual performance after fine-tuning on

$\mathcal{I}_i$. Second, the pruned experts will be able to recover their performance promptly during fine-tuning without requiring long fine-tuning iterations. Finally, our elastic fine-tuning provides a model zoo *within* the expert E_i for the denoising interval $\mathcal{I}_i$ without extreme computational and memory expensive training of several architectures from scratch like in OMS-DPM [10]. We first fine-tune the pre-trained model with elastic depth on each interval. Then, we fine-tune the resulting model with elastic width. We do not perform elastic depth and width training together to prevent instabilities.

Fine-tuning with elastic depth. We randomly drop the last layer in each stage of the U-Net's encoder and decoder (colored blocks in Fig. 6.4) for our elastic depth training. Formally, for each training batch, we randomly select to map the last depth layers $f_j^{(i)}$ in stages of the expert E_i independently with a probability p to the identity function:

$$\hat{f}_j^{(i)} = f_j^{(i)} \mathbf{1}_{\{s_j^{(i)}=0\}} + I \mathbf{1}_{\{s_j^{(i)}=1\}}, \quad s_j^{(i)} \sim \text{Bernoulli}(p) \tag{6.4}$$

We train selected sub-network's parameters with the interval denoising objective:

$$\mathcal{L}_{\text{DDPM},\mathcal{I}_i} = \mathbb{E}_{t\sim\mathcal{I}_i} \mathcal{L}_t \tag{6.5}$$

where $\mathcal{L}_t$ has the same formulation as Eq. 6.3.

Fine-tuning with elastic width. After fine-tuning the pre-trained model on each interval $\mathcal{I}_i$ with elastic depth, we fine-tune the resulting experts with elastic width. For each ResBlock in the U-Net, we sort the channels of its convolution layers based on an estimate of their importance (determined by their L_1 norm [328, 329]). Then, for each training batch, we randomly remove some ratio of the least important channels of each convolution

layer. Similarly, for the attention layers [324], we sort the attention heads based on the L_1 norm of their projection weights, and we randomly drop some of the least important heads during our elastic width fine-tuning. Finally, we update the selected sub-network's weights using $\mathcal{L}_{\text{DDPM},\mathcal{I}_i}$ (Eq. 6.5). We refer to supplementary S6.2.2 for more details.

6.3.5 Expert Routing Agent

We develop our Expert Routing Agent (ERA) to prune the elastically fine-tuned experts E_i ($i \in [1, N]$) into a mixture of efficient experts. We denote our ERA as a function $h_{\text{ERA}}(.;\beta)$ parameterized by β, predicting architecture vectors (u, v):

$$u, v = h_{\text{ERA}}(z;\beta) \tag{6.6}$$

z is a constant, randomly initialized input. Vectors $u = [u^{(i)}]_{i=1}^{N}$ determine pruning depth layers $f_j^{(i)}$ (Eq. 6.4). Similarly, vectors $v = [v^{(i)}]_{i=1}^{N}$ determine widths of blocks of N experts. Together, (u, v) select sub-networks e_i from experts E_i.

Given a total constraint on the computation budget (*e.g.*, MACs, latency, etc.), denoted as T_d, we optimize the ERA model's parameters β to predict architecture vectors (u, v) for an efficient and high-performing set of experts' architectures. Next, we describe how we parameterize and apply (u, v). We show the formulation to determine the compute budget of selected architectures and the final optimization procedure for the ERA in Eq. 6.15.

6.3.5.1 Pruning Width

Although one can prune widths of blocks in an expert E_i using a binary vector $v^{(i)}$, keeping the j^{th} channel when $v_j^{(i)}$ is 1 and vice versa, such an operation is not differentiable, making the optimization of parameters β of the ERA challenging. Thus, we introduce soft vectors $\mathbf{v}^{(i)}$, relax them to have continuous values, and use them for width pruning. We calculate them as follows:

$$\mathbf{v}^{(i)} = \text{sigmoid}(\frac{v^{(i)} + n}{\tau}) \tag{6.7}$$

$n \sim \text{Gumbel}(0, 1)$ is a noise from the Gumbel distribution [330]. Parameter τ is the temperature that when set appropriately, brings elements of $\mathbf{v}^{(i)}$ close to 0 or 1. The calculation from $v^{(i)}$ to $\mathbf{v}^{(i)}$ is called the Gumbel-Sigmoid trick [331, 332]. It is a differentiable estimation of sampling from a Bernoulli distribution with the Bernoulli parameter of $\texttt{sigmoid}(v^{(i)})$. We apply vectors $\mathbf{v}^{(i)} = [\mathbf{v}_l^{(i)}]_{l=1}^{L}$ to prune the width of blocks in all of the layers $f_l^{(i)}$ for the expert E_i:

$$\hat{f}_l^{(i)} = f_l^{(i)}(.; \mathbf{v}_l^{(i)}) \tag{6.8}$$

Here, we apply (multiply) the width vector $\mathbf{v}_l^{(i)}$ to feature maps of the first convolution layer in the ResBlocks and inputs of the attention operation in the attention blocks in the layer $f_l^{(i)}$. The granularity of our width pruning is similar to our elastic width fine-tuning, *i.e.,* we prune channels of convolution layers of ResBlocks and heads of the attention layers.

6.3.5.2 Pruning depth

Similarly, we employ relaxed continuous vectors $\mathbf{u}^{(i)} = [\mathbf{u}_j^{(i)}]$ for pruning the depth layers $f_j^{(i)}$ (Eq. 6.4) of the expert E_i:

$$\mathbf{u}^{(i)} = \text{sigmoid}(\frac{u^{(i)} + n}{\tau}) \tag{6.9}$$

As there are skip connections in the U-Net, we apply the depth architecture vectors for the encoder and decoder branches differently.

Encoder depth pruning. We use the following formulation to apply the vector $\mathbf{u}^{(i)}$ for pruning depth layers in the encoder of the expert E_i:

$$\widehat{\mathcal{F}}_j^{(i)} = \mathbf{u}_j^{(i)} f_j^{(i)}(\mathcal{F}_{j-1}^{(i)}) + (1 - \mathbf{u}_j^{(i)})\mathcal{F}_{j-1}^{(i)} \tag{6.10}$$

In other words, we interpolate between the feature map of the previous layer, $\mathcal{F}_{j-1}^{(i)}$, and the result of applying the current layer to it, $f_j^{(i)}(\mathcal{F}_{j-1}^{(i)})$. The $\mathbf{u}_j^{(i)}$ values close to 1 simulate preserving the layer, and 0 simulate removing the layer.

Decoder depth pruning. The input for the layer $f_j^{(i)}$ in the decoder of the expert E_i is the concatenation of feature maps $\mathcal{F}_{j-1}^{(i)}$ of its previous layer and the skip connection feature maps $\mathcal{F}_{j,skip}^{(i)}$. Thus, we apply the vector $\mathbf{u}_j^{(i)}$ to it as:

$$\widehat{\mathcal{F}}_j^{(i)} = \mathbf{u}_j^{(i)} f_j^{(i)}(\mathcal{F}_{j-1}^{(i)} \parallel \mathcal{F}_{j,skip}^{(i)}) + (1 - \mathbf{u}_j^{(i)})\mathcal{F}_{j-1}^{(i)} \tag{6.11}$$

where $\parallel$ denotes concatenation. Similar to Eq. 6.10, we interpolate between applying or

removing the layer in Eq. 6.11.

6.3.6 Pruning the Mixture of Experts

We train our Expert Routing Agent to select competent sub-networks of elastically trained experts given a desired total compute budget. We measure the compute budget of our models with MACs, following [9, 291]. Given an architecture width vector $\mathbf{v}_l^{(i)}$, the MACs of the layer $f_l^{(i)}(.)$ after applying $\mathbf{v}_l^{(i)}$ will be:

$$\widehat{T}_l^{(i)} = \mathbf{1}^T \times \lfloor \mathbf{v}_l^{(i)} \rceil \times T_l^{(i)} \tag{6.12}$$

where $\mathbf{1}$ denotes a vector of all ones. $\lfloor \cdot \rceil$ is the function that rounds to the nearest integer, and $T_l^{(i)}$ is the MACs of the layer $f_l^{(i)}(.)$. Similarly, the MACs for the layers $f_j^{(i)}(.)$ that we use for depth pruning (Eq. 6.4) after applying $\mathbf{u}_j^{(i)}$ will be:

$$\widehat{T}_j^{(i)} = \lfloor \mathbf{u}_j^{(i)} \rceil \times \mathbf{1}^T \times \lfloor \mathbf{v}_j^{(i)} \rceil \times T_j^{(i)} \tag{6.13}$$

After applying architecture vectors of each expert E_i, we calculate the total MACs of our mixture of experts as:

$$\widehat{T}(u, v) = \sum_{i=1}^{N} \frac{|\mathcal{I}_i|}{\sum_{k=1}^{N} |\mathcal{I}_k|} \widehat{T}^{(i)}(u^{(i)}, v^{(i)}) \tag{6.14}$$

where $\widehat{T}^{(i)}(u^{(i)}, v^{(i)})$ is the MACs of the expert E_i after applying its architecture width and depth vectors. In Eq. 6.14, we assume that the denoising schedule is *linear* such that the number of denoising steps that the expert E_i will contribute to the denoising process

is proportional to the size of its interval $|\mathcal{I}_i|$. One can alter Eq. 6.14 for other denoising schedules like quadratic [286], but we focus on the linear schedule as it has been widely adopted in the literature [1, 9, 286, 291].

Given a desired MACs budget T_d, we train our ERA with the following objective to encourage it to select sub-networks of experts such that each of them has a high performance and their mixture has a total MACs close to T_d:

$$\min_{\beta} \mathcal{J}(T_d) = [\frac{1}{N}\sum_{i=1}^{N} \mathcal{L}_{\text{DDPM},\mathcal{I}_i}(E_i(u^{(i)}, v^{(i)}))] + \mathcal{R}(\widehat{T}(u, v), T_d) \tag{6.15}$$

$\mathcal{L}_{\text{DDPM},\mathcal{I}_i}(E_i(u^{(i)}, v^{(i)}))$ is the interval denoising objective (Eq. 6.5) of the sub-network of E_i chosen by $(u^{(i)}, v^{(i)})$. $\widehat{T}(u, v)$ is the total MACs of the mixture of experts (Eq. 6.14) determined by the architecture vectors (u, v) that are functions of the ERA's parameters β. $\mathcal{R}(\cdot)$ is the MACs regularization term that we implement it as $\mathcal{R}(x, y) = \log(\max(x, y)/\min(x, y))$. Now, as the round function $\lfloor\cdot\rceil$ used in Eqs. (6.12, 6.13) is not differentiable, we use the Straight Through Estimator (STE) [333] to calculate the gradients of $\mathcal{R}$ *w.r.t* the parameters β of our ERA. We implement our ERA model with a GRU [334] layer followed by dense layers. We found in our experiments that a lightweight ($\sim 0.5M$ parameters) ERA model suffices to obtain a performant mixture of efficient experts. We show our pruning scheme in Fig. 6.2 and refer to supplementary S6.2.3 for more details of our ERA's architecture as well as our pruning algorithm.

6.3.6.1 Fine-tuning pruned models.

After our pruning stage, we use architecture vectors predicted by the ERA to prune experts. Then, we fine-tune the experts with the same settings as the original LDM model [1].

6.4 Experiments

We experiment on the LSUN-Church [307], LSUN-Bedroom [307], FFHQ [327], and class-conditional ImageNet [326] to verify our method's effectiveness. We apply our method to the LDM [1] that implements their denoising model with a U-Net [295] architecture. For all datasets, we prune the LDM model with three MACs budgets of 70%, 50%, and 30%. In all experiments, we mainly follow the same hyper-parameter settings as LDM [1], and we refer to supplementary S6.3 for more details of our experimental setup. We denote our method as DiffPruning in the rest of the experiments section. We mainly compare our method with a few recent baselines on architectural efficiency of pixel-space [9] and latent space [10, 291] DPMs. We do not benchmark with Spectral Diffusion (SD) [293] because it uses a package[1] that does not count MACs of the attention operation `QKVAttention` implemented in the LDM repository[2]. Hence, the MACs of models reported by SD [293] are not accurate. For instance, SD reports the LDM [1] for LSUN-Church has 18.7G MACs, but it actually has 20.96G (Tab. 6.1c).

[1] https://github.com/sovrasov/flops-counter.pytorch
[2] https://github.com/CompVis/latent-diffusion

6.4.1 Comparison Results

We summarize comparison results in Tab. 6.1 and refer to supplementary S6.1 for FID *vs.* MACs as well as FID *vs.* Throughput curves of our method and baselines.

LSUN-Bedroom. Tab. 6.1a presents the results on LSUN-Bedroom. First, we can observe that the LDM [1] can achieve better sample quality (lower FID) while having significantly lower MACs (higher sampling speed) than the pixel-space DPM, DDPM [11]. Although SP [9] prunes more than 44% MACs of the DDPM [11], its pruned model still has 36% more MACs than LDM while drastically degrading the sample quality to 18.6 FID. These results demonstrate that pixel-space DPMs have much more redundancies than LDMs, and pruning LDMs is significantly more challenging. Second, our pruned models can achieve higher throughput speed-up ratio than their pruning ratio while having competitive FID scores. DiffPruning 70%/50%/30% models reduce MACs by 30%/50%/70%, but can secure 43%/87%/135% sampling speed up compared to LDM [1]. Notably, DiffPruning 70% (50%) has 5.90 (6.73) FID score which is fairly close to the original LDM (4.39) while substantially better than the pruned model by SP [9] (18.6). Finally, although not directly comparable, our pruned models require less training iterations than SP, and we refer to supplementary S6.3.2 for details.

LSUN-Church. The architecture MACs of the LDM [1] for LSUN-Church is smaller than LDM models for other datasets as its encoder reduces the spatial dimension by 8 *vs.* 4 for others. Thus, pruning the LDM for LSUN-Church is more challenging than other ones, and Tab. 6.1c summarizes the results. First, we can observe that DiffPruning 70% achieves drastically better FID than the DDPM model pruned by SP [9] while having almost

Table 6.1: Comparison results of DiffPruning *vs.* baselines. Throughput values are calculated using an NVIDIA A100 GPU. †: the values are average of our two efficient experts. ∗: calculated by sampling from provided checkpoints. ‡: speed-ups relative to the LDM model. The shadowed values are inaccurate, and we refer to supplementary S6.3.3 for a detailed discussion.

LSUN-Bedroom (256 × 256)

Model	Complexity		Performance	
	Params	MACs	Throughput (↑) (Sample/Sec)	FID (↓)
DDPM [11]	113.7M	248.7G	0.74	6.62
SP [9]	63.2M	138.8G	-	18.6
LDM [1]	274.06M	101.32G	2.01	4.39*
DiffPruning (70%)	162.06M†	70.84G	**3.11** (×1.55)‡	5.90
DiffPruning (50%)	100.87M†	50.69G	**3.75** (×1.87)‡	6.73
DiffPruning (30%)	48.43M†	31.11G	**4.73** (×2.35)‡	9.22

(a)

FFHQ (256 × 256)

Model	Complexity		Performance	
	Params	MACs	Throughput (↑) (Sample/Sec)	FID (↓)
LDM [1]	274.06M	101.32G	1.01	9.53*
DiffPruning (70%)	194.79M†	71.05G	**1.35** (×1.33)‡	9.80
DiffPruning (50%)	134.67M†	51.87G	**1.83** (×1.81)‡	9.90
DiffPruning (30%)	63.07M†	30.68G	**2.90** (×2.87)‡	10.66

(b)

LSUN-Church (256 × 256)

Model	Complexity			Performance	
	Sampler	Params	MACs	Throughput (↑) (Sample/Sec)	FID (↓)
LDM [1]	DDIM-100	294.7M	20.96G	5.19	5.21*
LDM [1]	DDIM-200	294.7M	20.96G	2.60	5.11*
DDPM [286]		113.7M	248.7G	0.74	10.58
SP [9]	DDIM-100	63.2M	138.8G	-	13.9
DiffPruning (70%)		188.09M†	14.64G	**5.73** (×1.11)‡	9.39
OMS-DPM [10]	Searched	-	-	2.56	11.10
DiffPruning (50%)	DDIM-200	112.6M†	10.48G	3.15	10.22
DiffPruning (50%)	DDIM-100	112.6M†	10.48G	**6.28** (×1.21)‡	10.89
OMS-DPM [10]	Searched	-	-	6.4	13.7
DiffPruning (30%)	DDIM-100	36.9M†	6.35G	**6.87** (×1.32)‡	11.39

(c)

Class-Conditional ImageNet (256 × 256)

Model	Complexity		Performance	
	Params	MACs	Throughput (↑) (Sample/Sec)	FID (↓)
ADM [274]	607.9M	1186.4G	0.07	4.59
LDM [1]	400.82M	108.78G	0.32	3.60
DiffPruning (70%)	250.79M†	76.24G	**0.43** (×1.34)‡	8.03
LDM 50% Scratch [9]	189.43M	52.71G	-	51.45
Taylor [9]	189.43M	52.71G	-	11.18
SP [9]	189.43M	52.71G	-	9.16
DiffPruning (50%)	161.06M†	54.32G	**0.56** (×1.75)‡	8.45
DiffPruning (30%)	79.82M†	32.71G	**0.92** (×2.87)‡	13.18

(d)

9.5× fewer MACs. Remarkably, DiffPruning 70% achieves better FID than the full DDPM model, illustrating that LDMs have better computation-performance frontier than pixel-space DPMs. Second, DiffPruning 50% and 30% models can achieve both higher throughput and better FID while requiring more than 7× less training iterations than OMS-DPM [10] (details in supplementary S6.3.2). DiffPruning 50% with the 100-step DDIM [286] sampler has 2.45× higher throughput (6.28 *vs.* 2.56) than OMS-DPM with a lower FID. Also, DiffPruning 50% with 200 steps DDIM sampler still has a higher throughput and better FID than OMS-DPM. Notably, DiffPruning 50% with 200 steps DDIM sampler can outperform the full DDPM model (10.22 *vs.* 10.58 FID) while having 4.25× faster sampling throughput. Finally, DiffPruning 30% has higher throughput (6.87 *vs.* 6.4 FID) while outperforming OMS-DPM's model by 2.31 FID. In summary, the comparison results with

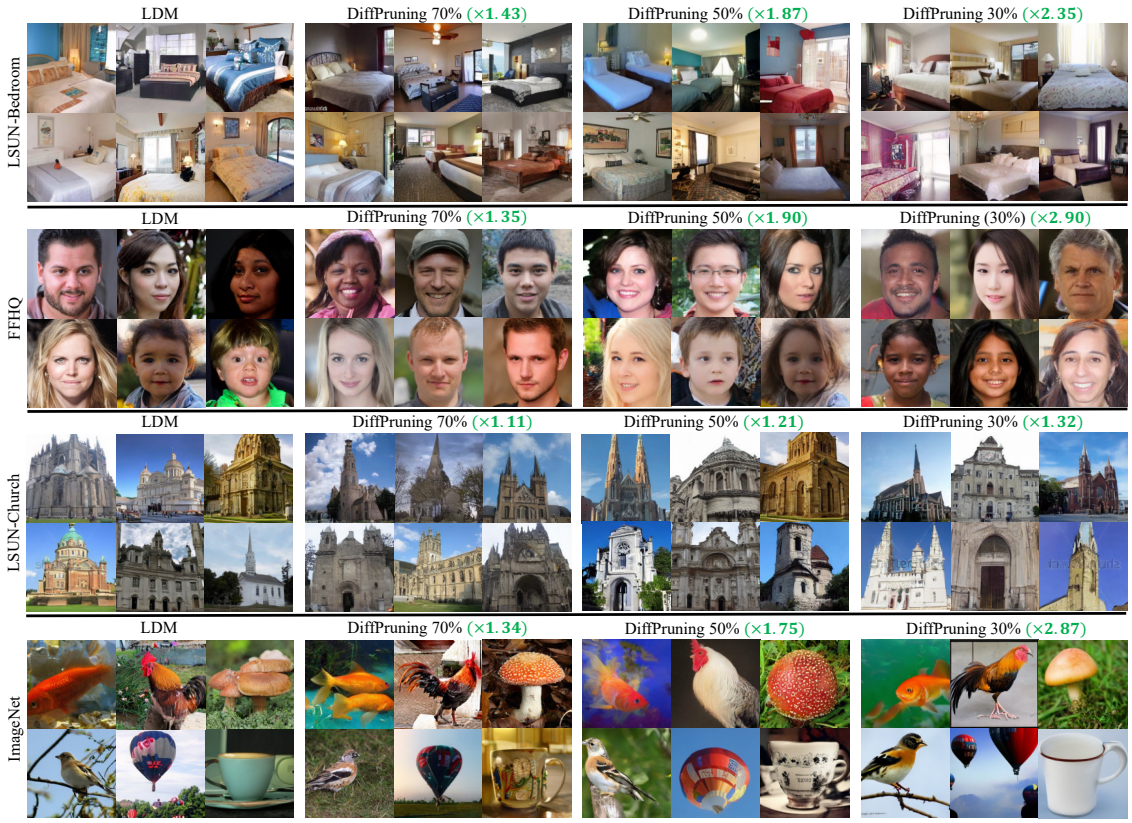


Figure 6.5: Samples from the LDM [1] model and our pruned mixture of experts for different MACs budgets. The green numbers show the relative sampling throughput speed-up of our pruned models compared to LDM on an NVIDIA A100 GPU.

OMS-DPM demonstrate the benefit of our elastic fine-tuning for our experts that enables our method to gather a model zoo without requiring training several models from scratch, thereby obtaining a higher-performing mixture of efficient experts with much lower training iterations than OMS-DPM.

FFHQ. Tab. 6.1b shows the results on FFHQ. DiffPruning 70%/50% models can achieve close FID scores to LDM [1] while enjoying 33%/81% throughput speed-ups. In the extreme case of 30% MACs budget, DiffPruning secures 2.87× speed-up while having a 1.13 worse FID score than LDM, which shows that it can successfully prune the LDM for small-scale datasets like FFHQ as well.

Class-Conditional ImageNet. Tab. 6.1d summarizes results for ImageNet. DiffPruning 50% model can achieve 0.71 better FID score than SP [9]. The reported MACs values by SP are not directly comparable to ours, and we elaborate on the reason in supplementary S6.3.3. In addition, DiffPruning 70%/30% obtain 1.34×/2.87× increase in throughput compared to LDM. Thus, DiffPruning can effectively prune conditional LDMs as well. In summary, our experimental results demonstrate that our method can effectively prune both unconditional and conditional LDM models for datasets with various scales. We provide samples generated by the original LDM and our pruned mixture of experts with different budgets in Fig. 6.5.

6.4.2 Ablation Study

We conduct an ablation study to explore the contribution of each component of our method to its final performance. We implement a Baseline that uses naive parameterizations $\mathbf{v} = \text{sigmoid}(-(\beta_v + n)/\tau)$ (Eq. 6.7) and $\mathbf{u} = \text{sigmoid}(-(\beta_u + n)/\tau)$ (Eq. 6.9) for pruning a single model. Then, we add the mixture of experts, our ERA model, elastic depth fine-tuning, and elastic width fine-tuning one at a time for pruning the model. Tab. 6.2 summarizes the results. First, we can observe that employing the mixture of experts improves both the sample quality FID score (which is aligned with the prior works [7, 8]) and the inference throughput of the pruned model. This result quantitatively justifies our design choice for clustering the denoising timesteps into intervals and using a specialized model for each of them. Employing our ERA model yields a faster model than the naive parameterization. The reason may be that the naive parameterization cannot properly model the

complex interactions between experts and between different layers within an expert. Noticeably, employing each component of our method improves *both dimensions* of sampling throughput and sample quality such that our method can obtain a 1.28× faster model (throughput 3.75 *vs.* 2.92) with 3.59 better FID than the naive Baseline. In summary, our ablation experiments verify our design choices for DiffPruning.

Table 6.2: Ablation results of our proposed method for pruning the LDM model [1] for LSUN-Bedroom to 50% MACs budget.

Model	Sampler	MACs	Throughput(↑) (Sample/Sec)	FID (↓)
Baseline	DDIM-100	50.69G	2.92	10.32
+ Mixture of Experts			3.05	9.65
+ Expert Routing Agent			3.25	8.53
+ Elastic Depth			3.61	8.03
+ Elastic Width (Ours)			3.75	6.73
LDM [1]		101.32G	2.01	4.39

6.5 Chapter Summary and Conclusions

We introduce a novel approach for pruning an LDM model into a mixture of efficient experts in which each expert performs the denoising task on an interval of the denoising path. We employ the model's denoising timesteps' alignment scores to cluster them into several intervals and empirically show that the optimal cluster assignments are different for distinct datasets. Thus, using static clustering schemes is sub-optimal. We propose to fine-tune the pre-trained LDM on each cluster interval with elastic dimensions to obtain our interval experts. By doing so, each expert contains an implicit model zoo within itself for its corresponding interval. Finally, we develop a new pruning scheme in which our Expert Routing Agent (ERA) learns to prune the elastically trained experts together in an end-to-end manner. Thus, our ERA automatically allocates the compute budget between experts.

Our experimental results validate our method's effectiveness, and our ablation studies show that our design choices improve both dimensions of the pruned model's throughput and its sample quality.

Supplementary Materials for Chapter 6

We elaborate on more details about our method, experimental setup, and experimental results in the supplementary materials. We follow the same notations introduced in Sec. 6.3.2.

S6.1 Experimental Results

We present the FID *vs.* MACs and FID *vs.* Throughput of our method and baselines in Fig. 6.6.

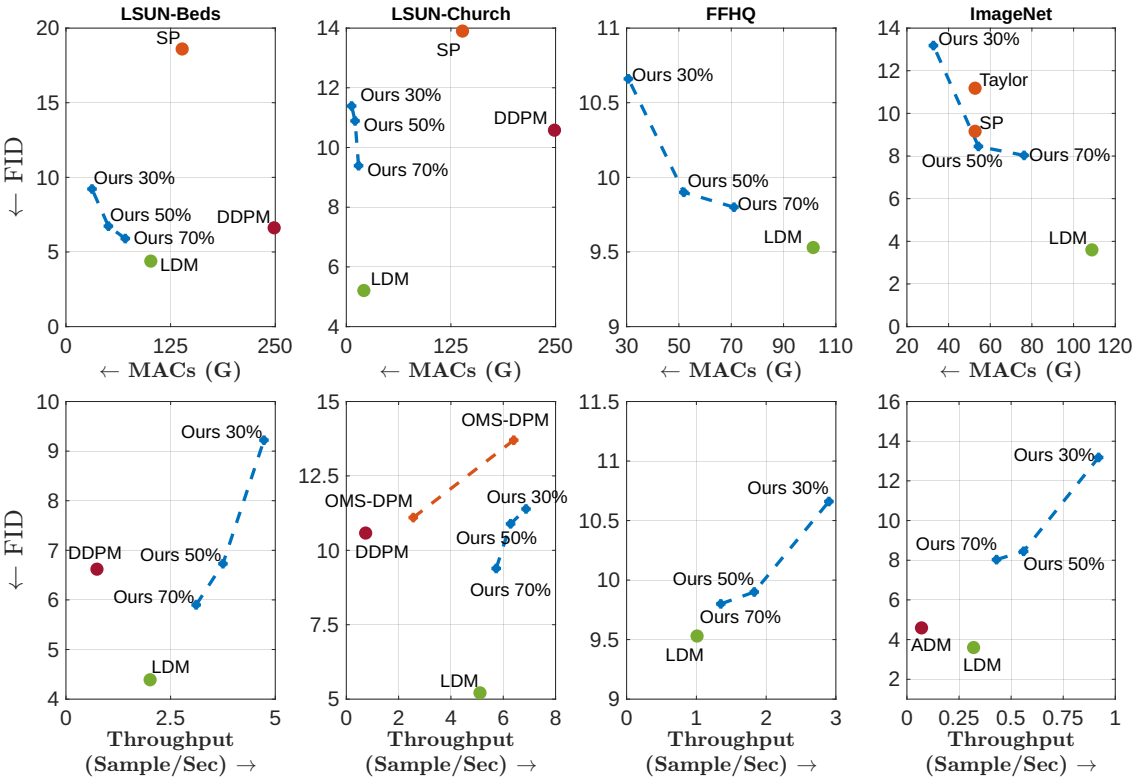


Figure 6.6: Comparison Results of our method *vs.* baselines, SP [9], OMS-DPM [10], DDPM [11], and LDM [1]. **First Row:** FID *vs.* MACs curves. **Second Row:** FID *vs.* Throughput curves. We calculate the Throughput values with an NVIDIA A100 GPU. Higher Throughput and Lower FID and MACs indicate a better performance.

S6.2 Details of Our Method

S6.2.1 Clustering Denoising Time-Steps into Intervals

In this section, we provide details of our method to cluster denoising time-steps of an LDM [1] into intervals. As mentioned in Sec. 6.3.3, we use two experts in our experiments for computational efficiency. Thus, we explain our approach for clustering the denoising path into two intervals, but one may extend it to a higher number of intervals as well.

Given denoising time-steps $\mathcal{T} = [1, T]$, we divide it into two intervals $\mathcal{I}_1 = [T, t_1]$ and $\mathcal{I}_2 = [t_1, 1]$. Now, the main question is how to determine the cut-off time-step t_1. We propose to leverage alignment scores of time-steps to find the optimal t_1. We take the cosine similarity between the gradients of training objectives $\mathcal{L}_t$ and $\mathcal{L}_s$ as the alignment score of the time-steps t and s and denote it with $a_{t,s}$. We propose to select the cut-off point that maximizes the weighted average of the mean of alignment scores in clusters:

$$\max_{t_1} \mathcal{J}(t_1) = \sum_{i=1}^{2} [w_i (\sum_{j \in \mathcal{I}_i} \sum_{k \in \mathcal{I}_i} \frac{a_{j,k}}{|\mathcal{I}_i|^2})] \tag{6.16}$$

$$w_i = \frac{|\mathcal{I}_i|}{T} \tag{6.17}$$

where $|\mathcal{I}_i|$ is the number of time-steps in $\mathcal{I}_i$ and T is the total number of denoising time-steps that is usually set to 1000 in practice [1, 11, 286]. The Obj 6.16 encourages to choose t_1 such that the average of alignment scores be high in each cluster while the weights w_i adjust the contribution of each cluster to the objective based on their size. Our weighting

scheme prevents degenerate solutions such as choosing a single time-step as a separate cluster. Figs (6.7-6.10) show the $\mathcal{J}(t_1)$ functions for different datasets. We choose the cut-off values 400, 625, 700, and 700 for the FFHQ, ImageNet, LSUN-Bedroom, and LSUN-Church models to maximize their $\mathcal{J}(t_1)$ values. These values result in the $(\mathcal{I}_1, \mathcal{I}_2)$ clusters that we chose in Sec. 6.3.3 and Fig. 6.3. We believe that one can readily extend our method to the cases with a higher number of experts by optimizing for the cut-off points that maximize similar objectives to $\mathcal{J}$. Specifically, if one decides to use C+1 experts (clusters), they should find C cut-off points $t_1 < t_2 < \cdots < t_C$ to optimize the following objective:

$$\max_{t_1, t_2, \cdots t_C} \mathcal{J} = \sum_{i=1}^{C+1} [w_i (\sum_{j \in \mathcal{I}_i} \sum_{k \in \mathcal{I}_i} \frac{a_{j,k}}{|\mathcal{I}_i|^2})] \tag{6.18}$$

with the same definitions as Eqs. (6.16, 6.17).

Finally, we note that we use 1024 random images to estimate the gradient of each time-step for the models for LSUN-Church [307], LSUN-Bedroom [307], and FFHQ [327]. We employ 16384 samples to do so on ImageNet [326].

S6.2.2 Fine-tuning with Elastic Width

We mentioned in Sec. 6.3.4 that we fine-tune the experts with elastic width after training them with elastic depth. Here, width means the channels of the convolution layers in the Residual Blocks [323] and heads of the attention layers [324] in the attention blocks of the U-Net. Before starting the elastic width fine-tuning, we sort the channels in the convolution layers in ResBlocks based on their importance, determined by their L_1 norm [328, 329]. Similarly, we sort the attention heads based on the L_1 norm of their

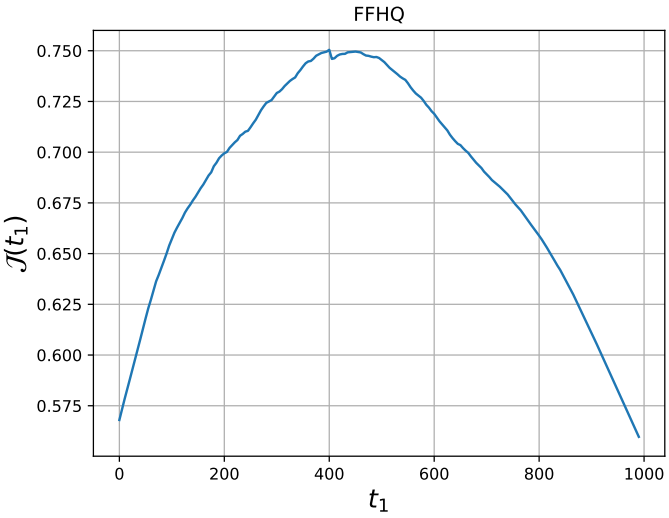


Figure 6.7: Weighted average $\mathcal{J}(t_1)$ (Eq. 6.16) of the mean of alignment scores in two clusters for the LDM trained on FFHQ.

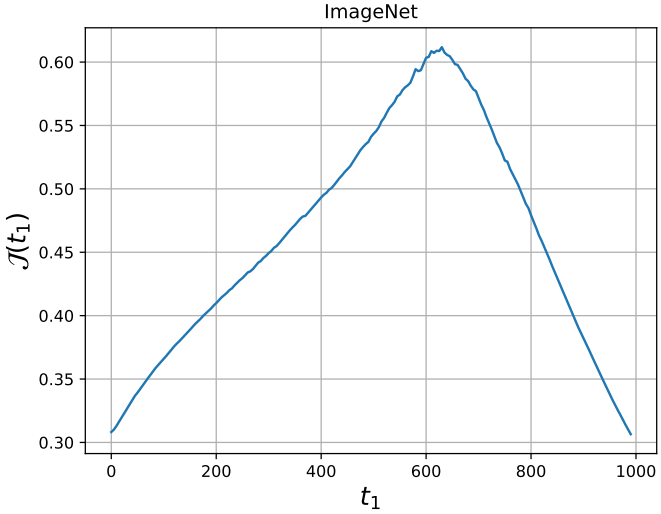


Figure 6.8: Weighted average $\mathcal{J}(t_1)$ (Eq. 6.16) of the mean of alignment scores in two clusters for the LDM trained on ImageNet.

projection weights. Then, in each batch of elastic width fine-tuning, We independently sample a random ratio r ($r \sim \mathcal{U}[0,\ 1)$) for each convolution layer with W channels (attention layer with W heads) and drop the $\lfloor Wr \rfloor$ least important channels (attention heads) of the layer. We illustrate our elastic width channel selection for a convolution layer with 4 channels in Fig. 6.11. The channels are sorted based on their L_1 norm (shown by their

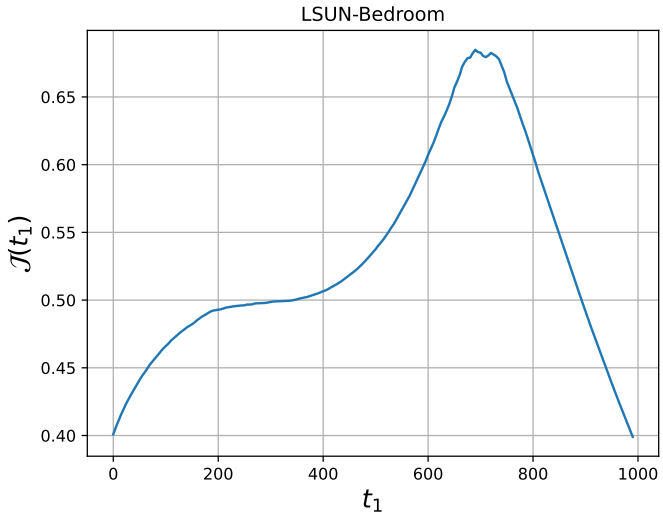


Figure 6.9: Weighted average $\mathcal{J}(t_1)$ (Eq. 6.16) of the mean of alignment scores in two clusters for the LDM trained on LSUN-Beds.

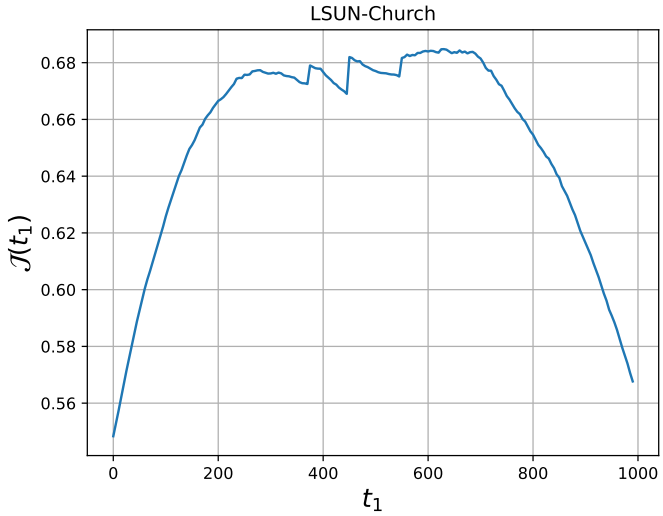


Figure 6.10: Weighted average $\mathcal{J}(t_1)$ (Eq. 6.16) of the mean of alignment scores in two clusters for the LDM trained on LSUN-Church.

color intensity). The values $o_{1:4}$ represent different possible channel dropping cases for our elastic width training of the convolution layers. We similarly drop a ratio of least important attention heads.

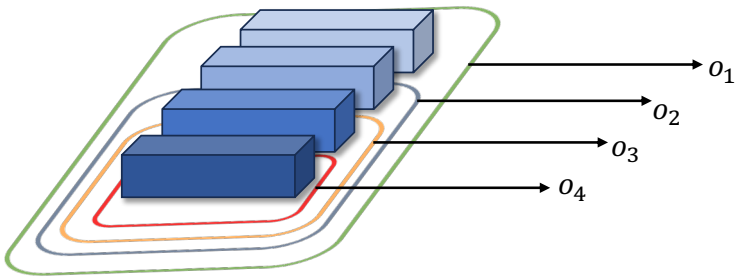


Figure 6.11: Illustration of our Elastic Width training. We sort the convolution channels (attention heads) based on their importance (L_1 norm) before starting elastic width training. We drop a random ratio of the least important channels (heads) for convolution layers (attention layers) for each batch of training. The values $o_{1:4}$ represent different possible dropping ratios for a convolution layer with 4 channels.

S6.2.3 Expert Routing Agent

We use a Gated Recurrent Unit (GRU) [334] and dense layers to implement our Expert Routing Agent (ERA). As mentioned in Sec. 6.3.5, our ERA predicts architecture vectors ($u^{(i)}$, $v^{(i)}$) that determine (depth, width) pruning for the expert E_i. We assume that each expert has D depth pruning layers and L layers for width pruning. We show the detailed architecture in Tab. 6.3. We randomly initialize the inputs $z^{(i)}$ and keep them fixed during our pruning process. The values $C_k^{(i)}$ when $k \in [1, \cdots, L]$ are equal to the widths of the layers. In addition, $C_{L+1}^{(i)} = D$ as we use the outputs of the Dense_{L+1} layer to calculate the depth architecture vectors $u^{(i)}$.

Table 6.3: The architecture of our Expert Routing Agent. We calculate width architecture vectors $v^{(i)}$ from the outputs $o_k^{(i)}$ ($k \in [1, L]$). We compute the depth architecture vector $u^{(i)}$ from $o_{L+1}^{(i)}$. We refer to Sec. S6.2.3.1 for detailed formulations.

Inputs $z^{(i)} = [z_k^{(i)}]$, $(k = 1, \cdots, L+1)$, $(i = 1, \cdots, N)$
GRU(128, 256), WeightNorm, ReLU $\text{Dense}_k(256, C_k^{(i)})$, WeightNorm, $(k = 1, \cdots, L+1)$
Outputs $o_k^{(i)}$, $(k = 1, \cdots, L+1)$

S6.2.3.1 Formulation of Architecture Vectors

In this section, we describe our approach to calculate the architecture vectors $(u^{(i)}, v^{(i)})$ from the output vectors $o^{(i)}$ (Tab. 6.3) of the ERA.

Width Architecture Vectors: We design our width pruning method while considering our elastic width fine-tuning scheme. As mentioned in Sec. S6.2.2 and Fig. 6.11, we drop a random ratio of the least important convolution channels (attention heads) when training the convolution layers (attention layers) in the elastic width manner. We call each convolution channel and attention head a 'width unit.' Due to our dropping scheme, the weights of more important width units get trained more than the lower-importance ones in a layer and are robust to removing the lower-importance units.

Accordingly, we embed such a prior into the calculation of our width pruning architecture vectors $v^{(i)} = [v_l^{(i)}]_{l=1}^{L}$. Let's assume that the l-th layer of the expert E_i has W width units $[c_w]_{w=1}^{W}$ that are sorted based on their orders, namely c_1 is the most important unit, and c_W is the least important one. As mentioned in Sec. 6.3.5.1, the calculation from $v^{(i)}$ to $\mathbf{v}^{(i)}$ is called the Gumbel-Sigmoid trick [331, 332], which is a differentiable estimation of sampling from a Bernoulli distribution with the Bernoulli parameter of $\texttt{sigmoid}(v^{(i)})$. We calculate the vector $v_l^{(i)} = [v_{l,w}^{(i)}]_{w=1}^{W}$ from the output vector $o_l^{(i)}$ (Tab. 6.3) such that the Bernoulli parameters $\texttt{sigmoid}(v_{l,w}^{(i)})$ follow the importance order for the width units c_w. By doing so, the probability of preserving the more important width units is higher than low-importance ones. Specifically, we calculate $v_l^{(i)}$ as follows:

$$y_l^{(i)} = \text{Softmax}(o_l^{(i)}) \tag{6.19}$$

$$p_l^{(i)} = \text{cumsum}(y_l^{(i)}) \tag{6.20}$$

$$v_l^{(i)} = \text{inverse-sigmoid}(p_l^{(i)} - \epsilon) \tag{6.21}$$

In other words, first, we calculate the Softmax of the output logits vector $o_l^{(i)}$. Then, we take the cumulative summation of the elements of $y_l^{(i)}$ as $p_l^{(i)}$ such that $p_{l,e}^{(i)} = \sum_{w=e+1}^{W} y_{l,w}^{(i)}$. Thus, $p_{l,1}^{(i)} > p_{l,2}^{(i)} > \cdots > p_{l,W}^{(i)}$. Finally, we calculate the inverse sigmoid function for the elements of the probability vector $p_l^{(i)}$ to obtain the vector $v_l^{(i)}$ (the small constant ϵ ensures numerical stability of the inverse sigmoid function). Doing so ensures that the ERA will preserve the more important width units with a higher probability than the low-importance ones.

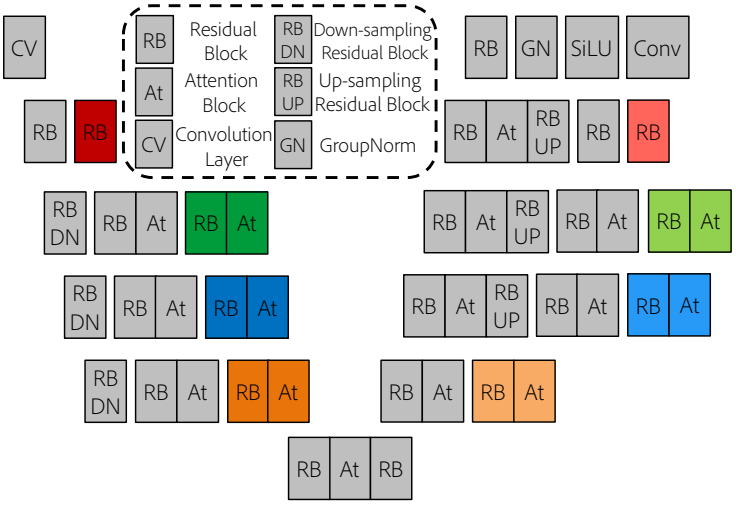


Figure 6.12: U-Net architecture of the LDM [1].

Depth Architecture Vectors: We calculate the depth architecture vectors $u^{(i)}$ similar to the scheme for $v_l^{(i)}$:

$$y_{L+1}^{(i)} = \text{Softmax}(o_{L+1}^{(i)}) \tag{6.22}$$

$$p_{L+1}^{(i)} = \text{cumsum}(y_{L+1}^{(i)}) \tag{6.23}$$

$$u^{(i)} = \text{inverse-sigmoid}(p_{L+1}^{(i)} - \epsilon) \tag{6.24}$$

For the depth layers, we empirically found that removing the shallower depth layers results in a more severe increase in training loss values as well as degradation in sample quality. Similarly we observed that for the depth blocks in the same stage of the U-Net, the depth block in the decoder branch is more crucial to the model's performance than the one in the encoder branch. Thus, we rank the depth blocks based on 1) their stage and 2) the branch of the U-Net that the block belongs to. For instance, we rank the depth pruning blocks in Fig 6.12 as: r = [decoder red block, encoder red block, decoder green block, encoder green block, decoder blue block, encoder blue block, decoder orange block, encoder orange block]. We then apply the elements of the soft relaxed depth pruning architecture vector $\mathbf{u}^{(i)}$ calculated from $u^{(i)}$ (Eq. 6.9) with the same order as the ranking r to the depth blocks. By doing so, the probabilities that our method preserve the depth pruning blocks will have the same order as the ranking r.

Table 6.4: Hyperparameters of fine-tuning our models with elastic dimensions.

		LSUN-Bedroom	LSUN-Church	FFHQ	ImageNet
Elastic Depth Fine-tuning	Batch Size×Num GPU	32×8	32×8	32×8	32×8
	learning rate	9.6e-5	5e-5	8.4e-5	8e-5
	Iterations	200k	50k	30k	40k
Elastic Width Fine-tuning	Batch Size×Num GPU	32×8	32×8	32×8	32×8
	learning rate	9.6e-5	5e-5	8.4e-5	8e-5
	Iterations	130k	50k	30k	40k

Algorithm 4 Our Pruning Algorithm

Input: Training dataset $\mathcal{D} = \{(x_m, c_m)\}_{m=1}^{D}$ of images x_i and possible conditional inputs c_i; ERA model $h_{\text{ERA}}(z;\beta)$; Elastically fine-tuned experts E_i; pruning iterations G; Total MACs budget T_d

Output: Trained Expert Routing Agent.

for $e := 1$ *to* G **do**

1. Sample a mini-batch $(\mathbf{x}, \mathbf{c})$ from $\mathcal{D}$.
2. Calculate architecture vectors $(u^{(i)}, v^{(i)})$ using the ERA model $h_{\text{ERA}}(z;\beta)$ and Eqs. 6.21, 6.24.
3. Compute soft pruning vectors $(\mathbf{u}^{(i)}, \mathbf{v}^{(i)})$ using Eqs 6.7, 6.9.
4. Apply the soft pruning vectors $(\mathbf{u}^{(i)}, \mathbf{v}^{(i)})$ to the experts using Eqs. 6.8, 6.10, 6.11.
5. Calculate the interval denoising objectives $\mathcal{L}_{\text{DDPM},\mathcal{I}_i}(E_i(u^{(i)}, v^{(i)}))$ for the experts using the samples $(\mathbf{x}, \mathbf{c})$ in the mini-batch.
6. Compute the MACs regularization term $\mathcal{R}(\widehat{T}(u, v), T_d)$.
7. Compute the training objective $\mathcal{J}(T_d)$, backpropagate the gradients $w.r.t$ the ERA parameters β and update them.

end

Return: Trained ERA model.

S6.2.4 Pruning Algorithm

We present our pruning algorithm to train our Expert Routing Agent to select proper sub-networks of the experts in Alg. 4.

S6.3 Experiments

We provide more details about our experimental setup as well as experimental results in this section.

S6.3.1 Setup

We implement our method upon the LDM codebase[3] and mainly follow the hyperparameter settings of the LDM [1]. We refer to Tables (12, 13) of LDM [4] for hyperparameters

[3] https://github.com/CompVis/latent-diffusion

[4] https://arxiv.org/pdf/2112.10752.pdf

of the architecture of the pretrained models that we use in our experiments. We use the DDIM sampler [286] for sampling from our pruned models. We set the number of sampling steps to 100 for the LSUN-Bedroom, 200 for the FFHQ, and 250 for the ImageNet experiments. We conduct all of our experiments on a server with 8 NVIDIA A100 GPUs. We calculate the inference throughput value for each model by sampling a batch of 64 samples from it 100 times and averaging the throughput values. We provide more details of our experimental setup for each stage of our method in the following.

S6.3.1.1 Fine-tuning with Elastic Dimensions

Tab 6.4 summarizes the hyperparameters that we use to fine-tune our experts with elastic depth and width on their intervals. We adopt the learning rate values from the settings used to train the pre-trained checkpoints in the LDM [1] paper.

S6.3.1.2 Pruning and Fine-tuning

We provide the hyperparameters for the pruning and fine-tuning stages of our method in Tab. 6.5.

For the pruning stage of our method on all datasets, we use the AdamW optimizer [335] with a learning rate of 0.001, weight decay of 0.01, and beta parameters $(\beta_1, \beta_2) = (0.9, 0.999)$ to train the ERA model's parameters. We also set the temperature parameter τ for the Gumbell-Sigmoid [332] estimations to 0.4 for all experiments.

Table 6.5: Hyperparameters for the pruning and fine-tuning stages of our method for different MACs pruning ratios (30%, 50%, and 70%).

		Pruning			Fine-tuning		
Dataset	Parameters	30%	50%	70%	30%	50%	70%
LSUN-Bedroom	Batch Size×Num GPU	12×8	12×8	12×8	32×8	32×8	24×8
	learning rate	-	-	-	9.6e-5	9.6e-5	9.6e-5
	Iterations	70k	60k	50k	270k	220k	195k
LSUN-Church	Batch Size×Num GPU	12×8	12×8	12×8	32×8	32×8	24×8
	learning rate	-	-	-	5e-5	5e-5	5e-5
	Iterations	70k	60k	50k	165k	180k	90k
FFHQ	Batch Size×Num GPU	12×8	12×8	12×8	32×8	32×8	24×8
	learning rate	-	-	-	8.4e-5	8.4e-5	8.4e-5
	Iterations	40k	30k	20k	90k	85k	100k
ImageNet	Batch Size×Num GPU	12×8	12×8	12×8	32×8	32×8	24×8
	learning rate	-	-	-	8e-5	8e-5	8e-5
	Iterations	50k	40k	30k	205k	130k	135k

S6.3.1.3 Ablation Experiments

We prune all the baselines in the ablation experiments for 60k iterations. Then, we match their fine-tuning iterations with the summation of the iterations of our elastic depth fine-tuning, elastic width fine-tuning, and fine-tuning the mixture of efficient experts for the 50% MACs budget (550k iterations) for a fair comparison.

S6.3.2 Comparison of Training Iterations

We provide a comparison of the number of training iterations for different methods to obtain a pruned model on LSUN-Bedroom and LSUN-Church in Tabs. 6.6, 6.7 respectively. For example, our method's total number of iterations is the summation of iterations for pre-training, elastic depth fine-tuning, elastic width fine-tuning, pruning, and fine-tuning the mixture of experts.

Although not directly comparable to SP [9] as it prunes a pixel-space DPM, our method can obtain a pruned model with significantly better quality with less iterations than SP. This shows that LDMs have much fewer redundancies than pixel-space DPMs. Thus,

Table 6.6: Comparison of the number of training iterations for different methods on LSUN-Bedroom. The "Method's Iterations" column denotes the number of all the training iterations that the pruning method performs to obtain its final efficient model.

LSUN-Bedroom (256 × 256)						
	Complexity				Performance	
Model	Pre-training Iterations	Method's Iterations	Total Iterations	MACs	Throughput (↑) (Sample/Sec)	FID (↓)
DDPM [11]	2.4M	-	2.4M	248.7G	0.74	6.62
SP [9]	2.4M	0.2M	2.6M	138.8G	-	18.6
LDM [1]	1.9M	-	1.9M	101.32G	2.01	4.39
DiffPruning (70%)	1.9M	0.575M	2.475M	70.84G	3.11	5.90
DiffPruning (50%)	1.9M	0.61M	2.51M	50.69G	3.75	6.73
DiffPruning (30%)	1.9M	0.67M	2.57M	31.11G	4.73	9.22

they can converge faster and pruning them is more challenging.

On LSUN-Church, on the one hand, DiffPruning (70%) converges with only 0.74M iterations while pixel-space pruned model by SP [9] requires 4.9M iterations to converge with a 4.51 worse FID score. On the other hand, the comparison results with OMS-DPM [10] clearly demonstrate the value of our elastic fine-tuning. DiffPruning 50% and 30% require more than 7× less training iterations than OMS-DPM to obtain a performant mixture of efficient experts. The reason is that our elastic fine-tuning scheme provides an implicit model zoo within the experts for each interval without requiring to train multiple models from scratch to obtain a model zoo as done in OMS-DPM [10].

S6.3.3 Errors in MACs Calculation

We mentioned in the caption of Tab. 6.1 as well as Sec. 6.4.1, the MACs values reported by SP [9] for the LDM [1] models for the ImageNet experiments are inaccurate. We describe the reason in the following. SP[9] adopts the 'flops-counter.pytorch' package[5] to measure models' MACs. This package defines a hook for each of the standard PyTorch [101]

[5]https://github.com/sovrasov/flops-counter.pytorch

Table 6.7: Comparison of the number of training iterations for different methods on LSUN-Church. The "Method's Iterations" column denotes the number of all the training iterations that the pruning method performs to obtain its final efficient model.

LSUN-Church (256 × 256)						
	Complexity				Performance	
Model	Pre-training Iterations	Method's Iterations	Total Iterations	MACs	Throughput (↑) (Sample/Sec)	FID (↓)
LDM [1]	0.5M	-	0.5M	20.96	5.19	5.21
DDPM [11, 286]	4.4M	-	4.4M	248.7G	0.74	10.58
SP [9]	4.4M	0.5M	4.9M	138.8G	-	13.9
DiffPruning (70%)	0.5M	0.24M	0.74M	14.64G	5.73	9.39
OMS-DPM [10]	0	>6M	>6M	-	2.56	11.10
DiffPruning (50%)	0.5M	0.34M	0.84M	10.48G	6.28	10.89
OMS-DPM [10]	0	>6M	>6M	-	6.4	13.7
DiffPruning (30%)	0.5M	0.335M	0.835M	6.35G	6.87	11.39

layers like `nn.Conv2d` and keeps a mapping dictionary between standard PyTorch layers and their hooks. The package calculates the MACs of the model by performing the forward pass of the model with a random input and counting the layers' MACs using the defined hooks. Now, SP [9] implements the Attention layer in the U-Net architecture of LDM manually, and the defined Attention module is not an element of the mapping dictionary for the MACs calculation hooks. Thus, the package does not count the number of MACs for the scaled dot product attention operation as it is not a native PyTorch layer. For instance, SP reports that (Tab. (3) in SP [9]) the LDM model for ImageNet has 99.8G MACs. However, we manually implemented counting the MACs for the attention layers and found that the model actually has 108.78G MACs.

We found a similar problem in the numbers reported by SD [293] . For instance, SD reports that the LDM for LSUN-Church has 18.7G MACs. We could reproduce the same number when directly using the 'flops-counter.pytorch' package. Yet, we found that the model actually has 20.96G MACs after adding the attention layers' MACs.

Chapter 7: Not All Prompts Are Made Equal: Prompt-based Pruning of Text-to-Image Diffusion Models

7.1 Introduction

In recent years, diffusion models [11, 272, 273, 336] have revolutionized generative modeling. For instance, modern Text-to-Image (T2I) diffusion models [17, 19, 337] like Stable Diffusion [1, 22] have achieved remarkable success in generating realistic images [18, 338, 339] and editing them [275, 340, 341]. Consequently, their integration into various applications is of great interest. However, the sampling process of T2I diffusion models is slow and computationally intensive, making them expensive to deploy on GPU clouds for a large number of users and preventing their utilization on edge devices. Thus, reducing the computational cost of T2I models is essential prior to serving them.

Two orthogonal factors underlie the sampling cost of T2I diffusion models: their large number of denoising steps and their parameter-heavy backbone architectures. Most acceleration methods for T2I models reduce the computation per sampling step or skip steps using techniques like distillation [289, 290, 342] and improved noise schedules [286, 287]. Other ideas address the second factor and propose efficient architectures for T2I models. Architecture modification methods [2, 308] modify the U-Net [295] of Stable Diffusion (SD) [1] to

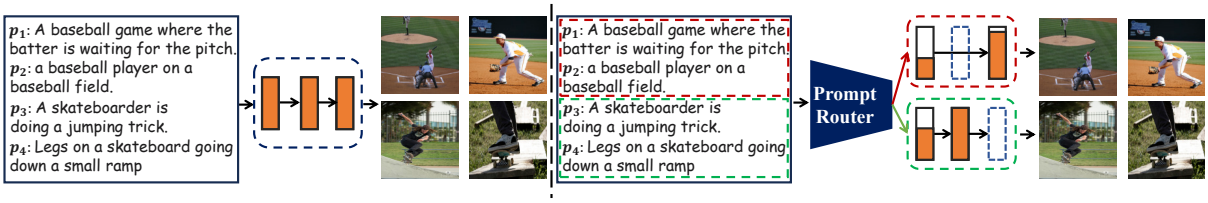


Figure 7.1: **Overview:** We prune a text-to-image diffusion model like Stable Diffusion (left) into a mixture of efficient experts (right) in a prompt-based manner. Our prompt router *routes* distinct types of prompts to different experts, allowing experts' architectures to be separately specialized by removing layers or channels.

reduce its parameters and FLOPs while preserving performance. Still, they only provide a few architectural configurations, and generalizing their design choices to new compute budgets or datasets is highly non-trivial. Search-based methods [10, 343] search for an efficient architecture [343] or an efficient mixture [10] of pretrained T2I models in a model zoo. Yet, evaluating each action in the search process and gathering a model zoo of T2I models are both extremely costly, making search methods impractical in resource-constrained scenarios.

The efficient architecture design methods [2, 308, 343] have shown promise, but they aim to develop 'one-size-fits-all' architectures for *all* applications. We argue that this approach is misaligned with the common practice for two reasons: 1) Organizations typically fine-tune pretrained T2I models (*e.g.,* SD) on their proprietary *target* data and deploy the resulting model for their specific application. They prioritize performance on their *target* data distribution while meeting their computation budget, and the trade-off between efficiency and performance can vary depending on the complexity of the *target* dataset between organizations. 2) Verifying the validity of each design choice on large-scale datasets used in one-size-fits-all methods is costly and slow to iterate, making them impractical.

Model pruning [322] can reduce a model's computational burden to any desired budget

with significantly less effort than designing [2, 308] or searching [343] for efficient architectures. Still, T2I models have unique characteristics making existing pruning techniques unsuitable for them. Static pruning methods are input-agnostic and use the same pruned model for all inputs, but distinct prompts of T2I models may require different model capacities. Dynamic pruning employs a separate model for each input sample, but it cannot benefit from batch-parallelism in modern hardware like GPUs and TPUs.

In this chapter, we introduce Adaptive Prompt-Tailored Pruning (APTP), a novel prompt-based pruning method for T2I diffusion models. APTP prunes a T2I model pretrained on a large-scale dataset (*e.g.,* SD) using a smaller *target* dataset given a desired compute budget. It tackles the challenges of static and dynamic pruning methods for T2I models by training a *prompt router* module along with a set of *architecture codes*. The prompt router learns to route an input prompt to an architecture code, determining the sub-architecture, called *expert*, of the T2I model to use. Each expert specializes in generating images for the prompts assigned to it by the prompt router (Fig. 7.1), and the number of experts is a hyperparameter. We train the prompt router and architecture codes using a contrastive learning objective that regularizes the prompt router to select similar architecture codes for similar prompts. In addition, we employ optimal transport to diversify the architecture codes and the resulting experts' budgets, allowing deploying them on hardware with varying capabilities. We take CC3M [13] and MS-COCO [3] as the *target* datasets and prune Stable Diffusion V2.1 [1] using APTP in our experiments. APTP outperforms the single-model pruning baselines, and we show that our prompt router learns to group the input prompts into semantic clusters. Further, our analysis demonstrates that APTP can automatically discover challenging prompts for SD, such as prompts for generating

text images, found empirically by prior work [344, 345]. We summarize our contributions as follows:

- We introduce APTP, a novel prompt-based pruning method for T2I diffusion models. It is more suitable than static pruning for T2I models as it is not input-agnostic. In addition, APTP enables batch parallelism on GPUs, which is not possible with dynamic pruning.
- APTP trains a prompt router and a set of architecture codes. The prompt router maps an input prompt to an architecture code. Each architecture code resembles a pruned sub-architecture expert of the T2I model, specialized in handling certain types of prompts assigned to it.
- We develop a framework to train the prompt router and architecture codes using contrastive learning and employ optimal transport to diversify the architecture codes and their corresponding sub-architectures' computational requirements, given a desired compute budget.

The contents of this chapter are based on our work [346] published in ICLR 2025.

7.2 Related Work

Efficient Diffusion Models: Methods for accelerating and improving the efficiency of diffusion models fall into two categories. The first group focuses on reducing the complexity or the number of sampling steps of diffusion models. They use techniques like distillation [289, 290, 342, 347], learning optimal denoising time-steps [302, 303], designing

improved noise schedules [286, 287, 300], caching intermediate computations [304, 348, 349], and proposing faster solvers [297, 298, 350]. The second group is aimed at improving the architectural efficiency of diffusion models, which is the main focus of our paper. **Multi-expert** [10, 292, 309, 351] methods employ multiple expert models each responsible for a separate part of the denoising process of diffusion models. These methods either design expert model architectures [292, 351], train several models with varying capacities from scratch [10], or utilize existing pretained experts [10, 309] with different capacities. However, pretrained experts are not necessarily available, and training several models from scratch, as required by multi-expert methods, is prohibitively expensive in practice. **Architecture Design** approaches [2, 293, 308] redesign the architecture of diffusion models to enhance efficiency. MobileDiffusion [308] modifies the U-Net [295] architecture of Stable Diffusion (SD) based on empirical heuristics derived from the model's performance on the MS-COCO [3] dataset. BK-SDM [2] removes some blocks from the U-Net model of SD and applies knowledge distillation from the original SD model to the pruned model. Spectral Diffusion [293] introduces a wavelet gating operation and performs frequency domain distillation from a pretrained teacher model into a small student model. Yet, generalization of the heuristics and design choices in the architecture design methods to other tasks and compute budgets is non-trivial. **Quantization** methods [352, 353, 354, 355, 356, 357, 358, 359] reduce the precision of model weights and activations during the forward pass to accelerate the sampling process. Different from these methods, SnapFusion [343] searches for an efficient architecture for T2I models. However, evaluating each action in the search process consumes about 2.5 A100 GPU hours, rendering it impractical for resource-constrained scenarios. Finally, SPDM [9] estimates the importance of different weights in the model

using Taylor expansion and removes the low-scored architectures. Despite their promising results, all these methods are 'static' in that they obtain an efficient model and utilize it for all inputs. This is suboptimal for T2I models as input prompts may vary in complexity, demanding different model capacity levels.

Pruning and Neural Architecture Search (NAS): Our paper also intersects with model pruning [322] and NAS [315, 316, 317, 318, 319, 320] methods. These methods prune pretrained models and search for suitable architectures given a specific task and computational budget. Existing pruning techniques can be categorized into Static and Dynamic methods. Static pruning approaches [310, 311, 313] prune a pretrained model and use the pruned model for all inputs. Conversely, dynamic pruning methods [360, 361, 362, 363] employ a separate sub-network of the model for each input. Although static and dynamic pruning methods have been successful for image classification, they are not suitable to be directly applied to T2I models. Static pruning methods neglect prompt complexity and employ the same model for all prompts while dynamic pruning ideas cannot utilize batch-parallelism in GPUs. We refer to recent surveys [320, 321, 322] for a comprehensive review of pruning and NAS methods.

Our method distinguishes itself from existing approaches by introducing a prompt-based pruning technique for T2I models. This work marks the first instance where computational resources are allocated to prompts based on their individual complexities while ensuring optimal utilization and batch-parallelizability.

7.3 Method

We introduce a framework for prompt-based pruning of T2I diffusion models termed Adaptive Prompt-Tailored Pruning (ATPT). APTP prunes a T2I model pretrained on large-scale datasets (*e.g.*, Stable Diffusion [1]) using a smaller *target* dataset. This approach mirrors the common practice where organizations fine-tune pretrained T2I models on their internal proprietary data and deploy the resulting model for their customers. The core component of APTP is a *prompt router* that learns to map an input prompt to an *architecture code* during the pruning process. Each architecture code corresponds to a specialized *expert* model, which is a sub-network of the T2I model. We train the prompt router and architecture codes in an end-to-end manner. After the pruning phase, APTP fine-tunes each specialized model using samples from the target dataset assigned to it by the prompt router. We elaborate on the components of APTP in the following subsections.

7.3.1 Background

Given an input text prompt, text-to-image (T2I) diffusion models generate a corresponding image by iteratively denoising a Gaussian noise [11, 272, 273]. They achieve this by training a denoising model, $\epsilon(\cdot;\theta)$, parameterized by θ. For a given training image-text pair $(x_0, p) \sim \mathcal{P}$, T2I models define a forward diffusion process, progressively adding Gaussian noise to the initial image x_0 over T steps. This process is defined as $q(x_t|x_0) = \mathcal{N}(x_t; \sqrt{\bar{\alpha}_t}x_0, (1-\bar{\alpha}_t)I)$, where $\bar{\alpha}_t$ is the forward noise schedule parameter, typically chosen such that $q(x_t|x_0) \to \mathcal{N}(0, I)$ as $t \to T$. T2I models are trained using the variational evidence lower bound (ELBO) objective [11]:

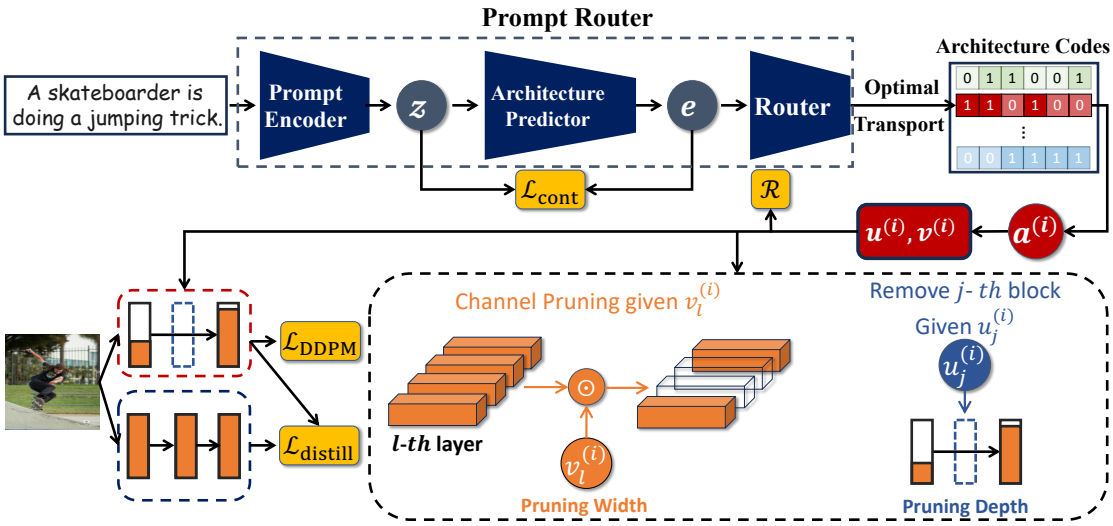


Figure 7.2: **Our pruning scheme.** We train our prompt router and the set of architecture codes to prune a text-to-image diffusion model into a mixture of experts. The prompt router consists of three modules. We use a Sentence Transformer [12] as our prompt encoder to encode the input prompt into a representation z. Then, the architecture predictor transforms z into the architecture embedding e that has the same dimensionality as architecture codes. Finally, the router routes the embedding e into an architecture code $a^{(i)}$. We use optimal transport to evenly assign the prompts in a training batch to the architecture codes. The architecture code $a^{(i)} = (u^{(i)}, v^{(i)})$ determines pruning the model's width and depth. We train the prompt router's parameters and architecture codes in an end-to-end manner using the denoising objective of the pruned model $\mathcal{L}_{\text{DDPM}}$, distillation loss between the pruned and original models $\mathcal{L}_{\text{distill}}$, average resource usage for the samples in the batch $\mathcal{R}$, and contrastive objective $\mathcal{L}_{\text{cont}}$, encouraging embeddings e preserving semantic similarity of the representations z.

$$\mathcal{L}_{\text{DDPM}}(\theta) = \mathbb{E}_{\substack{(x_0,p)\sim\mathcal{P} \\ t\sim[1,T] \\ \epsilon\sim\mathcal{N}(0,I) \\ x_t\sim q(x_t|x_0)}} ||\epsilon(x_t, p, t; \theta) - \epsilon||^2 \tag{7.1}$$

T2I models sample an image starting from a Gaussian noise $x_T \sim \mathcal{N}(0, I)$ and denoising it using the trained denoising model. We refer to supplementary S7.1 for a thorough review of diffusion models' formulation.

7.3.2 Prompt Router and Architecture Codes

We employ our prompt router to determine the specialized sub-network of the T2I model to be used for a given input prompt. Specifically, we denote our prompt router with the function $f_{\text{PR}}(\cdot; \eta, \mathcal{A})$, parameterized by η, which maps an input prompt p to an architecture code a:

$$a = f_{\text{PR}}(p; \eta, \mathcal{A}) \tag{7.2}$$

We define the set of learnable architecture codes as $\mathcal{A} = \{a^{(i)}\}_{i=1}^{N}$, where N is a hyper-parameter. Also, $a^{(i)} \in \mathbb{R}^D$, with D being the number of prunable width and depth units in the T2I model. Given a desired constraint T_d on the total compute budget (latency, MACs, etc.) for the prompts in the target dataset, we train the prompt router's parameters η and architecture codes in $\mathcal{A}$ to obtain a set of expert models. These experts are specialized, efficient, and performant sub-networks of the T2I model, as determined by the architecture codes. As illustrated in Fig. 7.2, the prompt router consists of a prompt encoder, an architecture predictor, and a router module. We provide details of these components in the following subsections and present our pruning objective in Eq. 7.15.

7.3.2.1 Prompt Encoder and Architecture Predictor

Our primary intuition in designing the prompt router is that it should route semantically similar prompts to similar sub-networks of a T2I model. Accordingly, we use a pretrained frozen Sentence Transformer model [12] as our prompt encoder module. It can

effectively encode input prompts p into semantically meaningful embeddings z:

$$z = f_{\text{PE}}(p) \tag{7.3}$$

$f_{\text{PE}}(\cdot)$ is the prompt encoder. We do not explicitly show the prompt encoder's parameters as we do not train them in our pruning method. The Sentence Transformer can encode semantically similar prompts to nearby embeddings and distant from dissimilar ones. We leverage this property in our framework (Eq. 7.13) to ensure the prompt router maps similar prompts to similar architecture codes.

The architecture predictor module transforms prompt embeddings z into architecture embeddings e:

$$e = f_{\text{AP}}(z; \eta) \tag{7.4}$$

η denotes the parameters of the architecture predictor $f_{\text{AP}}(\cdot)$, implemented with a single feed-forward layer (more details in supplementary S7.2). The embeddings e have the same dimensionality as the architecture codes a.

7.3.2.2 Router

The router module takes an architecture embedding e and routes it to an architecture code $a \in \mathcal{A}$. A straightforward way to implement the router is to map an input embedding e to its nearest architecture code in $\mathcal{A}$. However, we found that this approach may lead to the collapse of architecture codes, as they could converge to a single code that meets the desired compute budget T_d with a relatively decent performance, causing the prompt

router to route all input prompts to it.

To tackle this challenge, we employ optimal transport in our router module during the pruning phase. Formally, let $E = [e_1, \cdots, e_B]$ represent B architecture embeddings in a training batch, and $A = [a^{(1)}, \cdots, a^{(N)}]$ represent the N codes. The goal is to find an assignment matrix $Q = [q_1, \cdots, q_B]$ that maximizes the similarity between architecture embeddings and their assigned architecture codes:

$$\max_{Q \in \mathcal{Q}} \mathrm{Tr}(Q^T A^T E) + \epsilon \mathrm{H}(Q) \tag{7.5}$$

H is an entropy term $\mathrm{H}(Q) = -\sum_{i,j} q_{ij} \log(q_{ij})$ and ϵ is the regularization strength. It has been shown [364, 365] that high values of ϵ lead to a uniform assignment matrix Q, causing all codes in A to collapse to a single code. Thus, we set ϵ to a small value. Further, we impose an equipartition [364] constraint on Q so that architecture embeddings in a batch are assigned equally to architecture codes:

$$\mathcal{Q} = \{Q \in \mathbb{R}_+^{N \times B} \mid Q 1_B = \frac{1}{N} 1_N,\ Q^T 1_N = \frac{1}{B} 1_B\} \tag{7.6}$$

Here, $1_{\{B,N\}}$ are vectors of ones with length B and N, respectively. These constraints enforce that, on average, $\frac{B}{N}$ embeddings are assigned to an architecture code per training batch, thereby ensuring that each architecture code gets enough samples for training. The optimal transport problem in Eq. 7.5 with the constraints in Eq. 7.6 can be solved using the fast version [366] of the Sinkhorn-Knopp algorithm and the solution has the normalized exponential matrix form [366]:

$$Q^* = \text{diag}(m)\exp(\frac{A^T E}{\epsilon})\text{diag}(n) \tag{7.7}$$

Here, m and n are renormalization vectors that can be calculated using a few iterations of the Sinkhorn-Knopp algorithm. With ϵ set to a small value, the matrix BQ^* will have columns that are close to one-hot vectors, which we use to assign the architecture embeddings to the architecture codes. In summary, the router module's function during the pruning process is:

$$a = f_{\text{R}}(e, \mathcal{A}; Q^*) \tag{7.8}$$

After the pruning stage, the trained router simply routes an input architecture embedding to an architecture code with the highest cosine similarity. We provide the definition of f_{R} in supplementary S7.2.2.

7.3.3 Pruning

We divide each architecture code $a^{(i)} \in \mathcal{A}$ into two sub-vectors $a^{(i)} = (u^{(i)}, v^{(i)})$. We utilize the vectors $u^{(i)}$ and $v^{(i)}$ to prune the depth layers and determine widths of the layers, respectively.

A simple approach to prune the width of a layer (a depth layer) is to use binary vectors $v^{(i)}$ ($u^{(i)}$), indicating whether the channels (layers) should be pruned. Yet, doing so is not differentiable, and one needs to solve a discrete optimization problem to find the optimal binary vectors. Instead, we employ soft vectors $\mathbf{v}^{(i)}$ ($\mathbf{u}^{(i)}$) that are continuous and differentiable for pruning. We calculate them as:

$$\mathbf{v}^{(i)} = \text{sigmoid}(\frac{v^{(i)} + g_v}{\gamma}), \qquad \mathbf{u}^{(i)} = \text{sigmoid}(\frac{u^{(i)} + g_u}{\gamma}) \tag{7.9}$$

$g_{\{u,v\}} \sim \text{Gumbel}(0, 1)$ represents a noise vector sampled from the Gumbel distribution [367], and γ is the temperature. This formulation, known as the Gumbel-sigmoid reparameterization [368, 369], is a differentiable approximation of sampling from Bernoulli distributions with parameters $\texttt{sigmoid}(v^{(i)})$ and $\texttt{sigmoid}(u^{(i)})$. When the temperature γ is set appropriately, the vectors $\mathbf{v}^{(i)}$ and $\mathbf{u}^{(i)}$ will be close to binary vectors, and we use them for pruning the layers' width and depth layers. Assuming $\mathbf{v}^{(i)} = [\mathbf{v}_l^{(i)}]_{l=1}^{L}$ where L is the number of the model's layers, we prune the width of the l-th layer as:

$$\widehat{\mathcal{F}}_l = \mathcal{F}_l \odot \mathbf{v}_l^{(i)} \tag{7.10}$$

In this equation, $\mathcal{F}_l$ represents feature maps of the l-th layer, and $\odot$ is the element-wise multiplication. We prune the channels of the convolution layers in the ResBlocks [323], attention heads in the Transformer layers [324], and the channels of the feed-forward layers in the Transformer layers.

In a similar manner, we apply the vectors $\mathbf{u}^{(i)} = [\mathbf{u}_j^{(i)}]_{j=1}^{M}$ (where M is the number of depth layers that we prune) for pruning the model's depth. Specifically, we prune the j-th depth layer f_j as:

$$\widehat{\mathcal{F}}_j = \mathbf{u}_j^{(i)} f_j(\mathcal{F}_{j-1}) + (1 - \mathbf{u}_j^{(i)})\mathcal{F}_{j-1} \tag{7.11}$$

$\mathcal{F}_{j-1}$ denotes the previous layer's feature maps. The granularity of our depth pruning

is a ResBlock or a Transformer layer in the U-Net of the T2I model. We provide more details in supplementary S7.2.3. In summary, we employ the architecture vectors $\mathbf{a}^{(i)} = (\mathbf{v}^{(i)}, \mathbf{u}^{(i)})$ to prune the T2I model.

7.3.3.1 Training the Prompt Router and Architecture Codes

We jointly train the prompt router and architecture codes in an end-to-end manner, guiding the prompt router to map similar prompts to similar architecture codes. In addition, we regularize the architecture codes to correspond to performant sub-networks of the T2I model, be diverse, and adhere to the desired compute budget T_d on aggregate.

Contrastive Training. Given a training batch with B prompts, we compute their prompt embeddings z (Eq. 7.3) and architecture embeddings e (Eq. 7.4). Then, we calculate architecture vectors $\mathbf{e}'$:

$$\mathbf{e}' = \text{sigmoid}(\frac{e+g}{\gamma}) \tag{7.12}$$

where γ and g have the same definitions as Eq. 7.9. We define the cosine similarity of two vectors as $\text{sim}(\mathbf{m}, \mathbf{n}) = \mathbf{m}^T\mathbf{n}/||\mathbf{m}|| \cdot ||\mathbf{n}||$, and we use the following objective to train the architecture predictor:

$$\mathcal{L}_{\text{cont}}(\eta) = \frac{1}{B^2}\sum_{i=1}^{B}\sum_{j=1}^{B} r_{i,j}\log(s_{i,j}) + (1-r_{i,j})\log(1-s_{i,j}) \tag{7.13}$$

$$r_{i,j} = \frac{\exp(\text{sim}(z_i, z_j)/\tau)}{\sum_{k=1}^{B}\exp(\text{sim}(z_i, z_k)/\tau)}, \qquad s_{i,j} = \frac{\exp(\text{sim}(\mathbf{e}'_i, \mathbf{e}'_j)/\tau)}{\sum_{k=1}^{B}\exp(\text{sim}(\mathbf{e}'_i, \mathbf{e}'_k)/\tau)} \tag{7.14}$$

τ is a temperature parameter. Eq. 7.13 regularizes the architecture predictor to map

representations z to the regions of the space of the architecture embeddings e such that their corresponding architecture vectors $\mathbf{e}'$ maintain the similarity between the prompts. Note that we do not actually use the architecture vectors $\mathbf{e}'$ to prune the model (Sec. 7.3.2.2, 7.3.3). Yet, we apply our contrastive regularization to the vectors $\mathbf{e}'$ instead of embeddings e. This design choice is a result of our observation that applying $\mathcal{L}_{\text{cont}}$ to embeddings e may not lead to diverse architecture vectors $\mathbf{a}^{(i)} = (\mathbf{v}^{(i)}, \mathbf{u}^{(i)})$ that we use for pruning (Sec. 7.3.3). For instance, embeddings e (and their nearby architecture codes a) can be distributed in the embedding space (before Gumbel-Sigmoid estimation), but all be in saturation regions of the sigmoid function (Eqs. 7.9, 7.12), resulting in similar architecture vectors $\mathbf{e}'$ and $\mathbf{a}^{(i)}$.

In contrast, Eq. 7.13 implicitly diversifies the architecture vectors $\mathbf{a}^{(i)}$. The reason is that the router routes embeddings e to codes a (Eq. 7.8) that have high similarity with each other (Eq. 7.7). Also, Eq. 7.13 distributes embeddings e in the space of the architecture embeddings such that the architecture vectors $\mathbf{e}'$ become similar (different) for similar (dissimilar) prompts. As the vectors $\mathbf{e}'$ and $\mathbf{a}^{(i)}$ are calculated in a similar manner (Eqs. 7.9, 7.12), the diversity of vectors $\mathbf{e}'$ implies the same for vectors $\mathbf{a}^{(i)}$.

Assuming $B^{(i)}$ samples get routed to the architecture code $a^{(i)}$ in a training batch ($\sum_i B^{(i)} = B$), we train the prompt router and the architecture codes using the following objective:

$$\begin{aligned}\min_{\eta,\mathcal{A}} \mathcal{L} =&[\frac{1}{N}\sum_{i=1}^{N}[\frac{1}{B^{(i)}}\sum_{j=1}^{B^{(i)}}[\mathcal{L}_{\text{DDPM}}(x_j^{(i)}, p_j^{(i)}; a^{(i)}) + \lambda_{\text{distill}}\mathcal{L}_{\text{distill}}(x_j^{(i)}, p_j^{(i)}; a^{(i)})]]] \\ &+ \lambda_{\text{res}}\mathcal{R}(\widehat{T}(\mathcal{A}), T_d) + \lambda_{\text{cont}}\mathcal{L}_{\text{cont}}(\eta)\end{aligned} \tag{7.15}$$

$\mathcal{L}_{\text{DDPM}}((x_j^{(i)}, p_j^{(i)}); a^{(i)})$ denotes the denoising objective for the sample $(x_j^{(i)}, p_j^{(i)})$ routed to sub-network chosen by the architecture code $a^{(i)}$. $\mathcal{R}(\widehat{T}(\mathcal{A}), T_d)$ regularizes the weighted average of the MACs used by architecture codes ($\widehat{T}(\mathcal{A}) = \sum_i \frac{B^{(i)}}{B}[\widehat{T}(a^{(i)})]$) to be close to T_d. We define $\mathcal{R}(x, y) = \log(\max(x, y)/\min(x, y))$, and $\{\lambda_{\text{distill}}, \lambda_{\text{res}}, \lambda_{\text{cont}}\}$ are hyperparameters. Finally, $\mathcal{L}_{\text{distill}}$ is the distillation objective [2] regularizing the pruned model having similar outputs to the original one. We refer to supplementary S7.2.4 for more details about our distillation objective.

7.3.4 Fine-tuning the Pruned Expert Models

After the pruning stage, we use the learned architecture codes to prune the T2I model into our experts. Then, we fine-tune experts using samples routed to them. We use the same training objective as the pretraining stage while adding distillation to fine-tune experts and refer to supplementary S7.2.5 for more details. At test time, our trained prompt router routes an input prompt to one of the experts, and we use the expert to generate an image for it.

7.4 Experiments

We use Conceptual Captions 3M (CC3M) [13] and MS-COCO [3] as our *target* datasets to prune the Stable Diffusion (SD) V2.1 model to demonstrate APTP's effectiveness. On CC3M, we prune the model with Base: (0.85 MACs, 16 experts) and Small: (0.66 MACs, 8 experts) configurations. On MS-COCO, we perform Base: (0.78 MACs, 8 experts) and Small: (0.64 MACs, 8 experts) pruning settings. We set $(\lambda_{\text{distill}}, \lambda_{\text{res}}, \lambda_{\text{cont}})$=$(0.2, 2.0, 100)$

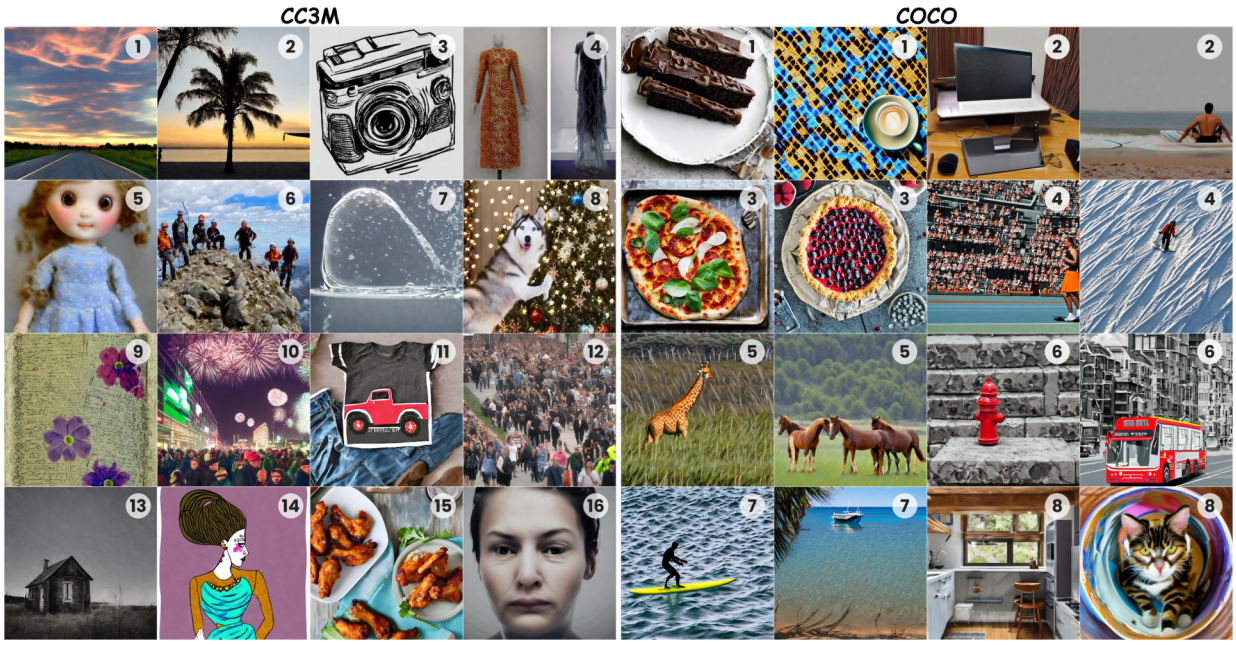


Figure 7.3: Samples of the APTP-Base experts after pruning the Stable Diffusion V2.1 using CC3M [13] and COCO [3] as the *target* datasets. Expert IDs are shown on the top right of images. (See Table 7.6 for prompts)

(Eq. 7.15) and the temperature τ (Eq. 7.14) to 0.03. We evaluate all models with FID [370], CLIP [371], and CMMD [372] scores using 14k samples in the validation set of CC3M and 30k samples from the MS-COCO's validation split. For all models, we sample the images at the resolution of 768 and resize them to 256 for calculating the metrics, using the 25-steps PNDM [373] sampler following BK-SDM [2]. We refer to supplementary S7.3 for more details.

7.4.1 Comparison Results

As APTP is the first pruning method specifically designed to prune a pretrained T2I model on a *target* dataset, we compare its performance with SD V2.1, weight norm pruning [310], and two recently proposed static pruning baselines, namely Structural Pruning (SP) [9] and BKSDM [2]. Table 7.1 shows the results. We fine-tune APTP, SP and BKSDM

for 30k iterations after pruning and give Norm-pruning 50k fine-tuning iterations to ensure they all reach their final performance level.

CC3M: Table 7.1a summarizes the results on CC3M. With a similar MACs budget and latency values, APTP (0.85) significantly outperforms the Norm pruning [310], SP [9], and BKSDM [2] baselines with a significant margin in terms of FID, CLIP, and CMMD scores. It also achieves 15% less latency while showing close performance scores to SD V2.1. Notably, APTP (0.66) has approximately 23% less latency and 21% lower MACs budget than the baselines, but it still outperforms them on all metrics. These results illustrate the advantages of prompt-based compared to static pruning.

MS-COCO: We present the results for MS-COCO in Table 7.1b. APTP (0.78) reduces the latency of SD by 22.5% while preserving its CLIP score and achieving a close CMMD score. Further, with a similar latency, APTP (0.78) significantly outperforms the static pruning baselines with at least 3.71 FID (BKSDM), 2.34 CLIP (SP), and 0.042 CMMD (BKSDM) scores. Similar to the results on CC3M, APTP (0.64) achieves approximately 19.3% less latency than the Norm pruning and SP while outperforming them on all scores. In summary, our quantitative evaluations show the clear advantage of prompt-based compared to static pruning for T2I models. We provide samples of APTP on the validation sets of CC3M and MS-COCO in Fig. 7.3 and Appendix S7.3.4.6.

7.4.2 Analysis of the Prompt Router

We analyze our prompt router's behavior by examining the prompts it assigns to experts in the APTP-Base (0.85) model for the CC3M experiment. Table 7.2 displays the

Table 7.1: Results on CC3M and MS-COCO. We report performance metrics using samples generated at the resolution of 768 then downsampled to 256 [2]. We measure models' MACs/Latency with the input resolution of 768 on an A100 GPU. @30/50k shows fine-tuning iterations after pruning.

CC3M					
Method	Complexity		Performance		
	MACs (@768)	Latency (↓) (Sec/Sample) (@768)	FID (↓)	CLIP (↑)	CMMD (↓)
Norm [310] @50k	1185.3G	3.4	141.04	26.51	1.646
SP [9] @30k	1192.1G	3.5	75.81	26.83	1.243
BKSDM [2] @30k	1180.0G	3.3	87.27	26.56	1.679
APTP(0.66) @30k	916.3G	2.6	60.04	28.64	1.094
APTP(0.85) @30k	1182.8G	3.4	36.77	30.84	0.675
SD 2.1	1384.2G	4.0	32.08	31.12	0.567

(a)

MS-COCO					
Method	Complexity		Performance		
	MACs (@768)	Latency (↓) (Sec/Sample) (@768)	FID (↓)	CLIP (↑)	CMMD (↓)
Norm [310] @50k	1077.4G	3.1	47.35	28.51	1.136
SP [9] @30k	1071.4G	3.3	53.09	28.98	0.926
BKSDM [2] @30k	1085.4G	3.1	26.31	28.89	0.611
APTP(0.64) @30k	890.0G	2.5	39.12	29.98	0.867
APTP(0.78) @30k	1076.6G	3.1	22.60	31.32	0.569
SD 2.1	1384.2G	4.0	15.47	31.33	0.500

(b)

Table 7.2: The most frequent words in prompts assigned to each expert of APTP-Base pruned on CC3M. The resource utilization of each expert is indicated in parentheses.

Expert 1 (0.72)	**Expert 2 (0.73)**	**Expert 3 (0.75)**	**Expert 4 (0.76)**
View - Sunset - City - Building - Sky	View - Boat - Sea	Artist - Actor	Actor - Dress - Portrait
Expert 5 (0.77)	**Expert 6 (0.78)**	**Expert 7 (0.79)**	**Expert 8 (0.79)**
Illustration - Portrait - Photo	Player - Ball - Game - Team	Background - Water - River - Tree	Biological Species - Dog - Cat
Expert 9 (0.79)	**Expert 10 (0.80)**	**Expert 11 (0.81)**	**Expert 12 (0.81)**
Illustration - Vector	People	Car - City - Road	Person - Player - Team - Couple
Expert 13 (0.86)	**Expert 14 (0.90)**	**Expert 15 (0.95)**	**Expert 16 (0.98)**
Room - House	Art - Artist - Digital	Food - Water	Person - Man - Woman - Text

most frequent words in the prompts routed to each expert, along with the experts' MACs budgets. Our prompt router effectively "specializes" experts by assigning distinct topics to experts with varying budgets. For instance, Expert 1 focuses on cityscapes, Expert 8 on animals, and Expert 13 on house interiors and exteriors. Notably, the prompt router assigns images of textual content and human beings to Expert 16, which has the highest budget. These categories have been empirically found to be challenging for SD 2.1 [344, 345], and our prompt router can automatically discover and route them to higher-capacity experts. In contrast, paintings and illustrations are assigned to lower-budget experts, as they seem to

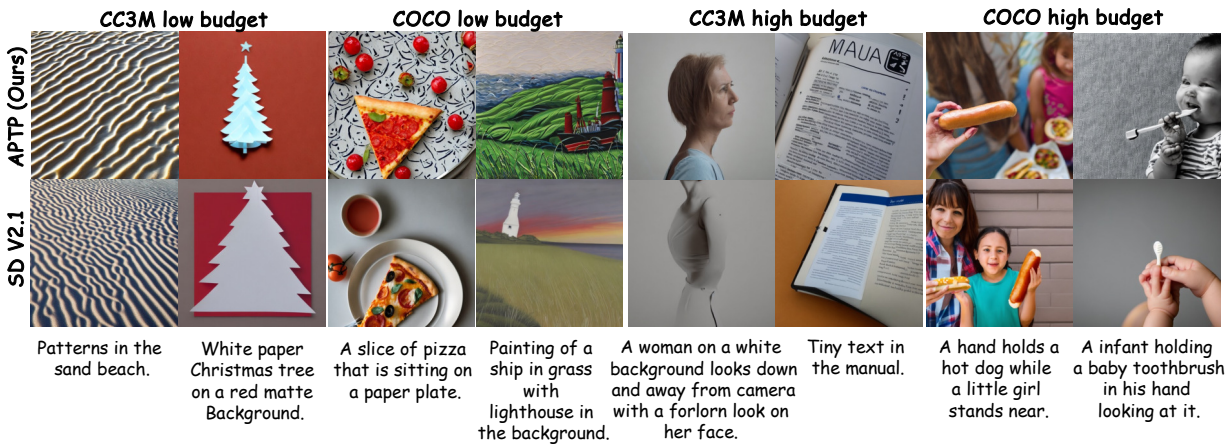


Figure 7.4: Comparison of samples generated by low and high budget experts of APTP-Base *vs.* SD V2.1 on CC3M and MS-COCO validation sets.

be easier for the model to generate. We provide examples of high and low resource prompts for APTP-Base on both CC3M and COCO in Fig. 7.4.

7.4.3 Ablation Study

We conduct two ablation experiments to study the impact of APTP's components on its performance. First, we implement a naive baseline that uses parameterizations $\mathbf{v} = \text{sigmoid}((\eta_v + g_v)/\tau)$ and $\mathbf{u} = \text{sigmoid}((\eta_u + g_u)/\tau)$ (Eq. 7.9) to prune a single model. It directly trains two vectors (η_v, η_u) to do so ('Uni-Arch Baseline' in Table 7.3). Then, we start with a router trained only using the contrastive objective (Eq. 7.13) and add optimal transport and distillation incrementally to prune SD V2.1 into 8 experts with 80% MACs budget on MS-COCO [3]. We fine-tune all pruned models with 10k iterations, and Table 7.3 presents the results. We observe that the contrastive training (Eq. 7.13) alone fails to improve results from pruning a single model to a mixture of experts. This is because although the contrastive objective makes the architecture codes diverse, it does not enforce

the prompt router to distribute the prompts between architecture codes. Thus, it routes most of the input prompts to a single expert. As a result, all experts except one receive insufficient training samples and generate low-quality samples after fine-tuning, leading to the mixture showing worse metrics than the baseline. Employing optimal transport (Eq. 7.5) in the prompt router significantly improves FID (10.22), CLIP (1.17), and CMMD (0.18) scores of the mixture. In addition, distillation can further improve the architecture search process of the experts, resulting in a more performant mixture. In summary, these results validate the effectiveness of our design choices for APTP.

Table 7.3: Ablation results of APTP's components on 30k samples from MS-COCO [3] validation set. We fine-tune all models for 10k iterations after pruning.

Method	MACs(@768)	Latency(@768)	FID (↓)	Clip Score (↑)	CMMD (↓)
Uni-Arch Baseline	1088.8G	3.1	46.56	29.11	0.91
Contrastive Router	1079.5G	3.1	48.78	28.90	0.92
+ Optimal Transport	1076.6G	3.1	38.56	30.07	0.74
+ Distillation (**APTP**)	1076.6G	3.1	25.57	31.13	0.58

In our second ablation experiment, we explore the impact of the number of experts on APTP. We prune SD V2.1 to 80% MACs budget using APTP with 4, 8, and 12 experts on MS-COCO. Fig. 7.5 shows the results. Interestingly, the results demonstrate that the relationship between FID and CLIP scores with the number of experts is nonlinear, and the optimal number of experts is dataset-dependent. This observation illustrates that prompt-based pruning is more suitable than static pruning for T2I models.

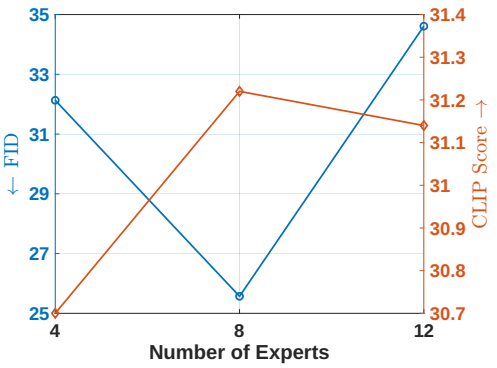


Figure 7.5: Ablation Results for the number of experts of APTP on MS-COCO.

7.5 Chapter Summary and Conclusions

In this chapter, we developed Adaptive Prompt-Tailored Pruning (APTP), the first prompt-based pruning method for text-to-image (T2I) diffusion models. APTP takes a T2I model, pretrained on large-scale data, and prunes it using a *target* dataset, resembling the common practice that organizations fine-tune T2I models on their internal data before deployment. The core element of APTP is a prompt router module that learns to decide the model capacity required to generate a sample for an input prompt, routing it to an architecture code given a desired compute budget. Each architecture code corresponds to a sub-network of the T2I model, specializing in generating images for the prompts that prompt router routes to it. APTP trains the prompt router and architecture codes in an end-to-end manner, encouraging the prompt router to route similar prompts to similar architecture codes. Further, we utilize optimal transport in the prompt router of APTP during pruning to diversify the architecture codes. Our experiments in which we prune Stable Diffusion (SD) V2.1 using CC3M and MS-COCO as *target* datasets demonstrate the benefit of prompt-based pruning compared to conventional static pruning methods for T2I models. Further, our analysis on APTP's prompt router reveals that it can automatically discover challenging prompt types for SD, like generating text, humans, and fingers, routing them to experts with high compute budgets.

Supplementary Materials for Chapter 7

S7.1 Overview of Diffusion Models

Given a random variable $\mathbf{x_0} \sim \mathcal{P}$, the goal of diffusion models [11, 272] is to model the underlying distribution $\mathcal{P}$ using a training set $\mathcal{D} = \{x_0\}$ of samples. To do so, first, diffusion models define a forward process parameterized by t in which they gradually perturb each sample x_0 with Gaussian noise with the variance schedule of β_t:

$$q(x_t|x_{t-1}) = \mathcal{N}(x_t; \sqrt{1-\beta_t}x_{t-1}, \beta_t I) \tag{7.16}$$

where $t \in [1, T]$. Thus, $q(x_t|x_0)$ has a Gaussian form:

$$q(x_t|x_0) = \mathcal{N}(x_t; \sqrt{\bar{\alpha}_t}x_0, (1-\bar{\alpha}_t)I) \tag{7.17}$$

where $\alpha_t = 1 - \beta_t$ and $\bar{\alpha}_t = \prod_{i=1}^{t} \alpha_i$. The noise schedule β_t is usually selected [11] such that $q(x_T) \to \mathcal{N}(0, I)$. Assuming β_t is small, diffusion models approximate the denoising distribution $q(x_{t-1}|x_t)$ by a parameterized Gaussian distribution $p_\theta(x_{t-1}|x_t) = \mathcal{N}(x_{t-1}; \frac{1}{\sqrt{\alpha_t}}(x_t - \frac{\beta_t}{\sqrt{1-\bar{\alpha}_t}}\epsilon_\theta(x_t, t)), \sigma_t^2 I)$, and σ_t^2 is often set to β_t. Diffusion models implement $\epsilon_\theta(.)$ with a neural network called the denoising model and train it with the variational evidence lower bound (ELBO) objective [11]:

$$\mathcal{L}_{\text{DDPM}}(\theta) = \mathbb{E}_{\substack{t\sim[1,T] \\ \epsilon\sim\mathcal{N}(0,I) \\ x_t\sim q(x_t|x_0)}} ||\epsilon(x_t, t; \theta) - \epsilon||^2 \tag{7.18}$$

Similarly, T2I diffusion models train a denoising model using pairs of image and text

prompts $(x_0, p) \sim \mathcal{P}$ to model the distribution $\mathcal{P}(\mathbf{x}_0|\mathbf{p})$ of images given an input text prompt:

$$\mathcal{L}_{\text{DDPM}}(\theta) = \mathbb{E}_{\substack{(x_0,p)\sim\mathcal{P} \\ t\sim[1,T] \\ \epsilon\sim\mathcal{N}(0,I) \\ x_t\sim q(x_t|x_0)}} ||\epsilon(x_t, p, t; \theta) - \epsilon||^2 \tag{7.19}$$

T2I Diffusion models generate a new sample by sampling an initial noise from $x_T \sim p(x_T) = \mathcal{N}(0, I)$ and iteratively denoising it using the denoising model by sampling from $p_\theta(x_{t-1}|x_t, p)$. Thus, the sampling process requires T sequential forward calculation of the denoising model, making it a slow and costly process.

S7.2 More Details of APTP

In this section we provide more details of the architecture predictor, the prompt router module, and our pruning method.

S7.2.1 Architecture Predictor

$f_{\text{AP}}(\cdot)$ in Eq. 7.4 is the architecture predictor. The architecture predictor changes the dimensionality of z to the dimensionality of e and codes a, which is the number of prunable width and depth units in the T2I model. We implement the f_{AP} function with a single feed-forward layer. Our prompt encoder, Sentence Transformer [12], has an output dimension of **768**. The total number of prunable units in SD 2.1 is **1620**, so the architecture predictor has an input dimension of **768** and an output dimension of **1620** in all of our experiments.

S7.2.2 Router Module Definition

During the pruning process, we use the assignment matrix Q^* calculated by optimal transport (Eq. 7.7) to route the architecture embeddings e to the architecture codes a. Thus, during the pruning process, the definition of function f_{R} is as follows:

$$I = \texttt{argmax}(BQ^*, \texttt{dim} = 0) \tag{7.20}$$

$$f_{\mathrm{R}}(e,\ \mathcal{A},\ Q^*) = A[:, I] \tag{7.21}$$

where e represents the architecture embeddings in a pruning batch, $\mathcal{A}$ is the set of architecture codes, Q^* is the calculated optimal assignment matrix, and A is the matrix of architecture codes.

At test time, the router function routes an input architecture embedding e to the most similar trained architecture code:

$$f_{\mathrm{R}}(e, \mathcal{A}) = \mathrm{argmax}_{a \in \mathcal{A}} \frac{e^T a}{||e||||a||} \tag{7.22}$$

S7.2.3 Pruning Depth

We apply the vectors $\mathbf{u}^{(i)} = [\mathbf{u}_j^{(i)}]_{j=1}^{M}$ (Eq. 7.9) to prune the model's depth with M being the total number of layers we prune. As the U-Net [295] architecture of T2I models has skip connections, we prune the depth layers in the encoder and decoder sides separately.

Pruning Depth in Encoder. We prune the j-*th* depth layer f_j in the encoder by applying the vector $u_j^{(i)}$ with the following formulation:

$$\widehat{\mathcal{F}}_j = \mathbf{u}_j^{(i)} f_j(\mathcal{F}_{j-1}) + (1 - \mathbf{u}_j^{(i)})\mathcal{F}_{j-1} \tag{7.23}$$

where $\mathcal{F}_{j-1}$ is the feature maps of the previous layer. When $u_j^{(i)}$ is close to one/zero, the j-th layer will be kept/pruned.

Pruning Depth in Decoder. In the decoder, the input of each layer is a concatenation of feature maps of the previous layer and feature maps of its corresponding encoder layer from the skip connection $\mathcal{F}_{j,skip}$. Thus, we prune the j-th depth layer f_j in the decoder as:

$$\widehat{\mathcal{F}}_j = \mathbf{u}_j^{(i)} f_j(\mathcal{F}_{j-1}||\mathcal{F}_{j,skip}) + (1 - \mathbf{u}_j^{(i)})\mathcal{F}_{j-1} \tag{7.24}$$

S7.2.4 Distillation Objective

Assuming $B^{(i)}$ samples get routed to the architecture code $a^{(i)}$ in a training batch ($\sum_i B^{(i)} = B$), we train the prompt router and the architecture codes using the following objective:

$$\begin{aligned}\min_{\eta,\mathcal{A}} \mathcal{L} =& [\frac{1}{N}\sum_{i=1}^{N}[\frac{1}{B^{(i)}}\sum_{j=1}^{B^{(i)}}[\mathcal{L}_{\text{DDPM}}(x_j^{(i)}, p_j^{(i)}; a^{(i)}) + \lambda_{\text{distill}}\mathcal{L}_{\text{distill}}(x_j^{(i)}, p_j^{(i)}; a^{(i)})]]] \\ &+ \lambda_{\text{res}}\mathcal{R}(\widehat{T}(\mathcal{A}), T_d) + \lambda_{\text{cont}}\mathcal{L}_{\text{cont}}(\eta)\end{aligned} \tag{7.25}$$

$\mathcal{L}_{\text{DDPM}}(x_j^{(i)}, p_j^{(i)}; a^{(i)})$ denotes the denoising objective for the sample $(x_j^{(i)}, p_j^{(i)})$ routed to the sub-network chosen by the architecture code $a^{(i)}$ (Eq. 7.19). $\mathcal{R}(\widehat{T}(\mathcal{A}), T_d)$ regularizes the weighted average of the MACs used by architecture codes ($\widehat{T}(\mathcal{A}) = \sum_i \frac{B^{(i)}}{B}\widehat{T}(a^{(i)})$) to be close to T_d. We define

$$\mathcal{R}(x, y) = \log(\max(x, y)/\min(x, y)) \tag{7.26}$$

as it can keep the resource usage close to the target value. $\mathcal{L}_{\text{cont}}$, given by Eq. 7.13, is the contrastive loss that guides the architecture predictor to map prompt representations to the regions of the space of the architecture embeddings such that their corresponding architecture vectors maintain the similarity between the prompts. Finally, $\mathcal{L}_{\text{distill}}$ is the distillation objective [374], regularizing the pruned model to have similar outputs to the original one. We do distillation at two levels, output level and block level. The output level distillation objective is:

$$\mathcal{L}_{\text{output-distill}}(x_j^{(i)}, p_j^{(i)}; a^{(i)}) = \mathbb{E}_{\substack{t\sim[1,T] \\ \epsilon\sim\mathcal{N}(0,I) \\ x_{j,t}^{(i)}\sim q(x_t|x_j^{(i)})}} [||\epsilon_{\text{Teacher}}(x_{j,t}^{(i)}, p_j^{(i)}, t; \theta) - \epsilon_{\text{Sub-Net}}(x_{j,t}^{(i)}, p_j^{(i)}, t; \theta, a^{(i)})||^2] \tag{7.27}$$

Here, $\epsilon_{\text{Teacher}}(x_{j,t}^{(i)}, p_j^{(i)}, t; \theta)$ denotes the original model's output and $\epsilon_{\text{Sub-Net}}(x_{j,t}^{(i)}, p_j^{(i)}, t; \theta, a^{(i)})$ denotes the output of the sub-network chosen by the architecture code $a^{(i)}$. The way we prune the U-Net preserves the output shape of each block. Doing so enables us to do distillation at block level as well and regularize the sub-network to match the output of the original model at each block. The block-level distillation objective is:

$$\mathcal{L}_{\text{block-distill}}(x_j^{(i)}, p_j^{(i)}; a^{(i)}) = \mathbb{E}_{\substack{t\sim[1,T] \\ \epsilon\sim\mathcal{N}(0,I) \\ x_{j,t}^{(i)}\sim q(x_t|x_j^{(i)})}} [\sum_b ||\epsilon_{\text{Teacher}}^b(x_{j,t}^{(i)}, p_j^{(i)}, t; \theta) - \epsilon_{\text{Sub-Net}}^b(x_{j,t}^{(i)}, p_j^{(i)}, t; \theta, a^{(i)})||^2] \tag{7.28}$$

Here, $\epsilon^b_{\text{Teacher}}$ and $\epsilon^b_{\text{Sub-Net}}$ denote the outputs of block b of the original model and the chosen sub-network of it, respectively. The total distillation loss $\mathcal{L}_{\text{distill}}$ is simply the sum of the output-level loss and the block-level distillation loss:

$$\mathcal{L}_{\text{distill}} = \mathcal{L}_{\text{output-distill}} + \mathcal{L}_{\text{block-distill}} \tag{7.29}$$

S7.2.5 Fine-tuning

The finetuning loss is a weighted average of the original DDPM objective (Eq. 7.19) and the distillation loss term (Eq. 7.29):

$$\mathcal{L}_{\text{finetuning}} = \alpha_{\text{DDPM}}\mathcal{L}_{\text{DDPM}} + \alpha_{\text{distill}}\mathcal{L}_{\text{distill}} \tag{7.30}$$

S7.3 Experiments

S7.3.1 Models and Datasets

We use two datasets as our *target* datasets: Conceptual Captions 3M (CC3M) [13] and MS-COCO Captions 2014 [3] with approximately 2.5M and 400K image-caption pairs, respectively. We apply APTP to the Stable Diffusion 2.1 (SD 2.1) [1] model. On CC3M, we prune SD2.1 with two settings: Base (0.85 budget, 16 experts) and Small (0.66 budget, 8 experts). Similarly, for COCO, we have two settings: Base (0.78 budget, 8 experts) and Small (0.64 budget, 8 experts).

S7.3.2 Experimental Settings

We train at a fixed resolution of 256×256 across all settings. During pruning, we first train the architecture predictor for 500 iterations as a warm-up phase. During this warm-up phase, we directly use its predicted architectures for pruning. Then, we start architecture codes and train the architecture predictor jointly with the codes for an additional 2500 iterations. We use the AdamW [375] optimizer and a constant learning rate of 0.0002 for both modules, with a 100-iteration linear warm-up. The effective pruning batch size is 1024, achieved by training on 16 NVIDIA A100 GPUs with a local batch size of 64. The temperature of the Gumbel-Sigmoid reparametrization (Eq. 7.9) is set to $\gamma = 0.4$. We set the regularization strength of the optimal transport objective (Eq. 7.5) to $\epsilon = 0.05$. We use 3 iterations of the Sinkhorn-Knopp algorithm [366] to solve the optimal transport problem [365]. We set the contrastive loss temperature τ to 0.03. The total pruning loss is the weighted average of DDPM loss, distillation loss, resource loss, and contrastive loss (see Eq. 7.15) with weights $\lambda_{\text{distill}} = 0.2$, $\lambda_{\text{res}} = 2.0$, and $\lambda_{\text{cont}} = 100.0$. After the pruning phase, we fine-tune the experts with the prompts assigned to them for 30,000 iterations using the AdamW optimizer, a fixed learning rate of 0.00001, and a batch size of 128. Upon experiments, we observed that higher weights of the DDPM loss result in unstable fine-tuning and slow convergence. As a result, we set the DDPM loss weight in the fine-tuning loss (Eq. 7.30) α_{DDPM} to 0.0001. We set $\alpha_{\text{distill}} = 1.0$. For sample generation, we use the classifier-free guidance [376] technique with the scale of 7.5 and 25 steps of the PNDM sampler [373].

S7.3.3 Evaluation

For quantitative evaluation of models pruned on CC3M, we use its validation dataset of approximately 14k samples. For COCO, we sample 30k captions of unique images from its 2014 validation dataset. We report Fréchet inception distance (FID) [370], CLIP score [371], and Maximum Mean Discrepancy with CLIP Embeddings (CMMD) [372] for APTP, the baselines, and SD 2.1 itself.

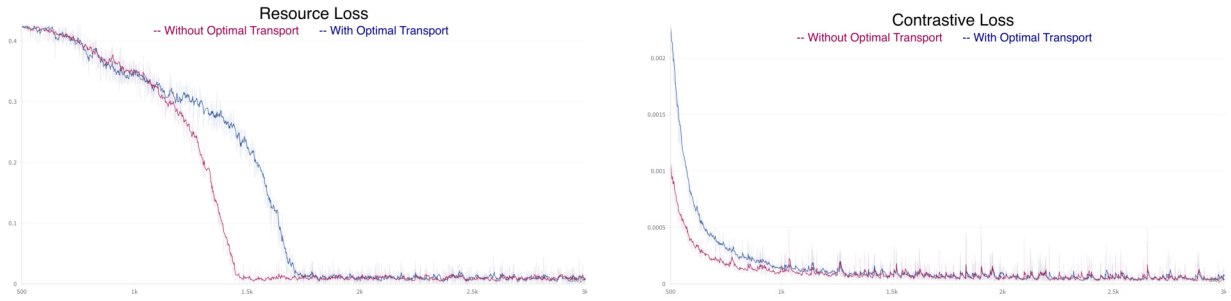
S7.3.4 Results

S7.3.4.1 Training Loss

Introduced in section 7.3.3.1, the resource loss regularizes the weighted average of the MACs used by architecture codes ($\widehat{T}(\mathcal{A}) = \sum_i \frac{B^{(i)}}{B}[\widehat{T}(a^{(i)})]$) to be close to T_d. We define resource loss as:

$$\mathcal{R}(x, y) = \log(\max(x, y)/\min(x, y)) \tag{7.31}$$

Fig. 7.6a illustrates the resource loss when applying APTP-Base to prune Stable Diffusion 2.1 using the COCO dataset as the target. This is shown under two conditions: with and without optimal transport following the initial warm-up phase (refer to S7.3 for details). APTP effectively regularizes the model so average MACs of the architecture codes is very close to the target budget. Fig. 7.6b presents the contrastive loss (Eq. 7.13) under the same conditions. APTP maps the prompt embeddings to regions of architecture embeddings such that their corresponding architectures maintain the similarity between prompts.



(a) Resource loss.

(b) Contrastive loss.

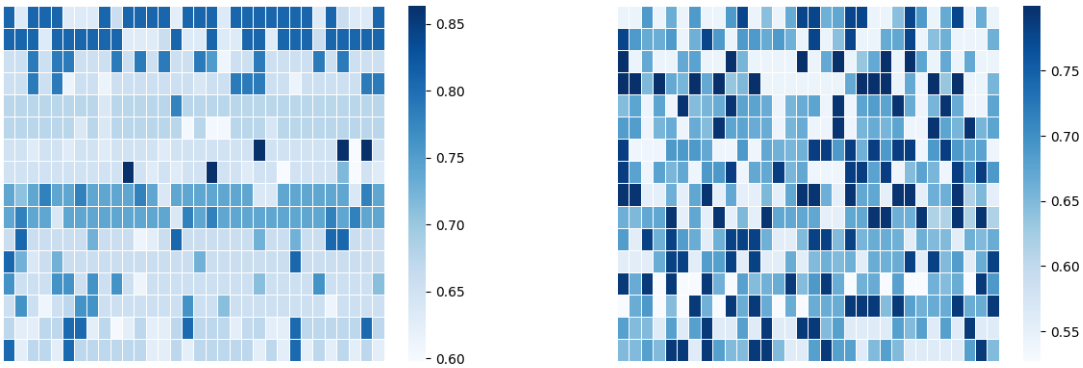
Figure 7.6: Resource and Contrastive loss observed when applying APTP-Base with a MAC budget of 0.77 to prune Stable Diffusion 2.1 using the COCO dataset. The comparison is made between two settings: with and without optimal transport. APTP both adheres to the target MAC budget and finds architecture vectors that maintain the similarity between the prompts.

S7.3.4.2 Visualization of the Impact of Optimal Transport

Fig. 7.7 displays the assignment of prompts to experts with and without optimal transport. By adding optimal transport, the assignment becomes more diverse, ensuring all experts get enough samples. This results in a significant improvement of performance metrics (See Table 7.3). Fig. 7.8 shows the distribution of the number of training CC3M samples mapped to each expert of APTP-Base.

S7.3.4.3 Analysis of Prompt Router on COCO

Table 7.4 demonstrates the most frequent words in the prompts assigned to each expert of APTP-Base pruned on COCO, revealing distinct topics and effective specialization. For example, Expert 1 specializes in indoor scences, Expert 5 in wildlife, and Expert 6 in urban scences. Expert 8 which has the highest resource utilization, focuses on images of human beings and hands, an observation consistent with our prompt analysis on CC3M (Table 7.2). Hands are another category that have been found to be challenging for SD



(a) Without Optimal Transport (b) With Optimal Transport

Figure 7.7: Comparison of sample assignments in a batch to experts with and without optimal transport. The incorporation of optimal transport results in a more diverse assignment pattern. In the figure, each square represents a prompt within the batch, and the color signifies the budget level of the expert assigned to the prompt. Higher-resource experts are indicated by darker blue.

2.1 [377].

Table 7.4: The most frequent words in prompts assigned to each expert of APTP-Base pruned on COCO. The resource utilization of each expert is indicated in parentheses.

Expert 1 (0.65, Indoor Scenes and Dining)	**Expert 2 (0.77, Food and Small Groups)**
table - plate - kitchen - sitting	food - pizza - sandwich
Expert 3 (0.78, People and Objects)	**Expert 4 (0.79, Sports and Activities)**
skateboard - surfboard - laptop - tie - phone	tennis - baseball - racquet - skateboard - skis
Expert 5 (0.79, Wildlife and Nature)	**Expert 6 (0.80, Urban Scenes and Transportation)**
giraffe - herd - sheep - zebra - elephants	street - train - bus - park - building
Expert 7 (0.81, Outdoor Activities and Nature)	**Expert 8 (0.83, Domestic Life and Pets)**
beach - ocean - surfboard - kite - wave	man - woman - girl - hand - bed - cat

S7.3.4.4 Quantitative Results

The quantitative results on the CC3M and MS-COCO datasets (Table 7.5) indicate that APTP significantly outperforms the baseline Norm method in terms of FID and CLIP scores while also reducing complexity. For both datasets, APTP at 30k and 50k fine-tuning

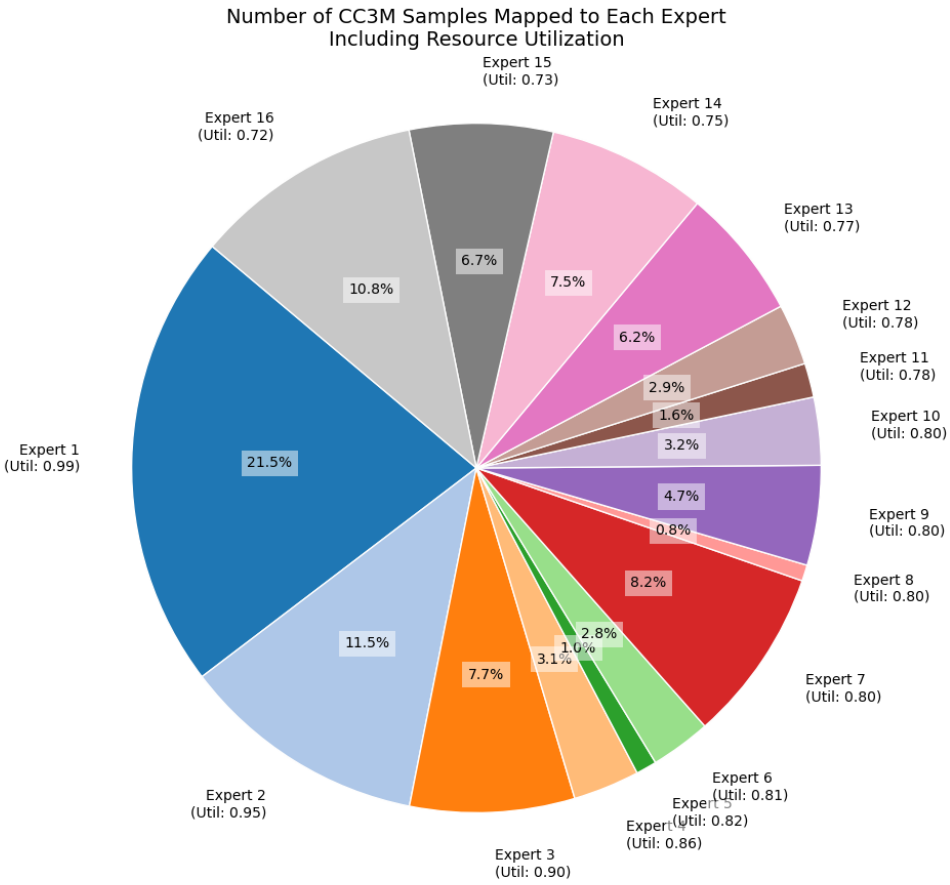


Figure 7.8: Distribution of CC3M Samples Mapped to Each Expert of APTP-Base, Including Resource Utilization Ratios

Table 7.5: Quantitative results on CC3M and MS-COCO. We report the performance metrics using samples generated at the resolution of 768 and downsampled to 256 [2]. We measure models' MACs and Latency with the input resolution of 768 on an A100 GPU. @30/50k shows the model's fine-tuning iterations after pruning.

CC3M					
Method	Complexity		Performance		
	MACs (@768)	Latency (↓) (Sec/Sample) (@768)	FID (↓)	CLIP (↑)	CMMD (↓)
Norm [310] @30k	1185.3G	3.4	157.51	26.23	1.778
Norm [310] @50k	1185.3G	3.4	141.04	26.51	1.646
APTP(0.66) @30k	916.3G	2.6	60.04	28.64	1.094
APTP(0.66) @50k	916.3G	2.6	54.95	29.08	1.017
APTP(0.85) @30k	1182.8G	3.4	36.77	30.84	0.675
APTP(0.85) @50k	1182.8G	3.4	36.09	30.90	0.669
SD 2.1	1384.2G	4.0	32.08	31.12	0.567

(a)

MS-COCO					
Method	Complexity		Performance		
	MACs (@768)	Latency (↓) (Sec/Sample) (@768)	FID (↓)	CLIP (↑)	CMMD (↓)
Norm [310] @30k	1077.4G	3.1	60.42	27.06	1.524
Norm [310] @50k	1077.4G	3.1	47.35	28.51	1.136
APTP(0.64) @30k	890.0G	2.5	39.12	29.98	0.867
APTP(0.64) @50k	890.0G	2.5	36.17	30.21	0.739
APTP(0.78) @30k	1076.6G	3.1	22.60	31.32	0.569
APTP(0.78) @50k	1076.6G	3.1	22.26	31.38	0.561
SD 2.1	1384.2G	4.0	15.47	31.33	0.500

(b)

iterations shows lower FID and CMMD values and higher CLIP scores compared to the baseline, demonstrating the advantage of prompt based pruning compared to static pruning ideas for T2I models.

S7.3.4.5 Expert Architectures

Figures 7.9 and 7.10 display the block-level U-Net architecture of all experts of APTP-Base, with CC3M and COCO as the target datasets, respectively. The retained MAC patterns are markedly different, further corroborating that different datasets require distinct architectures and that a 'one-size-fits-all' approach is not well-suited for T2I models. For CC3M, down-sampling blocks generally are retained with higher ratios. Overall, APTP tends to prune Resnet Blocks more frequently and aggressively compared to Attention Blocks.

S7.3.4.6 Samples of APTP

Table 7.6 displays the prompts for the images in Fig. 7.3 from CC3M and COCO validation sets.

We also provide more samples from APTP-Base pruned on CC3M and COCO. Fig. 7.11 presents samples from the validation set of CC3M generated by each 16 experts of APTP-Base at 0.85 MACs budget. Fig. 7.12 shows samples from the validation set of COCO from each of the 8 experts of APTP-Base pruned to 0.78 MACs budget.

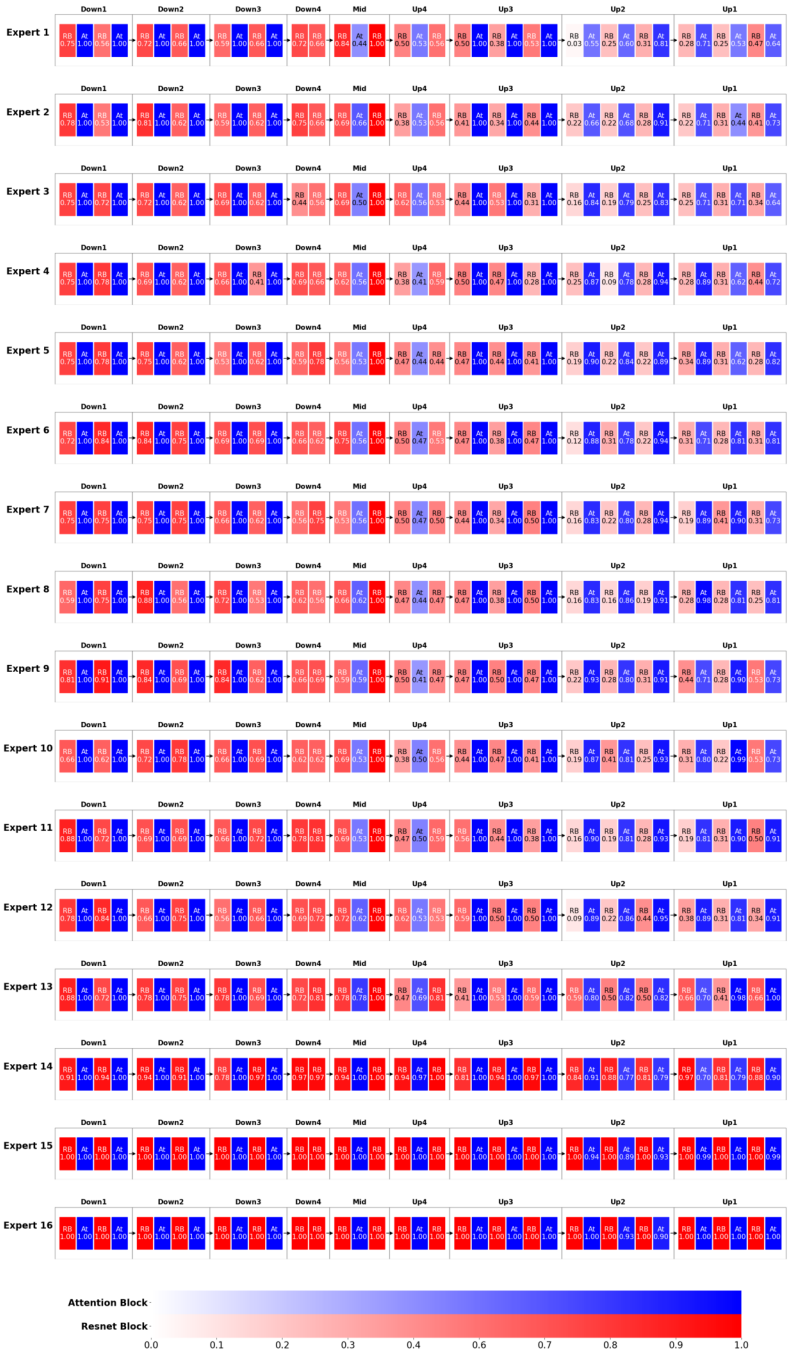


Figure 7.9: The block-level retained MAC ratio of the UNet architecture of all experts of APTP-Base applied to Stable Diffusion 2.1 with CC3M as the target dataset.

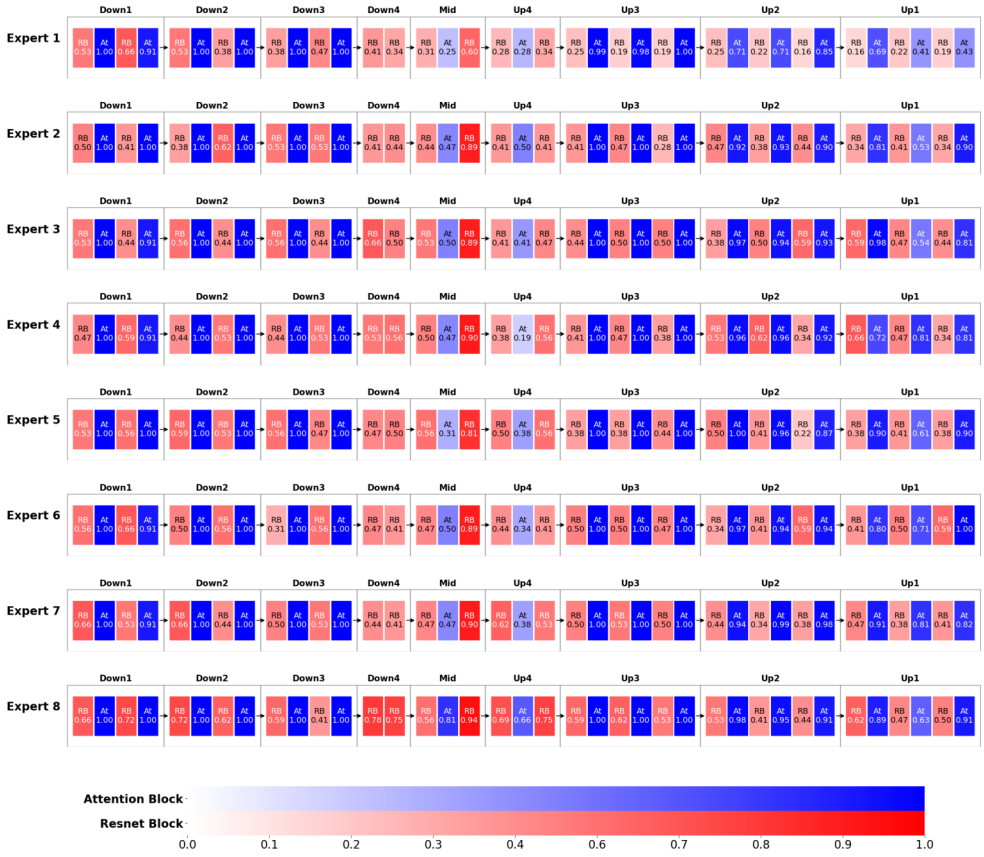


Figure 7.10: The block-level retained MAC ratio of the UNet architecture of all experts of APTP-Base applied to Stable Diffusion 2.1 with COCO as the target dataset. The groups of ResBlocks and the heads of Attention Blocks are pruned based on the outputs of the architecture predictor. The intensity of the color of each block represents the resource utilization of it. The number in each block indicates the precise ratio of retained MACs of the block. Conv_in, Conv_out, and skip connections between corresponding down and up blocks are omitted for brevity.

Figure 7.11: Samples of the APTP-Base experts after pruning the Stable Diffusion V2.1 using CC3M [13] as the *target* dataset. Each row corresponds to a unique expert. Please refer to Table 7.2 for the groups of prompts assigned to each expert.

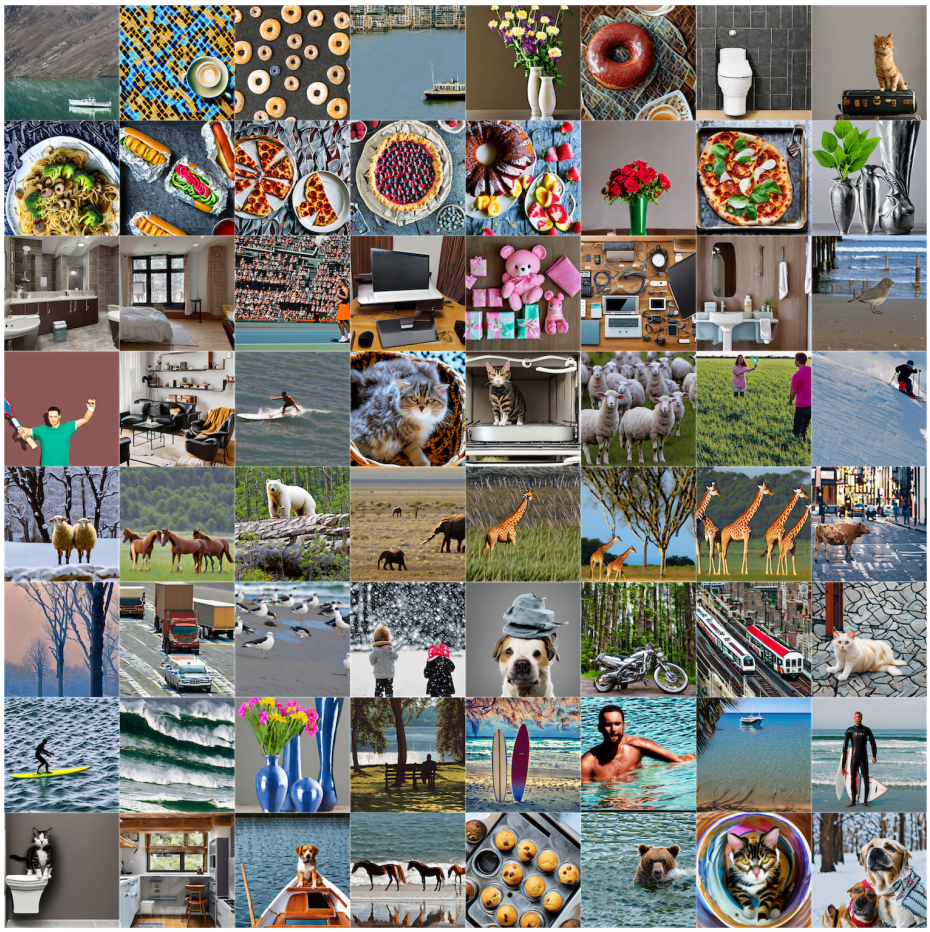


Figure 7.12: Samples of the APTP-Base experts after pruning the Stable Diffusion V2.1 using MS-COCO [3] as the *target* dataset. Each row corresponds to a unique expert. Please refer to Table 7.4 for the groups of prompts assigned to each expert.

CC3M Prompts
Saw this beautiful sky on my way home.
A silhouetted palm tree with boat tied to it rests on a beach that a man walks across during golden hour.
Sketch of a retro photo camera drawn by hand on a white background.
From left: person person, the dress on display.
Never saw a doll like this before but she sure is sweet looking.
The team on the summit.
A water drop falls towards a splash already made by another water drop.
Husky dog in a new year's interior.
Old paper with a picture of flowers, ranked in a moist environment.
People on new year's eve!
I drive over stuff - a pretty cool jeep, 4x4, or truck t-shirt.
The crowds arrive for day of festival.
A scary abandoned house under a starry sky.
Freehand fashion illustration with a lady with decorative hair.
Introduce some new flavors to your favorite finger food with these inspired chicken wings.
Gloomy face of a sad woman looking down, zoom in, gray background.
COCO Prompts
A white plate topped with a piece of chocolate covered cake.
Decorated coffee cup and knife sitting on a patterned surface.
A desk topped with a laptop computer and speakers.
A man sitting on the beach behind his surfboard.
A pizza type dish with vegetables and a lemon wedge.
The browned cracked crust of a baked berry pie.
A tennis player in an orange skirt walks off the court.
The skier in the helmet moves through thick snow.
A giraffe walks leisurely through the tall grass.
Several brown horses are standing in a field.
A red fire hydrant on a concrete block.
A bus driving in a city area with traffic signs.
There is a man on a surf board in the ocean.
A boat parked on top of a beach in crystal blue water.
A small kitchen with low a ceiling.
A calico cat curls up inside a bowl to sleep.

Table 7.6: Prompts for Fig. 7.3

Chapter 8: Conclusion and Discussion

In this thesis, we introduced innovative methods to enhance the efficiency and performance of deep learning models for visual recognition and generative modeling tasks through pruning and architecture search. The unifying theme of the proposed ideas is our differentiable pruning framework, where one or more selection modules, implemented as neural networks, learn to determine the optimal structure of a target deep model given a computational budget constraint. Except in Chapter 4, we leveraged the Gumbel-Softmax trick [368] and the Straight-Through Estimator [333] to make the selection operation in pruning and architecture search differentiable, enabling gradient-based optimization to train the selector module. This approach significantly reduces the computational burden and manual effort compared to traditional discrete neural architecture search and architecture design methods. Due to the inherent differences between visual recognition and generative modeling tasks, we structured this dissertation in two high-level parts, adapting our differentiable pruning scheme to each domain while taking their unique characteristics and requirements into account.

In part I, we design model pruning techniques for CNN classifiers for visual recognition. In Chapter 2, we proposed to prune a pre-trained CNN classifier by leveraging the interpretations of its decisions. We introduced a novel Amortized Explanation Model

(AEM) that provides real-time smooth saliency maps for the classifier's predictions. Then, we regularized the pruned model to maintain similar saliency maps to the original one. Our experiments verified that interpretations of a classifier contain valuable information for pruning, complementary to its outputs and weights that previous methods used for pruning.

We then addressed the problem of designing kernel sizes for CNNs in Chapter 3. We developed a size predictor and a kernel predictor models that we collaboratively train them to determine the kernel sizes and their weights in a differentiable manner, given a desired parameter budget for the model. Our experiments showed that our method can achieve both significant speed up for the search process and better final accuracy than the baselines. Therefore, our method provides and effective and efficient approach for the kernel size learning problem.

In Chapter 4, we proposed a method to reduce the complexity of the pruning process for CNNs. Specifically, model pruning methods follow a three-step process of pre-training, pruning, and fine-tuning to prune a given architecture. Our method accomplishes the first two steps by jointly training a reinforcement learning agent and the model's weights. The agent learns to determine the proper sub-network of the model during the pre-training phase, thereby improving the efficiency of the pruning process while achieving a competitive final pruned model's performance. As the model's weights evolve during pre-training, we introduced a mechanism to model the changing dynamics of the reward function for the agent during training. Doing so enabled us to train the agent using its episodic observations during the pre-training phase.

In part II, we developed model pruning techniques for deep visual generative models.

Specifically, we focused on inference efficiency of Generative Adversarial Networks (GANs) as well as diffusion models and introduced methods tailored for their characteristics. In Chapter 5, we pruned conditional image-to-image translation GANs using local density structures on their manifold. Additionally, we introduced a framework to prune both the generator and discriminator using our pruning agents in a collaborative manner. The pruning agents exchange feedback during pruning, thereby alleviating mode collapse and improving the stability of the pruning process. Our experiments showed that our method can reduce the MACs of a pretrained Pix2Pix or Cycle-GAN model by 80% or more on the benchmarks while almost recovering their performance or even outperforming them.

We then proposed a structural pruning method for diffusion models in Chapter 6. We leveraged the gradual sampling process of diffusion models to prune a pre-trained model into a mixture of efficient experts. Inspired by the previous empirical findings indicating that different stages of the denoising process have varying roles, our pruning method provides a mixture of experts, each handling a distinct part of the denoising process while having a specialized architecture. We train an Expert Routing Agent (ERA) that automatically allocates resources between experts in a differentiable manner, without requiring manual heuristics. Our experiments illustrated that our mixture of experts achieves better quality and sampling throughput than employing a single pruned model for all of the denoising process.

Finally, we developed a prompt-based pruning technique for modern text-to-image diffusion models in Chapter 7. We built it upon the intuition that different prompts have varying architectural capacity requirements. Thus, we prune a pre-trained text-to-image diffusion model into a set of efficient experts such that each expert is specialized to generate

samples for the prompts routed to it. We train a prompt router module along with a set of architecture codes in an end-to-end manner to find the optimal configurations of the experts given a desired computational budget constraint for the mixture. We showed that our prompt-based pruning scheme is more suitable than static pruning baselines for text-to-image models, achieving significantly better sample quality metrics with similar or lower latency values.

8.1 Broader Context and Future Directions

The ideas presented in this thesis primarily focus on differentiable and efficient model pruning for various deep vision models, including those designed for visual recognition and generative modeling. We hope this work serves as a foundation for other researchers to develop more comprehensive and effective techniques in the following areas:

First, model pruning represents just one avenue within the broader context of architectural optimization and inference efficiency, which includes other directions such as quantization, neural architecture search (NAS), architecture design, and distillation. These approaches are often pursued independently and tailored to specific applications. For instance, quantization [378, 379] is the widely adopted method for improving the inference efficiency of large language models (LLMs), and distillation has been extensively applied to reduce the number of denoising steps in diffusion models [380].

As model sizes continue to scale into the tens of billions or even trillions of parameters [14, 15], a promising direction for future research lies in integrating different architectural optimization techniques in a unified framework. This thesis's contributions

to differentiable pruning can serve as inspirations toward such integration. For instance, one could design a selector module that simultaneously prunes a model's layers and determines their precision in an end-to-end manner, combining the strengths of pruning and quantization.

Second, in the domain of generative modeling, our work on pruning diffusion models introduces a mixture of experts specialized for specific denoising intervals or prompt types. Yet, further improvements can be achieved by incorporating weight sharing and model merging techniques. Currently, the experts in our framework have distinct architectures and are trained independently for their specialized tasks. Future research could explore methods to reduce the parameter count of the mixture of experts, either by introducing regularization terms to limit the overall number of parameters or by leveraging weight-sharing and model-merging strategies to minimize memory costs without compromising performance.

Third, on the visual recognition front, our framework could be extended to jointly prune both the vision and language backbones of modern vision-language models. These models hold immense potential for real-world applications such as robotics and autonomous vehicles, where computational efficiency is critical. By reducing the deployment cost of vision-language models, this line of research could unlock significant economic and societal benefits, paving the way for broader adoption in diverse industries.

Bibliography

[1] Robin Rombach, Andreas Blattmann, Dominik Lorenz, Patrick Esser, and Björn Ommer. High-resolution image synthesis with latent diffusion models. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 10684–10695, 2022.

[2] Bo-Kyeong Kim, Hyoung-Kyu Song, Thibault Castells, and Shinkook Choi. On architectural compression of text-to-image diffusion models. *arXiv preprint arXiv:2305.15798*, 2023.

[3] 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 *Computer Vision–ECCV 2014: 13th European Conference, Zurich, Switzerland, September 6-12, 2014, Proceedings, Part V 13*, pages 740–755. Springer, 2014.

[4] Neil Jethani, Mukund Sudarshan, Yindalon Aphinyanaphongs, and Rajesh Ranganath. Have we learned to explain?: How interpretability methods can learn to encode predictions in their interpretations. In *International Conference on Artificial Intelligence and Statistics*, pages 1459–1467. PMLR, 2021.

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

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

[7] Yogesh Balaji, Seungjun Nah, Xun Huang, Arash Vahdat, Jiaming Song, Karsten Kreis, Miika Aittala, Timo Aila, Samuli Laine, Bryan Catanzaro, et al. ediffi: Text-to-image diffusion models with an ensemble of expert denoisers. *arXiv preprint arXiv:2211.01324*, 2022.

[8] Zhida Feng, Zhenyu Zhang, Xintong Yu, Yewei Fang, Lanxin Li, Xuyi Chen, Yuxiang Lu, Jiaxiang Liu, Weichong Yin, Shikun Feng, et al. Ernie-vilg 2.0: Improving text-to-image diffusion model with knowledge-enhanced mixture-of-denoising-experts. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 10135–10145, 2023.

[9] Gongfan Fang, Xinyin Ma, and Xinchao Wang. Structural pruning for diffusion models. In *Advances in Neural Information Processing Systems*, 2023.

[10] Enshu Liu, Xuefei Ning, Zinan Lin, Huazhong Yang, and Yu Wang. Oms-dpm: Optimizing the model schedule for diffusion probabilistic models. In *International Conference on Machine Learning*, pages 21915–21936. PMLR, 2023.

[11] Jonathan Ho, Ajay Jain, and Pieter Abbeel. Denoising diffusion probabilistic models. *Advances in neural information processing systems*, 33:6840–6851, 2020.

[12] Nils Reimers and Iryna Gurevych. Sentence-BERT: Sentence embeddings using Siamese BERT-networks. In Kentaro Inui, Jing Jiang, Vincent Ng, and Xiaojun Wan, editors, *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*, pages 3982–3992, Hong Kong, China, November 2019. Association for Computational Linguistics. doi: 10.18653/v1/D19-1410. URL `https://aclanthology.org/D19-1410`.

[13] Piyush Sharma, Nan Ding, Sebastian Goodman, and Radu Soricut. Conceptual captions: A cleaned, hypernymed, image alt-text dataset for automatic image captioning. In *Proceedings of ACL*, 2018.

[14] Josh Achiam, Steven Adler, Sandhini Agarwal, Lama Ahmad, Ilge Akkaya, Florencia Leoni Aleman, Diogo Almeida, Janko Altenschmidt, Sam Altman, Shyamal Anadkat, et al. Gpt-4 technical report. *arXiv preprint arXiv:2303.08774*, 2023.

[15] Gemini Team, Rohan Anil, Sebastian Borgeaud, Yonghui Wu, Jean-Baptiste Alayrac, Jiahui Yu, Radu Soricut, Johan Schalkwyk, Andrew M Dai, Anja Hauth, et al. Gemini: a family of highly capable multimodal models. *arXiv preprint arXiv:2312.11805*, 2023.

[16] Haotian Liu, Chunyuan Li, Qingyang Wu, and Yong Jae Lee. Visual instruction tuning. *Advances in neural information processing systems*, 36, 2024.

[17] James Betker, Gabriel Goh, Li Jing, Tim Brooks, Jianfeng Wang, Linjie Li, Long Ouyang, Juntang Zhuang, Joyce Lee, Yufei Guo, et al. Improving image generation with better captions. *Computer Science. https://cdn. openai. com/papers/dall-e-3. pdf*, 2(3):8, 2023.

[18] Chitwan Saharia, William Chan, Saurabh Saxena, Lala Li, Jay Whang, Emily L Denton, Kamyar Ghasemipour, Raphael Gontijo Lopes, Burcu Karagol Ayan, Tim Salimans, et al. Photorealistic text-to-image diffusion models with deep language understanding. *Advances in neural information processing systems*, 35:36479–36494, 2022.

[19] Adobe FireFly. Firefly, 2023. URL `https://www.adobe.com/sensei/generative-ai/firefly.html`.

[20] Haotian Liu, Chunyuan Li, Yuheng Li, and Yong Jae Lee. Improved baselines with visual instruction tuning. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 26296–26306, 2024.

[21] Peng Wang, Shuai Bai, Sinan Tan, Shijie Wang, Zhihao Fan, Jinze Bai, Keqin Chen, Xuejing Liu, Jialin Wang, Wenbin Ge, et al. Qwen2-vl: Enhancing vision-language model's perception of the world at any resolution. *arXiv preprint arXiv:2409.12191*, 2024.

[22] Dustin Podell, Zion English, Kyle Lacey, Andreas Blattmann, Tim Dockhorn, Jonas Müller, Joe Penna, and Robin Rombach. SDXL: Improving latent diffusion models for high-resolution image synthesis. In *The Twelfth International Conference on Learning Representations*, 2024. URL https://openreview.net/forum?id=di52zR8xgf.

[23] Patrick Esser, Sumith Kulal, Andreas Blattmann, Rahim Entezari, Jonas Müller, Harry Saini, Yam Levi, Dominik Lorenz, Axel Sauer, Frederic Boesel, et al. Scaling rectified flow transformers for high-resolution image synthesis. In *Forty-first International Conference on Machine Learning*, 2024.

[24] Sanjeev Arora, Rong Ge, Behnam Neyshabur, and Yi Zhang. Stronger generalization bounds for deep nets via a compression approach. In *International conference on machine learning*, pages 254–263. PMLR, 2018.

[25] Zeyuan Allen-Zhu, Yuanzhi Li, and Yingyu Liang. Learning and generalization in overparameterized neural networks, going beyond two layers. *Advances in neural information processing systems*, 32, 2019.

[26] Song Han, Xingyu Liu, Huizi Mao, Jing Pu, Ardavan Pedram, Mark A Horowitz, and William J Dally. Eie: Efficient inference engine on compressed deep neural network. *ACM SIGARCH Computer Architecture News*, 44(3):243–254, 2016.

[27] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Ł ukasz Kaiser, and Illia Polosukhin. Attention is all you need. In I. Guyon, U. Von Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 30. Curran Associates, Inc., 2017. URL https://proceedings.neurips.cc/paper_files/paper/2017/file/3f5ee243547dee91fbd053c1c4a845aa-Paper.pdf.

[28] Alexey Dosovitskiy, Lucas Beyer, Alexander Kolesnikov, Dirk Weissenborn, Xiaohua Zhai, Thomas Unterthiner, Mostafa Dehghani, Matthias Minderer, Georg Heigold, Sylvain Gelly, Jakob Uszkoreit, and Neil Houlsby. An image is worth 16x16 words: Transformers for image recognition at scale. In *International Conference on Learning Representations*, 2021. URL https://openreview.net/forum?id=YicbFdNTTy.

[29] Sanghyun Woo, Shoubhik Debnath, Ronghang Hu, Xinlei Chen, Zhuang Liu, In So Kweon, and Saining Xie. Convnext v2: Co-designing and scaling convnets with masked autoencoders. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 16133–16142, 2023.

[30] Kirill Vishniakov, Zhiqiang Shen, and Zhuang Liu. ConvNet vs transformer, supervised vs CLIP: Beyond ImageNet accuracy. In *Proceedings of the 41st International Conference on Machine Learning*, volume 235 of *Proceedings of Machine Learning Research*, pages 49545–49557. PMLR, 21–27 Jul 2024. URL `https://proceedings.mlr.press/v235/vishniakov24a.html`.

[31] Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. Imagenet: A large-scale hierarchical image database. In *Computer Vision and Pattern Recognition, 2009. CVPR 2009. IEEE Conference on*, pages 248–255. Ieee, 2009.

[32] Ross Girshick. Fast r-cnn. In *Proceedings of the IEEE international conference on computer vision*, pages 1440–1448, 2015.

[33] Jonathan Long, Evan Shelhamer, and Trevor Darrell. Fully convolutional networks for semantic segmentation. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 3431–3440, 2015.

[34] Karen Simonyan and Andrew Zisserman. Two-stream convolutional networks for action recognition in videos. In *Advances in neural information processing systems*, pages 568–576, 2014.

[35] Mariusz Bojarski, Davide Del Testa, Daniel Dworakowski, Bernhard Firner, Beat Flepp, Prasoon Goyal, Lawrence D Jackel, Mathew Monfort, Urs Muller, Jiakai Zhang, et al. End to end learning for self-driving cars. *arXiv preprint arXiv:1604.07316*, 2016.

[36] Joseph Redmon, Santosh Divvala, Ross Girshick, and Ali Farhadi. You only look once: Unified, real-time object detection. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 779–788, 2016.

[37] Joseph Redmon and Ali Farhadi. Yolov3: An incremental improvement. *arXiv preprint arXiv:1804.02767*, 2018.

[38] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. In Yoshua Bengio and Yann LeCun, editors, *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings*, 2015. URL `http://arxiv.org/abs/1409.1556`.

[39] 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.

[40] Gao Huang, Zhuang Liu, Laurens Van Der Maaten, and Kilian Q Weinberger. Densely connected convolutional networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 4700–4708, 2017.

[41] Song Han, Jeff Pool, John Tran, and William Dally. Learning both weights and connections for efficient neural network. In *Advances in neural information processing systems*, pages 1135–1143, 2015.

[42] Wenlin Chen, James Wilson, Stephen Tyree, Kilian Weinberger, and Yixin Chen. Compressing neural networks with the hashing trick. In *International conference on machine learning*, pages 2285–2294, 2015.

[43] Mohammad Rastegari, Vicente Ordonez, Joseph Redmon, and Ali Farhadi. Xnor-net: Imagenet classification using binary convolutional neural networks. In *European conference on computer vision*, pages 525–542. Springer, 2016.

[44] Hao Li, Asim Kadav, Igor Durdanovic, Hanan Samet, and Hans Peter Graf. Pruning filters for efficient convnets. *ICLR*, 2017.

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

[46] Yihui He, Ji Lin, Zhijian Liu, Hanrui Wang, Li-Jia Li, and Song Han. Amc: Automl for model compression and acceleration on mobile devices. In *Proceedings of the European Conference on Computer Vision (ECCV)*, pages 784–800, 2018.

[47] Mao Ye, Chengyue Gong, Lizhen Nie, Denny Zhou, Adam Klivans, and Qiang Liu. Good subnetworks provably exist: Pruning via greedy forward selection. *International Conference on Machine Learning*, 2020.

[48] Ting-Wu Chin, Ruizhou Ding, Cha Zhang, and Diana Marculescu. Towards efficient model compression via learned global ranking. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 1518–1528, 2020.

[49] Shangqian Gao, Feihu Huang, Jian Pei, and Heng Huang. Discrete model compression with resource constraint for deep neural networks. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 1899–1908, 2020.

[50] 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. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 3296–3305, 2019.

[51] Marco Tulio Ribeiro, Sameer Singh, and Carlos Guestrin. " why should i trust you?" explaining the predictions of any classifier. In *Proceedings of the 22nd ACM SIGKDD international conference on knowledge discovery and data mining*, pages 1135–1144, 2016.

[52] Scott M Lundberg and Su-In Lee. A unified approach to interpreting model predictions. In *Proceedings of the 31st international conference on neural information processing systems*, pages 4768–4777, 2017.

[53] Sara Hooker, Dumitru Erhan, Pieter-Jan Kindermans, and Been Kim. A benchmark for interpretability methods in deep neural networks. In H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 32. Curran Associates, Inc., 2019.

[54] Julius Adebayo, Justin Gilmer, Michael Muelly, Ian Goodfellow, Moritz Hardt, and Been Kim. Sanity checks for saliency maps. In S. Bengio, H. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 31. Curran Associates, Inc., 2018.

[55] 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.

[56] Avanti Shrikumar, Peyton Greenside, and Anshul Kundaje. Learning important features through propagating activation differences. In *International Conference on Machine Learning*, pages 3145–3153. PMLR, 2017.

[57] Suraj Srinivas and François Fleuret. Rethinking the role of gradient-based attribution methods for model interpretability. In *9th International Conference on Learning Representations, ICLR 2021, Virtual Event, Austria, May 3-7, 2021*. OpenReview.net, 2021. URL `https://openreview.net/forum?id=dYeAHXnpWJ4`.

[58] Will Grathwohl, Kuan-Chieh Wang, Jörn-Henrik Jacobsen, David Duvenaud, Mohammad Norouzi, and Kevin Swersky. Your classifier is secretly an energy based model and you should treat it like one. In *8th International Conference on Learning Representations, ICLR 2020, Addis Ababa, Ethiopia, April 26-30, 2020*. OpenReview.net, 2020. URL `https://openreview.net/forum?id=Hkxzx0NtDB`.

[59] Matthew D Zeiler and Rob Fergus. Visualizing and understanding convolutional networks. In *European conference on computer vision*, pages 818–833. Springer, 2014.

[60] Luisa M. Zintgraf, Taco S. Cohen, Tameem Adel, and Max Welling. Visualizing deep neural network decisions: Prediction difference analysis. In *5th International Conference on Learning Representations, ICLR 2017, Toulon, France, April 24-26, 2017, Conference Track Proceedings*. OpenReview.net, 2017. URL `https://openreview.net/forum?id=BJ5UeU9xx`.

[61] Jianbo Chen, Le Song, Martin Wainwright, and Michael Jordan. Learning to explain: An information-theoretic perspective on model interpretation. In *International Conference on Machine Learning*, pages 883–892. PMLR, 2018.

[62] Jinsung Yoon, James Jordon, and Mihaela van der Schaar. Invase: Instance-wise variable selection using neural networks. In *International Conference on Learning Representations*, 2018.

[63] Michael M. Bronstein, Joan Bruna, Taco Cohen, and Petar Velickovic. Geometric deep learning: Grids, groups, graphs, geodesics, and gauges. *CoRR*, abs/2104.13478, 2021. URL `https://arxiv.org/abs/2104.13478`.

[64] Alireza Ganjdanesh, Shangqian Gao, and Heng Huang. Interpretations steered network pruning via amortized inferred saliency maps. In *Computer Vision–ECCV 2022:*

17th European Conference, Tel Aviv, Israel, October 23–27, 2022, Proceedings, Part XXI, pages 278–296. Springer, 2022.

[65] Bolei Zhou, Aditya Khosla, Agata Lapedriza, Aude Oliva, and Antonio Torralba. Learning deep features for discriminative localization. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 2921–2929, 2016.

[66] Sebastian Bach, Alexander Binder, Grégoire Montavon, Frederick Klauschen, Klaus-Robert Müller, and Wojciech Samek. On pixel-wise explanations for non-linear classifier decisions by layer-wise relevance propagation. *PloS one*, 10(7):e0130140, 2015.

[67] Karen Simonyan, Andrea Vedaldi, and Andrew Zisserman. Deep inside convolutional networks: Visualising image classification models and saliency maps. In *In Workshop at International Conference on Learning Representations*. Citeseer, 2014.

[68] Jost Tobias Springenberg, Alexey Dosovitskiy, Thomas Brox, and Martin A. Riedmiller. Striving for simplicity: The all convolutional net. In Yoshua Bengio and Yann LeCun, editors, *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Workshop Track Proceedings*, 2015. URL `http://arxiv.org/abs/1412.6806`.

[69] Daniel Smilkov, Nikhil Thorat, Been Kim, Fernanda Viégas, and Martin Wattenberg. Smoothgrad: removing noise by adding noise. *arXiv preprint arXiv:1706.03825*, 2017.

[70] Harshay Shah, Prateek Jain, and Praneeth Netrapalli. Do input gradients highlight discriminative features? *Advances in Neural Information Processing Systems*, 34, 2021.

[71] Jian Zhou and Olga G Troyanskaya. Predicting effects of noncoding variants with deep learning–based sequence model. *Nature methods*, 12(10):931–934, 2015.

[72] Piotr Dabkowski and Yarin Gal. Real time image saliency for black box classifiers. In Isabelle Guyon, Ulrike von Luxburg, Samy Bengio, Hanna M. Wallach, Rob Fergus, S. V. N. Vishwanathan, and Roman Garnett, editors, *Advances in Neural Information Processing Systems 30: Annual Conference on Neural Information Processing Systems 2017, December 4-9, 2017, Long Beach, CA, USA*, pages 6967–6976, 2017.

[73] Wei Wen, Chunpeng Wu, Yandan Wang, Yiran Chen, and Hai Li. Learning structured sparsity in deep neural networks. In *Advances in neural information processing systems*, pages 2074–2082, 2016.

[74] Yang He, Ping Liu, Ziwei Wang, Zhilan Hu, and Yi Yang. Filter pruning via geometric median for deep convolutional neural networks acceleration. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 4340–4349, 2019.

[75] Pavlo Molchanov, Arun Mallya, Stephen Tyree, Iuri Frosio, and Jan Kautz. Importance estimation for neural network pruning. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 11264–11272, 2019.

[76] 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. In *Advances in Neural Information Processing Systems*, pages 875–886, 2018.

[77] Hanyu Peng, Jiaxiang Wu, Shifeng Chen, and Junzhou Huang. Collaborative channel pruning for deep networks. In *International Conference on Machine Learning*, pages 5113–5122, 2019.

[78] Mingbao Lin, Rongrong Ji, Yuxin Zhang, Baochang Zhang, Yongjian Wu, and Yonghong Tian. Channel pruning via automatic structure search. In *Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI)*, pages 673 – 679, 2020.

[79] Lucas Liebenwein, Cenk Baykal, Harry Lang, Dan Feldman, and Daniela Rus. Provable filter pruning for efficient neural networks. In *International Conference on Learning Representations*, 2020. URL `https://openreview.net/forum?id=BJxkOlSYDH`.

[80] Yanfu Zhang, Shangqian Gao, and Heng Huang. Exploration and estimation for model compression. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 487–496, 2021.

[81] Shangqian Gao, Feihu Huang, Yanfu Zhang, and Heng Huang. Disentangled differentiable network pruning. In *Proceedings of the European Conference on Computer Vision (ECCV)*, 2022.

[82] Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In Francis Bach and David Blei, editors, *Proceedings of the 32nd International Conference on Machine Learning*, volume 37 of *Proceedings of Machine Learning Research*, pages 448–456, Lille, France, 07–09 Jul 2015. PMLR. URL `https://proceedings.mlr.press/v37/ioffe15.html`.

[83] Zhuang Liu, Jianguo Li, Zhiqiang Shen, Gao Huang, Shoumeng Yan, and Changshui Zhang. Learning efficient convolutional networks through network slimming. In *ICCV*, 2017.

[84] Yanfu Zhang, Shangqian Gao, and Heng Huang. Recover fair deep classification models via altering pre-trained structure. In *Proceedings of the European Conference on Computer Vision (ECCV)*, 2022.

[85] Tailin Liang, John Glossner, Lei Wang, Shaobo Shi, and Xiaotong Zhang. Pruning and quantization for deep neural network acceleration: A survey. *Neurocomputing*, 461:370–403, 2021.

[86] Muhammad Sabih, Frank Hannig, and Juergen Teich. Utilizing explainable ai for quantization and pruning of deep neural networks. *arXiv preprint arXiv:2008.09072*, 2020.

[87] Seul-Ki Yeom, Philipp Seegerer, Sebastian Lapuschkin, Alexander Binder, Simon Wiedemann, Klaus-Robert Müller, and Wojciech Samek. Pruning by explaining: A novel criterion for deep neural network pruning. *Pattern Recognition*, 115:107899, 2021.

[88] Kaixuan Yao, Feilong Cao, Yee Leung, and Jiye Liang. Deep neural network compression through interpretability-based filter pruning. *Pattern Recognition*, 119:108056, 2021.

[89] Ali Alqahtani, Xianghua Xie, Mark W Jones, and Ehab Essa. Pruning cnn filters via quantifying the importance of deep visual representations. *Computer Vision and Image Understanding*, 208:103220, 2021.

[90] James M Joyce. Kullback-leibler divergence. *International encyclopedia of statistical science*, 720:722, 2011.

[91] Eric Jang, Shixiang Gu, and Ben Poole. Categorical reparameterization with gumbel-softmax. In *5th International Conference on Learning Representations, ICLR 2017, Toulon, France, April 24-26, 2017, Conference Track Proceedings*. OpenReview.net, 2017. URL `https://openreview.net/forum?id=rkE3y85ee`.

[92] Chris J. Maddison, Andriy Mnih, and Yee Whye Teh. The concrete distribution: A continuous relaxation of discrete random variables. In *5th International Conference on Learning Representations, ICLR 2017, Toulon, France, April 24-26, 2017, Conference Track Proceedings*. OpenReview.net, 2017. URL `https://openreview.net/forum?id=S1jE5L5gl`.

[93] Olaf Ronneberger, Philipp Fischer, and Thomas Brox. U-net: Convolutional networks for biomedical image segmentation. In *International Conference on Medical image computing and computer-assisted intervention*, pages 234–241. Springer, 2015.

[94] Minsoo Kang and Bohyung Han. Operation-aware soft channel pruning using differentiable masks. *International Conference on Machine Learning*, 2020.

[95] Yang He, Guoliang Kang, Xuanyi Dong, Yanwei Fu, and Yi Yang. Soft filter pruning for accelerating deep convolutional neural networks. In *International Joint Conference on Artificial Intelligence (IJCAI)*, pages 2234–2240, 2018.

[96] Yang He, Yuhang Ding, Ping Liu, Linchao Zhu, Hanwang Zhang, and Yi Yang. Learning filter pruning criteria for deep convolutional neural networks acceleration. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 2009–2018, 2020.

[97] Mingbao Lin, Rongrong Ji, Yan Wang, Yichen Zhang, Baochang Zhang, Yonghong Tian, and Ling Shao. Hrank: Filter pruning using high-rank feature map. *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2020.

[98] 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.

[99] Yuchao Li, Shaohui Lin, Jianzhuang Liu, Qixiang Ye, Mengdi Wang, Fei Chao, Fan Yang, Jincheng Ma, Qi Tian, and Rongrong Ji. Towards compact cnns via collaborative compression. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 6438–6447, 2021.

[100] Olga Russakovsky, Jia Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, Michael S. Bernstein, Alexander C. Berg, and Li Fei-Fei. Imagenet large scale visual recognition challenge. *Int. J. Comput. Vis.*, 115(3):211–252, 2015. doi: 10.1007/s11263-015-0816-y. URL `https://doi.org/10.1007/s11263-015-0816-y`.

[101] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, et al. Pytorch: An imperative style, high-performance deep learning library. In *Advances in Neural Information Processing Systems*, pages 8024–8035, 2019.

[102] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization. In Yoshua Bengio and Yann LeCun, editors, *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings*, 2015. URL `http://arxiv.org/abs/1412.6980`.

[103] Wenzhe Shi, Jose Caballero, Ferenc Huszár, Johannes Totz, Andrew P Aitken, Rob Bishop, Daniel Rueckert, and Zehan Wang. Real-time single image and video super-resolution using an efficient sub-pixel convolutional neural network. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 1874–1883, 2016.

[104] Vinod Nair and Geoffrey E. Hinton. Rectified linear units improve restricted boltzmann machines. In Johannes Fürnkranz and Thorsten Joachims, editors, *Proceedings of the 27th International Conference on Machine Learning (ICML-10), June 21-24, 2010, Haifa, Israel*, pages 807–814. Omnipress, 2010. URL `https://icml.cc/Conferences/2010/papers/432.pdf`.

[105] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*, pages 1097–1105, 2012.

[106] Christian Szegedy, Vincent Vanhoucke, Sergey Ioffe, Jon Shlens, and Zbigniew Wojna. Rethinking the inception architecture for computer vision. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 2818–2826, 2016.

[107] 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.

[108] Xiaohan Ding, Xiangyu Zhang, Jungong Han, and Guiguang Ding. Scaling up your kernels to 31x31: Revisiting large kernel design in cnns. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 11963–11975, 2022.

[109] Barret Zoph and Quoc V. Le. Neural architecture search with reinforcement learning. In *5th International Conference on Learning Representations, ICLR 2017, Toulon, France, April 24-26, 2017, Conference Track Proceedings*. OpenReview.net, 2017. URL `https://openreview.net/forum?id=r1Ue8Hcxg`.

[110] Piotr Dollár, Mannat Singh, and Ross Girshick. Fast and accurate model scaling. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 924–932, 2021.

[111] Olga Russakovsky, Jia Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, Michael Bernstein, et al. Imagenet large scale visual recognition challenge. *International journal of computer vision*, 115 (3):211–252, 2015.

[112] Lukas Tuggener, Jürgen Schmidhuber, and Thilo Stadelmann. Is it enough to optimize cnn architectures on imagenet? *arXiv preprint arXiv:2103.09108*, 2021.

[113] Sharath Girish, Debadeepta Dey, Neel Joshi, Vibhav Vineet, Shital Shah, Caio Cesar Teodoro Mendes, Abhinav Shrivastava, and Yale Song. One network doesn't rule them all: Moving beyond handcrafted architectures in self-supervised learning. *arXiv preprint arXiv:2203.08130*, 2022.

[114] Jifeng Dai, Haozhi Qi, Yuwen Xiong, Yi Li, Guodong Zhang, Han Hu, and Yichen Wei. Deformable convolutional networks. In *Proceedings of the IEEE international conference on computer vision*, pages 764–773, 2017.

[115] Silvia L Pintea, Nergis Tömen, Stanley F Goes, Marco Loog, and Jan C van Gemert. Resolution learning in deep convolutional networks using scale-space theory. *IEEE Transactions on Image Processing*, 30:8342–8353, 2021.

[116] David W. Romero, Robert-Jan Bruintjes, Jakub Mikolaj Tomczak, Erik J Bekkers, Mark Hoogendoorn, and Jan van Gemert. Flexconv: Continuous kernel convolutions with differentiable kernel sizes. In *International Conference on Learning Representations*, 2022. URL `https://openreview.net/forum?id=3jooF27-0Wy`.

[117] Adam Coates, Andrew Ng, and Honglak Lee. An analysis of single-layer networks in unsupervised feature learning. In *Proceedings of the fourteenth international conference on artificial intelligence and statistics*, pages 215–223. JMLR Workshop and Conference Proceedings, 2011.

[118] David Ha, Andrew M. Dai, and Quoc V. Le. Hypernetworks. In *International Conference on Learning Representations*, 2017. URL `https://openreview.net/forum?id=rkpACe1lx`.

[119] Alireza Ganjdanesh, Shangqian Gao, and Heng Huang. Effconv: efficient learning of kernel sizes for convolution layers of cnns. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 37, pages 7604–7612, 2023.

[120] Mingxing Tan and Quoc V Le. Efficientnet: Rethinking model scaling for convolutional neural networks. *ICML*, 2019.

[121] 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. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 1314–1324, 2019.

[122] Kai Han, Yunhe Wang, Qiulin Zhang, Wei Zhang, Chunjing Xu, and Tong Zhang. Model rubik's cube: Twisting resolution, depth and width for tinynets. *Advances in Neural Information Processing Systems*, 33:19353–19364, 2020.

[123] Shiwei Liu, Tianlong Chen, Xiaohan Chen, Xuxi Chen, Qiao Xiao, Boqian Wu, Mykola Pechenizkiy, Decebal Mocanu, and Zhangyang Wang. More convnets in the 2020s: Scaling up kernels beyond 51x51 using sparsity. *arXiv preprint arXiv:2207.03620*, 2022.

[124] 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.

[125] Nergis Tomen, Silvia-Laura Pintea, and Jan Van Gemert. Deep continuous networks. In Marina Meila and Tong Zhang, editors, *Proceedings of the 38th International Conference on Machine Learning*, volume 139 of *Proceedings of Machine Learning Research*, pages 10324–10335. PMLR, 18–24 Jul 2021. URL `https://proceedings.mlr.press/v139/tomen21a.html`.

[126] Shizhong Han, Zibo Meng, Zhiyuan Li, James O'Reilly, Jie Cai, Xiaofeng Wang, and Yan Tong. Optimizing filter size in convolutional neural networks for facial action unit recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 5070–5078, 2018.

[127] Evan Shelhamer, Dequan Wang, and Trevor Darrell. Blurring the line between structure and learning to optimize and adapt receptive fields. *arXiv preprint arXiv:1904.11487*, 2019.

[128] Zhitong Xiong, Yuan Yuan, Nianhui Guo, and Qi Wang. Variational context-deformable convnets for indoor scene parsing. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 3992–4002, 2020.

[129] Domen Tabernik, Matej Kristan, and Aleš Leonardis. Spatially-adaptive filter units for compact and efficient deep neural networks. *International Journal of Computer Vision*, 128(8):2049–2067, 2020.

[130] Shenlong Wang, Simon Suo, Wei-Chiu Ma, Andrei Pokrovsky, and Raquel Urtasun. Deep parametric continuous convolutional neural networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 2589–2597, 2018.

[131] Shaoshuai Shi, Zhe Wang, Jianping Shi, Xiaogang Wang, and Hongsheng Li. From points to parts: 3d object detection from point cloud with part-aware and part-aggregation network. *IEEE transactions on pattern analysis and machine intelligence*, 43(8):2647–2664, 2020.

[132] Nathaniel Thomas, Tess Smidt, Steven Kearnes, Lusann Yang, Li Li, Kai Kohlhoff, and Patrick Riley. Tensor field networks: Rotation-and translation-equivariant neural networks for 3d point clouds. *arXiv preprint arXiv:1802.08219*, 2018.

[133] Kristof Schütt, Pieter-Jan Kindermans, Huziel Enoc Sauceda Felix, Stefan Chmiela, Alexandre Tkatchenko, and Klaus-Robert Müller. Schnet: A continuous-filter convolutional neural network for modeling quantum interactions. *Advances in neural information processing systems*, 30, 2017.

[134] Martin Simonovsky and Nikos Komodakis. Dynamic edge-conditioned filters in convolutional neural networks on graphs. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 3693–3702, 2017.

[135] David W. Romero, Anna Kuzina, Erik J Bekkers, Jakub Mikolaj Tomczak, and Mark Hoogendoorn. CKConv: Continuous kernel convolution for sequential data. In *International Conference on Learning Representations*, 2022. URL `https://openreview.net/forum?id=8FhxBtXSl0`.

[136] Kyunghyun Cho, Bart van Merrienboer, Dzmitry Bahdanau, and Yoshua Bengio. On the properties of neural machine translation: Encoder-decoder approaches. In Dekai Wu, Marine Carpuat, Xavier Carreras, and Eva Maria Vecchi, editors, *Proceedings of SSST@EMNLP 2014, Eighth Workshop on Syntax, Semantics and Structure in Statistical Translation, Doha, Qatar, 25 October 2014*, pages 103–111. Association for Computational Linguistics, 2014. doi: 10.3115/v1/W14-4012. URL `https://aclanthology.org/W14-4012/`.

[137] Alireza Ganjdanesh, Shangqian Gao, and Heng Huang. Interpretations steered network pruning via amortized inferred saliency maps. In Shai Avidan, Gabriel Brostow, Moustapha Cissé, Giovanni Maria Farinella, and Tal Hassner, editors, *Computer Vision – ECCV 2022*, pages 278–296, Cham, 2022. Springer Nature Switzerland.

[138] Shangqian Gao, Feihu Huang, Jian Pei, and Heng Huang. Discrete model compression with resource constraint for deep neural networks. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2020.

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

[140] Sainbayar Sukhbaatar, Edouard Grave, Piotr Bojanowski, and Armand Joulin. Adaptive attention span in transformers. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pages 331–335, Florence, Italy, July 2019. Association for Computational Linguistics. doi: 10.18653/v1/P19-1032. URL https://aclanthology.org/P19-1032.

[141] Sepp Hochreiter, Yoshua Bengio, Paolo Frasconi, Jürgen Schmidhuber, et al. Gradient flow in recurrent nets: the difficulty of learning long-term dependencies, 2001.

[142] Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998.

[143] Patryk Chrabaszcz, Ilya Loshchilov, and Frank Hutter. A downsampled variant of imagenet as an alternative to the cifar datasets. *arXiv preprint arXiv:1707.08819*, 2017.

[144] Sergey Zagoruyko and Nikos Komodakis. Wide residual networks. In Richard C. Wilson, Edwin R. Hancock, and William A. P. Smith, editors, *Proceedings of the British Machine Vision Conference 2016, BMVC 2016, York, UK, September 19-22, 2016*. BMVA Press, 2016. URL http://www.bmva.org/bmvc/2016/papers/paper087/index.html.

[145] Ilya Loshchilov and Frank Hutter. Decoupled weight decay regularization. In *International Conference on Learning Representations*, 2019. URL https://openreview.net/forum?id=Bkg6RiCqY7.

[146] Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *International conference on machine learning*, pages 448–456. PMLR, 2015.

[147] Vittorio Mazzia, Francesco Salvetti, and Marcello Chiaberge. Efficient-capsnet: Capsule network with self-attention routing. *Scientific reports*, 11(1):1–13, 2021.

[148] Min Lin, Qiang Chen, and Shuicheng Yan. Network in network. *arXiv preprint arXiv:1312.4400*, 2013.

[149] HM Dipu Kabir, Moloud Abdar, Abbas Khosravi, Seyed Mohammad Jafar Jalali, Amir F Atiya, Saeid Nahavandi, and Dipti Srinivasan. Spinalnet: Deep neural network with gradual input. *IEEE Transactions on Artificial Intelligence*, 2022.

[150] Jorn-Henrik Jacobsen, Jan Van Gemert, Zhongyu Lou, and Arnold WM Smeulders. Structured receptive fields in cnns. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 2610–2619, 2016.

[151] Shaoqing Ren, Kaiming He, Ross Girshick, and Jian Sun. Faster r-cnn: Towards real-time object detection with region proposal networks. In *Advances in neural information processing systems*, pages 91–99, 2015.

[152] Chien-Yao Wang, Alexey Bochkovskiy, and Hong-Yuan Mark Liao. Yolov7: Trainable bag-of-freebies sets new state-of-the-art for real-time object detectors. *arXiv preprint arXiv:2207.02696*, 2022.

[153] Shiwei Liu, Tianlong Chen, Xiaohan Chen, Xuxi Chen, Qiao Xiao, Boqian Wu, Tommi Kärkkäinen, Mykola Pechenizkiy, Decebal Constantin Mocanu, and Zhangyang Wang. More convnets in the 2020s: Scaling up kernels beyond 51x51 using sparsity. In *International Conference on Learning Representations*, 2023. URL `https://openreview.net/forum?id=bXNl-myZkJl`.

[154] Sanghyun Woo, Shoubhik Debnath, Ronghang Hu, Xinlei Chen, Zhuang Liu, In So Kweon, and Saining Xie. Convnext v2: Co-designing and scaling convnets with masked autoencoders. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 16133–16142, 2023.

[155] Barret Zoph and Quoc Le. Neural architecture search with reinforcement learning. In *International Conference on Learning Representations*, 2017. URL `https://openreview.net/forum?id=r1Ue8Hcxg`.

[156] 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.

[157] Ningning Ma, Xiangyu Zhang, Hai-Tao Zheng, and Jian Sun. Shufflenet v2: Practical guidelines for efficient cnn architecture design. In *Proceedings of the European conference on computer vision (ECCV)*, pages 116–131, 2018.

[158] David Silver, Aja Huang, Chris J Maddison, Arthur Guez, Laurent Sifre, George Van Den Driessche, Julian Schrittwieser, Ioannis Antonoglou, Veda Panneershelvam, Marc Lanctot, et al. Mastering the game of go with deep neural networks and tree search. *nature*, 529(7587):484–489, 2016.

[159] Christopher Berner, Greg Brockman, Brooke Chan, Vicki Cheung, Przemysław Debiak, Christy Dennison, David Farhi, Quirin Fischer, Shariq Hashme, Chris Hesse, et al. Dota 2 with large scale deep reinforcement learning. *arXiv preprint arXiv:1912.06680*, 2019.

[160] Alhussein Fawzi, Matej Balog, Aja Huang, Thomas Hubert, Bernardino Romera-Paredes, Mohammadamin Barekatain, Alexander Novikov, Francisco J R Ruiz, Julian Schrittwieser, Grzegorz Swirszcz, et al. Discovering faster matrix multiplication algorithms with reinforcement learning. *Nature*, 610(7930):47–53, 2022.

[161] Anubhav Ashok, Nicholas Rhinehart, Fares Beainy, and Kris M. Kitani. N2n learning: Network to network compression via policy gradient reinforcement learning. In *International Conference on Learning Representations*, 2018. URL https://openreview.net/forum?id=B1hcZZ-AW.

[162] Zuxuan Wu, Tushar Nagarajan, Abhishek Kumar, Steven Rennie, Larry S Davis, Kristen Grauman, and Rogerio Feris. Blockdrop: Dynamic inference paths in residual networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 8817–8826, 2018.

[163] Xuanyang Zhang, Hao Liu, Zhanxing Zhu, and Zenglin Xu. Learning to search efficient densenet with layer-wise pruning. In *2020 International Joint Conference on Neural Networks (IJCNN)*, pages 1–8. IEEE, 2020.

[164] Qiangui Huang, Kevin Zhou, Suya You, and Ulrich Neumann. Learning to prune filters in convolutional neural networks. In *2018 IEEE Winter Conference on Applications of Computer Vision (WACV)*, pages 709–718. IEEE, 2018.

[165] Sixing Yu, Arya Mazaheri, and Ali Jannesari. Topology-aware network pruning using multi-stage graph embedding and reinforcement learning. In *International Conference on Machine Learning*, pages 25656–25667. PMLR, 2022.

[166] Timothy P. Lillicrap, Jonathan J. Hunt, Alexander Pritzel, Nicolas Heess, Tom Erez, Yuval Tassa, David Silver, and Daan Wierstra. Continuous control with deep reinforcement learning. In Yoshua Bengio and Yann LeCun, editors, *4th International Conference on Learning Representations, ICLR 2016, San Juan, Puerto Rico, May 2-4, 2016, Conference Track Proceedings*, 2016. URL http://arxiv.org/abs/1509.02971.

[167] Khimya Khetarpal, Matthew Riemer, Irina Rish, and Doina Precup. Towards continual reinforcement learning: A review and perspectives. *Journal of Artificial Intelligence Research*, 75:1401–1476, 2022.

[168] Tuomas Haarnoja, Aurick Zhou, Pieter Abbeel, and Sergey Levine. Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor. In *International conference on machine learning*, pages 1861–1870. PMLR, 2018.

[169] Alireza Ganjdanesh, Shangqian Gao, and Heng Huang. Jointly training and pruning cnns via learnable agent guidance and alignment. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 16058–16069, 2024.

[170] Deepak Ghimire, Dayoung Kil, and Seong-heum Kim. A survey on efficient convolutional neural networks and hardware acceleration. *Electronics*, 11(6):945, 2022.

[171] Giosué Cataldo Marinó, Alessandro Petrini, Dario Malchiodi, and Marco Frasca. Deep neural networks compression: A comparative survey and choice recommendations. *Neurocomputing*, 520:152–170, 2023.

[172] Yang Sui, Miao Yin, Yi Xie, Huy Phan, Saman Aliari Zonouz, and Bo Yuan. Chip: Channel independence-based pruning for compact neural networks. *Advances in Neural Information Processing Systems*, 34:24604–24616, 2021.

[173] Miao Yin, Yang Sui, Wanzhao Yang, Xiao Zang, Yu Gong, and Bo Yuan. Hodec: Towards efficient high-order decomposed convolutional neural networks. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 12299–12308, 2022.

[174] Shangqian Gao, Feihu Huang, Yanfu Zhang, and Heng Huang. Disentangled differentiable network pruning. In *Computer Vision–ECCV 2022: 17th European Conference, Tel Aviv, Israel, October 23–27, 2022, Proceedings, Part XI*, pages 328–345. Springer, 2022.

[175] Shangqian Gao, Burak Uzkent, Yilin Shen, Heng Huang, and Hongxia Jin. Learning to jointly share and prune weights for grounding based vision and language models. In *The Eleventh International Conference on Learning Representations*, 2022.

[176] Shangqian Gao, Zeyu Zhang, Yanfu Zhang, Feihu Huang, and Heng Huang. Structural alignment for network pruning through partial regularization. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 17402–17412, 2023.

[177] Alireza Ganjdanesh, Shangqian Gao, Hirad Alipanah, and Heng Huang. Compressing image-to-image translation gans using local density structures on their learned manifold. *arXiv preprint arXiv:2312.14776*, 2023.

[178] Longguang Wang, Xiaoyu Dong, Yingqian Wang, Li Liu, Wei An, and Yulan Guo. Learnable lookup table for neural network quantization. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 12423–12433, 2022.

[179] Xin Yu, Thiago Serra, Srikumar Ramalingam, and Shandian Zhe. The combinatorial brain surgeon: Pruning weights that cancel one another in neural networks. In *International Conference on Machine Learning*, pages 25668–25683. PMLR, 2022.

[180] Jonathan Frankle and Michael Carbin. The lottery ticket hypothesis: Finding sparse, trainable neural networks. In *International Conference on Learning Representations*, 2019. URL `https://openreview.net/forum?id=rJl-b3RcF7`.

[181] Colin White, Mahmoud Safari, Rhea Sukthanker, Binxin Ru, Thomas Elsken, Arber Zela, Debadeepta Dey, and Frank Hutter. Neural architecture search: Insights from 1000 papers. *arXiv preprint arXiv:2301.08727*, 2023.

[182] Jianping Gou, Baosheng Yu, Stephen J. Maybank, and Dacheng Tao. Knowledge distillation: A survey. *Int. J. Comput. Vis.*, 129(6):1789–1819, 2021. doi: 10.1007/s11263-021-01453-z. URL `https://doi.org/10.1007/s11263-021-01453-z`.

[183] Yihui He, Xiangyu Zhang, and Jian Sun. Channel pruning for accelerating very deep neural networks. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 1389–1397, 2017.

[184] Robert Tibshirani. Regression shrinkage and selection via the lasso. *Journal of the Royal Statistical Society: Series B (Methodological)*, 58(1):267–288, 1996.

[185] Pavlo Molchanov, Stephen Tyree, Tero Karras, Timo Aila, and Jan Kautz. Pruning convolutional neural networks for resource efficient inference. In *International Conference on Learning Representations*, 2017. URL `https://openreview.net/forum?id=SJGCiw5gl`.

[186] Xu Wang, Sen Wang, Xingxing Liang, Dawei Zhao, Jincai Huang, Xin Xu, Bin Dai, and Qiguang Miao. Deep reinforcement learning: A survey. *IEEE Transactions on Neural Networks and Learning Systems*, pages 1–15, 2022. doi: 10.1109/TNNLS.2022.3207346.

[187] Feihu Huang, Shangqian Gao, and Heng Huang. Bregman gradient policy optimization. In *International Conference on Learning Representations*, 2022. URL `https://openreview.net/forum?id=ZU-zFnTum1N`.

[188] Christopher JCH Watkins and Peter Dayan. Q-learning. *Machine learning*, 8:279–292, 1992.

[189] Richard S Sutton, David McAllester, Satinder Singh, and Yishay Mansour. Policy gradient methods for reinforcement learning with function approximation. *Advances in neural information processing systems*, 12, 1999.

[190] Emmanuel Bengio, Joelle Pineau, and Doina Precup. Interference and generalization in temporal difference learning. In *International Conference on Machine Learning*, pages 767–777. PMLR, 2020.

[191] Tianhe Yu, Saurabh Kumar, Abhishek Gupta, Sergey Levine, Karol Hausman, and Chelsea Finn. Gradient surgery for multi-task learning. *Advances in Neural Information Processing Systems*, 33:5824–5836, 2020.

[192] Chelsea Finn, Pieter Abbeel, and Sergey Levine. Model-agnostic meta-learning for fast adaptation of deep networks. In *International conference on machine learning*, pages 1126–1135. PMLR, 2017.

[193] Yan Duan, John Schulman, Xi Chen, Peter L. Bartlett, Ilya Sutskever, and Pieter Abbeel. RL^2: Fast reinforcement learning via slow reinforcement learning, 2017. URL `https://openreview.net/forum?id=HkLXCE9lx`.

[194] Jane X Wang, Zeb Kurth-Nelson, Dhruva Tirumala, Hubert Soyer, Joel Z Leibo, Remi Munos, Charles Blundell, Dharshan Kumaran, and Matt Botvinick. Learning to reinforcement learn. *arXiv preprint arXiv:1611.05763*, 2016.

[195] Jürgen Schmidhuber. Powerplay: Training an increasingly general problem solver by continually searching for the simplest still unsolvable problem. *Frontiers in psychology*, 4:313, 2013.

[196] Rémy Portelas, Cédric Colas, Lilian Weng, Katja Hofmann, and Pierre-Yves Oudeyer. Automatic curriculum learning for deep rl: A short survey. In Christian Bessiere, editor, *Proceedings of the Twenty-Ninth International Joint Conference on Artificial Intelligence, IJCAI-20*, pages 4819–4825. International Joint Conferences on Artificial Intelligence Organization, 7 2020. doi: 10.24963/ijcai.2020/671. URL `https://doi.org/10.24963/ijcai.2020/671`. Survey track.

[197] Jiayu Chen, Yuanxin Zhang, Yuanfan Xu, Huimin Ma, Huazhong Yang, Jiaming Song, Yu Wang, and Yi Wu. Variational automatic curriculum learning for sparse-reward cooperative multi-agent problems. *Advances in Neural Information Processing Systems*, 34:9681–9693, 2021.

[198] Wang Chi Cheung, David Simchi-Levi, and Ruihao Zhu. Reinforcement learning for non-stationary markov decision processes: The blessing of (more) optimism. In *International Conference on Machine Learning*, pages 1843–1854. PMLR, 2020.

[199] Omar Darwiche Domingues, Pierre Ménard, Matteo Pirotta, Emilie Kaufmann, and Michal Valko. A kernel-based approach to non-stationary reinforcement learning in metric spaces. In *International Conference on Artificial Intelligence and Statistics*, pages 3538–3546. PMLR, 2021.

[200] Ahmed Touati and Pascal Vincent. Efficient learning in non-stationary linear markov decision processes. *arXiv preprint arXiv:2010.12870*, 2020.

[201] Kyunghyun Cho, B van Merrienboer, Caglar Gulcehre, F Bougares, H Schwenk, and Yoshua Bengio. Learning phrase representations using rnn encoder-decoder for statistical machine translation. In *Conference on Empirical Methods in Natural Language Processing (EMNLP 2014)*, 2014.

[202] Yunqiang Li, Jan C van Gemert, Torsten Hoefler, Bert Moons, Evangelos Eleftheriou, and Bram-Ernst Verhoef. Differentiable transportation pruning. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 16957–16967, 2023.

[203] Yehui Tang, Yunhe Wang, Yixing Xu, Dacheng Tao, Chunjing Xu, Chao Xu, and Chang Xu. Scop: Scientific control for reliable neural network pruning. *Advances in Neural Information Processing Systems*, 33:10936–10947, 2020.

[204] Jinyang Guo, Wanli Ouyang, and Dong Xu. Multi-dimensional pruning: A unified framework for model compression. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 1508–1517, 2020.

[205] Xuanyi Dong, Junshi Huang, Yi Yang, and Shuicheng Yan. More is less: A more complicated network with less inference complexity. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 5840–5848, 2017.

[206] Lucas Liebenwein, Cenk Baykal, Harry Lang, Dan Feldman, and Daniela Rus. Provable filter pruning for efficient neural networks. In *International Conference on Learning Representations*, 2020.

[207] Charles Herrmann, Richard Strong Bowen, and Ramin Zabih. Channel selection using gumbel softmax. In *Computer Vision–ECCV 2020: 16th European Conference, Glasgow, UK, August 23–28, 2020, Proceedings, Part XXVII*, pages 241–257. Springer, 2020.

[208] Linhang Cai, Zhulin An, Chuanguang Yang, Yangchun Yan, and Yongjun Xu. Prior gradient mask guided pruning-aware fine-tuning. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 36, pages 140–148, 2022.

[209] Sara Elkerdawy, Mostafa Elhoushi, Hong Zhang, and Nilanjan Ray. Fire together wire together: A dynamic pruning approach with self-supervised mask prediction. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 12454–12463, 2022.

[210] Yawei Li, Kamil Adamczewski, Wen Li, Shuhang Gu, Radu Timofte, and Luc Van Gool. Revisiting random channel pruning for neural network compression. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 191–201, 2022.

[211] Ilya Sutskever, James Martens, George Dahl, and Geoffrey Hinton. On the importance of initialization and momentum in deep learning. In *International conference on machine learning*, pages 1139–1147. PMLR, 2013.

[212] Ilya Loshchilov and Frank Hutter. SGDR: Stochastic gradient descent with warm restarts. In *International Conference on Learning Representations*, 2017. URL `https://openreview.net/forum?id=Skq89Scxx`.

[213] Phillip Isola, Jun-Yan Zhu, Tinghui Zhou, and Alexei A Efros. Image-to-image translation with conditional adversarial networks. In *CVPR*, pages 1125–1134, 2017.

[214] Jun-Yan Zhu, Taesung Park, Phillip Isola, and Alexei A Efros. Unpaired image-to-image translation using cycle-consistent adversarial networks. In *ICCV*, 2017.

[215] Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial nets. *NeurIPS*, 27, 2014.

[216] Xun Huang and Serge Belongie. Arbitrary style transfer in real-time with adaptive instance normalization. In *ICCV*, 2017.

[217] Taesung Park, Ming-Yu Liu, Ting-Chun Wang, and Jun-Yan Zhu. Semantic image synthesis with spatially-adaptive normalization. In *CVPR*, 2019.

[218] Christian Ledig, Lucas Theis, Ferenc Huszár, Jose Caballero, Andrew Cunningham, Alejandro Acosta, Andrew Aitken, Alykhan Tejani, Johannes Totz, Zehan Wang, et al. Photo-realistic single image super-resolution using a generative adversarial network. In *CVPR*, 2017.

[219] Xintao Wang, Ke Yu, Shixiang Wu, Jinjin Gu, Yihao Liu, Chao Dong, Yu Qiao, and Chen Change Loy. Esrgan: Enhanced super-resolution generative adversarial networks. In *ECCV workshops*, 2018.

[220] Ting-Chun Wang, Ming-Yu Liu, Jun-Yan Zhu, Guilin Liu, Andrew Tao, Jan Kautz, and Bryan Catanzaro. Video-to-video synthesis. *arXiv preprint arXiv:1808.06601*, 2018.

[221] Caroline Chan, Shiry Ginosar, Tinghui Zhou, and Alexei A Efros. Everybody dance now. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 5933–5942, 2019.

[222] Alvin Wan, Xiaoliang Dai, Peizhao Zhang, Zijian He, Yuandong Tian, Saining Xie, Bichen Wu, Matthew Yu, Tao Xu, Kan Chen, et al. Fbnetv2: Differentiable neural architecture search for spatial and channel dimensions. In *CVPR*, 2020.

[223] Xinyu Gong, Shiyu Chang, Yifan Jiang, and Zhangyang Wang. Autogan: Neural architecture search for generative adversarial networks. In *ICCV*, pages 3224–3234, 2019.

[224] Muyang Li, Ji Lin, Yaoyao Ding, Zhijian Liu, Jun-Yan Zhu, and Song Han. Gan compression: Efficient architectures for interactive conditional gans. In *CVPR*, 2020.

[225] Qing Jin, Jian Ren, Oliver J Woodford, Jiazhuo Wang, Geng Yuan, Yanzhi Wang, and Sergey Tulyakov. Teachers do more than teach: Compressing image-to-image models. In *CVPR*, 2021.

[226] Chen Gao, Yunpeng Chen, Si Liu, Zhenxiong Tan, and Shuicheng Yan. Adversarialnas: Adversarial neural architecture search for gans. In *CVPR*, pages 5680–5689, 2020.

[227] Shaojie Li, Mingbao Lin, Yan Wang, Chao Fei, Ling Shao, and Rongrong Ji. Learning efficient gans for image translation via differentiable masks and co-attention distillation. *IEEE Transactions on Multimedia*, 2022.

[228] Angeline Aguinaldo, Ping-Yeh Chiang, Alex Gain, Ameya Patil, Kolten Pearson, and Soheil Feizi. Compressing gans using knowledge distillation. *arXiv preprint arXiv:1902.00159*, 2019.

[229] Haotao Wang, Shupeng Gui, Haichuan Yang, Ji Liu, and Zhangyang Wang. Gan slimming: All-in-one gan compression by a unified optimization framework. In *ECCV*, 2020.

[230] Ting-Yun Chang and Chi-Jen Lu. Tinygan: Distilling biggan for conditional image generation. In *Proceedings of the Asian Conference on Computer Vision*, 2020.

[231] Hanting Chen, Yunhe Wang, Han Shu, Changyuan Wen, Chunjing Xu, Boxin Shi, Chao Xu, and Chang Xu. Distilling portable generative adversarial networks for image translation. In *Proceedings of the AAAI Conference on Artificial Intelligence*, 2020.

[232] Yonggan Fu, Wuyang Chen, Haotao Wang, Haoran Li, Yingyan Lin, and Zhangyang Wang. Autogan-distiller: Searching to compress generative adversarial networks. In *ICML*, 2020.

[233] Liang Hou, Zehuan Yuan, Lei Huang, Huawei Shen, Xueqi Cheng, and Changhu Wang. Slimmable generative adversarial networks. In *AAAI*. AAAI Press, 2021.

[234] Linfeng Zhang, Xin Chen, Xiaobing Tu, Pengfei Wan, Ning Xu, and Kaisheng Ma. Wavelet knowledge distillation: Towards efficient image-to-image translation. In *CVPR*, 2022.

[235] Linfeng Zhang, Xin Chen, Runpei Dong, and Kaisheng Ma. Region-aware knowledge distillation for efficient image-to-image translation. *arXiv preprint arXiv:2205.12451*, 2022.

[236] Shaojie Li, Jie Wu, Xuefeng Xiao, Fei Chao, Xudong Mao, and Rongrong Ji. Revisiting discriminator in gan compression: A generator-discriminator cooperative compression scheme. *Advances in Neural Information Processing Systems*, 34, 2021.

[237] Emanuel Parzen. On estimation of a probability density function and mode. *The annals of mathematical statistics*, 33(3), 1962.

[238] Alireza Ganjdanesh, Shangqian Gao, Hirad Alipanah, and Heng Huang. Compressing image-to-image translation gans using local density structures on their learned manifold. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 38, pages 12118–12126, 2024.

[239] Han Shu, Yunhe Wang, Xu Jia, Kai Han, Hanting Chen, Chunjing Xu, Qi Tian, and Chang Xu. Co-evolutionary compression for unpaired image translation. In *ICCV*, 2019.

[240] Ji Lin, Richard Zhang, Frieder Ganz, Song Han, and Jun-Yan Zhu. Anycost gans for interactive image synthesis and editing. In *CVPR*, 2021.

[241] Jiahao Wang, Han Shu, Weihao Xia, Yujiu Yang, and Yunhe Wang. Coarse-to-fine searching for efficient generative adversarial networks. *arXiv preprint arXiv:2104.09223*, 2021.

[242] Chong Yu and Jeff Pool. Self-supervised generative adversarial compression. *NeurIPS*, 33, 2020.

[243] Yuxi Ren, Jie Wu, Xuefeng Xiao, and Jianchao Yang. Online multi-granularity distillation for gan compression. In *ICCV*, 2021.

[244] Joshua B Tenenbaum, Vin de Silva, and John C Langford. A global geometric framework for nonlinear dimensionality reduction. *science*, 290(5500):2319–2323, 2000.

[245] Arantxa Casanova, Marlène Careil, Jakob Verbeek, Michal Drozdzal, and Adriana Romero-Soriano. Instance-conditioned gan. In *Advances in Neural Information Processing Systems*, 2021.

[246] Mahyar Khayatkhoei, Maneesh K Singh, and Ahmed Elgammal. Disconnected manifold learning for generative adversarial networks. *NeurIPS*, 31, 2018.

[247] Yao Ni, Piotr Koniusz, Richard Hartley, and Richard Nock. Manifold learning benefits gans. In *CVPR*, 2022.

[248] Jiezhang Cao, Yong Guo, Qingyao Wu, Chunhua Shen, Junzhou Huang, and Mingkui Tan. Adversarial learning with local coordinate coding. In *International Conference on Machine Learning*, pages 707–715, 2018.

[249] James Oldfield, Georgopoulos Markos, Yannis Panagakis, Mihalis A. Nicolaou, and Patras Ioannis. Tensor component analysis for interpreting the latent space of gans. In *BMVC*, November 2021.

[250] Erik Härkönen, Aaron Hertzmann, Jaakko Lehtinen, and Sylvain Paris. Ganspace: Discovering interpretable gan controls. *Advances in Neural Information Processing Systems*, 2020.

[251] Jimmy Ba and Rich Caruana. Do deep nets really need to be deep? In *NeurIPS*, 2014.

[252] Bichen Wu, Xiaoliang Dai, Peizhao Zhang, Yanghan Wang, Fei Sun, Yiming Wu, Yuandong Tian, Peter Vajda, Yangqing Jia, and Kurt Keutzer. Fbnet: Hardware-aware efficient convnet design via differentiable neural architecture search. In *CVPR*, 2019.

[253] Christos Tzelepis, Georgios Tzimiropoulos, and Ioannis Patras. Warpedganspace: Finding non-linear rbf paths in gan latent space. In *ICCV*, 2021.

[254] Yujun Shen and Bolei Zhou. Closed-form factorization of latent semantics in gans. In *CVPR*, 2021.

[255] Jaewoong Choi, Junho Lee, Changyeon Yoon, Jung Ho Park, Geonho Hwang, and Myungjoo Kang. Do not escape from the manifold: Discovering the local coordinates on the latent space of GANs. In *International Conference on Learning Representations*, 2022. URL `https://openreview.net/forum?id=aTzMi4yV_RO`.

[256] Andrey Voynov and Artem Babenko. Unsupervised discovery of interpretable directions in the gan latent space. In *International conference on machine learning*, 2020.

[257] Ting Chen, Simon Kornblith, Mohammad Norouzi, and Geoffrey Hinton. A simple framework for contrastive learning of visual representations. In *International conference on machine learning*, 2020.

[258] Xinlei Chen, Haoqi Fan, Ross Girshick, and Kaiming He. Improved baselines with momentum contrastive learning. *arXiv preprint arXiv:2003.04297*, 2020.

[259] 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. *NeurIPS*, 2020.

[260] Oriol Vinyals, Charles Blundell, Timothy Lillicrap, Daan Wierstra, et al. Matching networks for one shot learning. *Advances in neural information processing systems*, 29, 2016.

[261] Hugo Touvron, Alexandre Sablayrolles, Matthijs Douze, Matthieu Cord, and Hervé Jégou. Grafit: Learning fine-grained image representations with coarse labels. In *ICCV*, 2021.

[262] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: a simple way to prevent neural networks from overfitting. *The journal of machine learning research*, 15(1):1929–1958, 2014.

[263] Kyunghyun Cho, Bart van Merrienboer, Çaglar Gülçehre, Dzmitry Bahdanau, Fethi Bougares, Holger Schwenk, and Yoshua Bengio. Learning phrase representations using RNN encoder-decoder for statistical machine translation. In *EMNLP 2014*, 2014.

[264] Emil Julius Gumbel. *Statistical theory of extreme values and some practical applications: a series of lectures*, volume 33. US Government Printing Office, 1954.

[265] Róbert Csordás, Sjoerd van Steenkiste, and Jürgen Schmidhuber. Are neural nets modular? inspecting functional modularity through differentiable weight masks. In *International Conference on Learning Representations*, 2021. URL `https://openreview.net/forum?id=7uVcpu-gMD`.

[266] Bo-Kyeong Kim, Shinkook Choi, and Hancheol Park. Cut inner layers: A structured pruning strategy for efficient u-net gans. *arXiv preprint arXiv:2206.14658*, 2022.

[267] Aron Yu and Kristen Grauman. Fine-grained visual comparisons with local learning. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 192–199, 2014.

[268] Marius Cordts, Mohamed Omran, Sebastian Ramos, Timo Rehfeld, Markus Enzweiler, Rodrigo Benenson, Uwe Franke, Stefan Roth, and Bernt Schiele. The cityscapes dataset for semantic urban scene understanding. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 3213–3223, 2016.

[269] Martin Heusel, Hubert Ramsauer, Thomas Unterthiner, Bernhard Nessler, and Sepp Hochreiter. Gans trained by a two time-scale update rule converge to a local nash equilibrium. *Advances in neural information processing systems*, 30, 2017.

[270] Dmitry Ulyanov, Andrea Vedaldi, and Victor Lempitsky. Instance normalization: The missing ingredient for fast stylization. *arXiv preprint arXiv:1607.08022*, 2016.

[271] Justin Johnson, Alexandre Alahi, and Li Fei-Fei. Perceptual losses for real-time style transfer and super-resolution. In *European conference on computer vision*, pages 694–711. Springer, 2016.

[272] Jascha Sohl-Dickstein, Eric Weiss, Niru Maheswaranathan, and Surya Ganguli. Deep unsupervised learning using nonequilibrium thermodynamics. In *International conference on machine learning*, pages 2256–2265. PMLR, 2015.

[273] Yang Song and Stefano Ermon. Generative modeling by estimating gradients of the data distribution. *Advances in neural information processing systems*, 32, 2019.

[274] Prafulla Dhariwal and Alexander Nichol. Diffusion models beat gans on image synthesis. *Advances in neural information processing systems*, 34:8780–8794, 2021.

[275] Lvmin Zhang and Maneesh Agrawala. Adding conditional control to text-to-image diffusion models. *arXiv preprint arXiv:2302.05543*, 2023.

[276] Shihao Zhao, Dongdong Chen, Yen-Chun Chen, Jianmin Bao, Shaozhe Hao, Lu Yuan, and Kwan-Yee K Wong. Uni-controlnet: All-in-one control to text-to-image diffusion models. *arXiv preprint arXiv:2305.16322*, 2023.

[277] Chitwan Saharia, Jonathan Ho, William Chan, Tim Salimans, David J Fleet, and Mohammad Norouzi. Image super-resolution via iterative refinement. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 45(4):4713–4726, 2022.

[278] Sicheng Gao, Xuhui Liu, Bohan Zeng, Sheng Xu, Yanjing Li, Xiaoyan Luo, Jianzhuang Liu, Xiantong Zhen, and Baochang Zhang. Implicit diffusion models for continuous super-resolution. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 10021–10030, 2023.

[279] Jonathan Ho, William Chan, Chitwan Saharia, Jay Whang, Ruiqi Gao, Alexey Gritsenko, Diederik P Kingma, Ben Poole, Mohammad Norouzi, David J Fleet, et al. Imagen video: High definition video generation with diffusion models. *arXiv preprint arXiv:2210.02303*, 2022.

[280] Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial nets. *Advances in neural information processing systems*, 27, 2014.

[281] Diederik P. Kingma and Max Welling. Auto-encoding variational bayes. In Yoshua Bengio and Yann LeCun, editors, *2nd International Conference on Learning Representations, ICLR 2014, Banff, AB, Canada, April 14-16, 2014, Conference Track Proceedings*, 2014. URL `http://arxiv.org/abs/1312.6114`.

[282] Cheng Lu, Yuhao Zhou, Fan Bao, Jianfei Chen, Chongxuan Li, and Jun Zhu. Dpm-solver: A fast ode solver for diffusion probabilistic model sampling in around 10 steps. *Advances in Neural Information Processing Systems*, 35:5775–5787, 2022.

[283] Fan Bao, Chongxuan Li, Jun Zhu, and Bo Zhang. Analytic-DPM: an analytic estimate of the optimal reverse variance in diffusion probabilistic models. In *International Conference on Learning Representations*, 2022. URL `https://openreview.net/forum?id=0xiJLKH-ufZ`.

[284] Luping Liu, Yi Ren, Zhijie Lin, and Zhou Zhao. Pseudo numerical methods for diffusion models on manifolds. In *International Conference on Learning Representations*, 2022. URL `https://openreview.net/forum?id=PlKWVd2yBkY`.

[285] Qinsheng Zhang and Yongxin Chen. Fast sampling of diffusion models with exponential integrator. In *The Eleventh International Conference on Learning Representations*, 2023. URL `https://openreview.net/forum?id=Loek7hfb46P`.

[286] Jiaming Song, Chenlin Meng, and Stefano Ermon. Denoising diffusion implicit models. In *International Conference on Learning Representations*, 2021. URL `https://openreview.net/forum?id=St1giarCHLP`.

[287] Alexander Quinn Nichol and Prafulla Dhariwal. Improved denoising diffusion probabilistic models. In *International Conference on Machine Learning*, pages 8162–8171. PMLR, 2021.

[288] Chenlin Meng, Robin Rombach, Ruiqi Gao, Diederik Kingma, Stefano Ermon, Jonathan Ho, and Tim Salimans. On distillation of guided diffusion models. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 14297–14306, 2023.

[289] Tim Salimans and Jonathan Ho. Progressive distillation for fast sampling of diffusion models. In *International Conference on Learning Representations*, 2022. URL `https://openreview.net/forum?id=TIdIXIpzhoI`.

[290] Amirhossein Habibian, Amir Ghodrati, Noor Fathima, Guillaume Sautiere, Risheek Garrepalli, Fatih Porikli, and Jens Petersen. Clockwork diffusion: Efficient generation with model-step distillation. *arXiv preprint arXiv:2312.08128*, 2023.

[291] Yunsung Lee, Jin-Young Kim, Hyojun Go, Myeongho Jeong, Shinhyeok Oh, and Seungtaek Choi. Multi-architecture multi-expert diffusion models. *arXiv preprint arXiv:2306.04990*, 2023.

[292] Huijie Zhang, Yifu Lu, Ismail Alkhouri, Saiprasad Ravishankar, Dogyoon Song, and Qing Qu. Improving efficiency of diffusion models via multi-stage framework and tailored multi-decoder architectures. *arXiv preprint arXiv:2312.09181*, 2023.

[293] Xingyi Yang, Daquan Zhou, Jiashi Feng, and Xinchao Wang. Diffusion probabilistic model made slim. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 22552–22562, 2023.

[294] Hyojun Go, Jinyoung Kim, Yunsung Lee, Seunghyun Lee, Shinhyeok Oh, Hyeongdon Moon, and Seungtaek Choi. Addressing negative transfer in diffusion models. In *Thirty-seventh Conference on Neural Information Processing Systems*, 2023. URL https://openreview.net/forum?id=3G2ec833mW.

[295] Olaf Ronneberger, Philipp Fischer, and Thomas Brox. U-net: Convolutional networks for biomedical image segmentation. In *Medical Image Computing and Computer-Assisted Intervention–MICCAI 2015: 18th International Conference, Munich, Germany, October 5-9, 2015, Proceedings, Part III 18*, pages 234–241. Springer, 2015.

[296] Alireza Ganjdanesh, Yan Kang, Yuchen Liu, Richard Zhang, Zhe Lin, and Heng Huang. Mixture of efficient diffusion experts through automatic interval and sub-network selection. In *European Conference on Computer Vision*, pages 54–71. Springer, 2024.

[297] Cheng Lu, Yuhao Zhou, Fan Bao, Jianfei Chen, Chongxuan Li, and Jun Zhu. DPM-solver++: Fast solver for guided sampling of diffusion probabilistic models, 2023. URL https://openreview.net/forum?id=4vGwQqviud5.

[298] Yilun Xu, Mingyang Deng, Xiang Cheng, Yonglong Tian, Ziming Liu, and Tommi Jaakkola. Restart sampling for improving generative processes. *arXiv preprint arXiv:2306.14878*, 2023.

[299] Huangjie Zheng, Pengcheng He, Weizhu Chen, and Mingyuan Zhou. Truncated diffusion probabilistic models and diffusion-based adversarial auto-encoders. In *The Eleventh International Conference on Learning Representations*, 2023. URL https://openreview.net/forum?id=HDxgaKk956l.

[300] Qinsheng Zhang, Molei Tao, and Yongxin Chen. gDDIM: Generalized denoising diffusion implicit models. In *The Eleventh International Conference on Learning Representations*, 2023. URL https://openreview.net/forum?id=1hKE9qjvz-.

[301] Diederik Kingma, Tim Salimans, Ben Poole, and Jonathan Ho. Variational diffusion models. *Advances in neural information processing systems*, 34:21696–21707, 2021.

[302] Daniel Watson, William Chan, Jonathan Ho, and Mohammad Norouzi. Learning fast samplers for diffusion models by differentiating through sample quality. In *International Conference on Learning Representations*, 2022. URL https://openreview.net/forum?id=VFBjuF8HEp.

[303] Daniel Watson, Jonathan Ho, Mohammad Norouzi, and William Chan. Learning to efficiently sample from diffusion probabilistic models, 2022. URL https://openreview.net/forum?id=LOz0xDpw4Y.

[304] Xinyin Ma, Gongfan Fang, and Xinchao Wang. Deepcache: Accelerating diffusion models for free. *arXiv preprint arXiv:2312.00858*, 2023.

[305] Arash Vahdat, Karsten Kreis, and Jan Kautz. Score-based generative modeling in latent space. *Advances in Neural Information Processing Systems*, 34:11287–11302, 2021.

[306] Patrick Esser, Robin Rombach, and Bjorn Ommer. Taming transformers for high-resolution image synthesis. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 12873–12883, 2021.

[307] Fisher Yu, Yinda Zhang, Shuran Song, Ari Seff, and Jianxiong Xiao. Lsun: Construction of a large-scale image dataset using deep learning with humans in the loop. *arXiv preprint arXiv:1506.03365*, 2015.

[308] Yang Zhao, Yanwu Xu, Zhisheng Xiao, and Tingbo Hou. Mobilediffusion: Subsecond text-to-image generation on mobile devices. *arXiv preprint arXiv:2311.16567*, 2023.

[309] Zizheng Pan, Bohan Zhuang, De-An Huang, Weili Nie, Zhiding Yu, Chaowei Xiao, Jianfei Cai, and Anima Anandkumar. T-stitch: Accelerating sampling in pre-trained diffusion models with trajectory stitching. *arXiv preprint arXiv:2402.14167*, 2024.

[310] Hao Li, Asim Kadav, Igor Durdanovic, Hanan Samet, and Hans Peter Graf. Pruning filters for efficient convnets. In *International Conference on Learning Representations*, 2017. URL `https://openreview.net/forum?id=rJqFGTslg`.

[311] Song Han, Jeff Pool, John Tran, and William Dally. Learning both weights and connections for efficient neural network. *Advances in neural information processing systems*, 28, 2015.

[312] Pavlo Molchanov, Arun Mallya, Stephen Tyree, Iuri Frosio, and Jan Kautz. Importance estimation for neural network pruning. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2019.

[313] Yihui He, Ji Lin, Zhijian Liu, Hanrui Wang, Li-Jia Li, and Song Han. Amc: Automl for model compression and acceleration on mobile devices. In *Proceedings of the European conference on computer vision (ECCV)*, pages 784–800, 2018.

[314] Alireza Ganjdanesh, Shangqian Gao, and Heng Huang. Interpretations steered network pruning via amortized inferred saliency maps. In *European Conference on Computer Vision*, pages 278–296. Springer, 2022.

[315] Barret Zoph and Quoc Le. Neural architecture search with reinforcement learning. In *International Conference on Learning Representations*, 2017. URL `https://openreview.net/forum?id=r1Ue8Hcxg`.

[316] Hanxiao Liu, Karen Simonyan, and Yiming Yang. DARTS: Differentiable architecture search. In *International Conference on Learning Representations*, 2019. URL `https://openreview.net/forum?id=S1eYHoC5FX`.

[317] Han Cai, Chuang Gan, Tianzhe Wang, Zhekai Zhang, and Song Han. Once-for-all: Train one network and specialize it for efficient deployment. In *International Conference on Learning Representations*, 2020. URL `https://openreview.net/forum?id=HylxE1HKwS`.

[318] Lewei Yao, Renjie Pi, Hang Xu, Wei Zhang, Zhenguo Li, and Tong Zhang. Joint-detnas: Upgrade your detector with nas, pruning and dynamic distillation. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 10175–10184, 2021.

[319] Lu Hou, Zhiqi Huang, Lifeng Shang, Xin Jiang, Xiao Chen, and Qun Liu. Dynabert: Dynamic bert with adaptive width and depth. *Advances in Neural Information Processing Systems*, 33:9782–9793, 2020.

[320] Colin White, Mahmoud Safari, Rhea Sukthanker, Binxin Ru, Thomas Elsken, Arber Zela, Debadeepta Dey, and Frank Hutter. Neural architecture search: Insights from 1000 papers. *arXiv preprint arXiv:2301.08727*, 2023.

[321] Yang He and Lingao Xiao. Structured pruning for deep convolutional neural networks: A survey. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, pages 1–20, 2023. doi: 10.1109/TPAMI.2023.3334614.

[322] Hongrong Cheng, Miao Zhang, and Javen Qinfeng Shi. A survey on deep neural network pruning-taxonomy, comparison, analysis, and recommendations. *arXiv preprint arXiv:2308.06767*, 2023.

[323] 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.

[324] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. *Advances in neural information processing systems*, 30, 2017.

[325] Jooyoung Choi, Jungbeom Lee, Chaehun Shin, Sungwon Kim, Hyunwoo Kim, and Sungroh Yoon. Perception prioritized training of diffusion models. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 11472–11481, 2022.

[326] Olga Russakovsky, Jia Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, Michael Bernstein, Alexander C. Berg, and Li Fei-Fei. ImageNet Large Scale Visual Recognition Challenge. *International Journal of Computer Vision (IJCV)*, 115(3):211–252, 2015. doi: 10.1007/s11263-015-0816-y.

[327] Tero Karras, Samuli Laine, and Timo Aila. A style-based generator architecture for generative adversarial networks. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 4401–4410, 2019.

[328] Hao Li, Asim Kadav, Igor Durdanovic, Hanan Samet, and Hans Peter Graf. Pruning filters for efficient convnets. In *International Conference on Learning Representations*, 2017. URL https://openreview.net/forum?id=rJqFGTslg.

[329] Han Cai, Chuang Gan, Tianzhe Wang, Zhekai Zhang, and Song Han. Once-for-all: Train one network and specialize it for efficient deployment. In *International Conference on Learning Representations*, 2020. URL https://openreview.net/forum?id=HylxE1HKwS.

[330] Emil Julius Gumbel. *Statistical theory of extreme values and some practical applications: a series of lectures*, volume 33. US Government Printing Office, 1954.

[331] Chris J. Maddison, Andriy Mnih, and Yee Whye Teh. The concrete distribution: A continuous relaxation of discrete random variables. In *International Conference on Learning Representations*, 2017. URL https://openreview.net/forum?id=S1jE5L5gl.

[332] Eric Jang, Shixiang Gu, and Ben Poole. Categorical reparameterization with gumbel-softmax. In *International Conference on Learning Representations*, 2017. URL https://openreview.net/forum?id=rkE3y85ee.

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

[334] Kyunghyun Cho, Bart van Merrienboer, Çaglar Gülçehre, Dzmitry Bahdanau, Fethi Bougares, Holger Schwenk, and Yoshua Bengio. Learning phrase representations using RNN encoder-decoder for statistical machine translation. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing, EMNLP 2014, October 25-29, 2014, Doha, Qatar, A meeting of SIGDAT, a Special Interest Group of the ACL*. ACL, 2014. doi: 10.3115/v1/d14-1179. URL https://doi.org/10.3115/v1/d14-1179.

[335] Ilya Loshchilov and Frank Hutter. Decoupled weight decay regularization. In *International Conference on Learning Representations*, 2019. URL https://openreview.net/forum?id=Bkg6RiCqY7.

[336] Yang Song, Jascha Sohl-Dickstein, Diederik P Kingma, Abhishek Kumar, Stefano Ermon, and Ben Poole. Score-based generative modeling through stochastic differential equations. In *International Conference on Learning Representations*, 2021. URL https://openreview.net/forum?id=PxTIG12RRHS.

[337] Midjourney. Midjourney, 2023. URL https://www.midjourney.com/home.

[338] Alex Nichol, Prafulla Dhariwal, Aditya Ramesh, Pranav Shyam, Pamela Mishkin, Bob McGrew, Ilya Sutskever, and Mark Chen. Glide: Towards photorealistic image generation and editing with text-guided diffusion models. *arXiv preprint arXiv:2112.10741*, 2021.

[339] Aditya Ramesh, Prafulla Dhariwal, Alex Nichol, Casey Chu, and Mark Chen. Hierarchical text-conditional image generation with clip latents. *arXiv preprint arXiv:2204.06125*, 1(2):3, 2022.

[340] Gwanghyun Kim, Taesung Kwon, and Jong Chul Ye. Diffusionclip: Text-guided diffusion models for robust image manipulation. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 2426–2435, 2022.

[341] Amir Hertz, Ron Mokady, Jay Tenenbaum, Kfir Aberman, Yael Pritch, and Daniel Cohen-Or. Prompt-to-prompt image editing with cross attention control. *arXiv preprint arXiv:2208.01626*, 2022.

[342] Chenlin Meng, Robin Rombach, Ruiqi Gao, Diederik Kingma, Stefano Ermon, Jonathan Ho, and Tim Salimans. On distillation of guided diffusion models. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 14297–14306, 2023.

[343] Yanyu Li, Huan Wang, Qing Jin, Ju Hu, Pavlo Chemerys, Yun Fu, Yanzhi Wang, Sergey Tulyakov, and Jian Ren. Snapfusion: Text-to-image diffusion model on mobile devices within two seconds. *Advances in Neural Information Processing Systems*, 36, 2024.

[344] Haoxing Chen, Zhuoer Xu, Zhangxuan Gu, Yaohui Li, Changhua Meng, Huijia Zhu, Weiqiang Wang, et al. Diffute: Universal text editing diffusion model. *Advances in Neural Information Processing Systems*, 36, 2024.

[345] Yukang Yang, Dongnan Gui, Yuhui Yuan, Weicong Liang, Haisong Ding, Han Hu, and Kai Chen. Glyphcontrol: Glyph conditional control for visual text generation. *Advances in Neural Information Processing Systems*, 36, 2024.

[346] Alireza Ganjdanesh, Reza Shirkavand, Shangqian Gao, and Heng Huang. Not all prompts are made equal: Prompt-based pruning of text-to-image diffusion models. In *The Thirteenth International Conference on Learning Representations*, 2025. URL `https://openreview.net/forum?id=3BhZCfJ73Y`.

[347] Yuda Song, Zehao Sun, and Xuanwu Yin. Sdxs: Real-time one-step latent diffusion models with image conditions. *arXiv preprint arXiv:2403.16627*, 2024.

[348] Shubham Agarwal, Subrata Mitra, Sarthak Chakraborty, Srikrishna Karanam, Koyel Mukherjee, and Shiv Saini. Approximate caching for efficiently serving diffusion models. *arXiv preprint arXiv:2312.04429*, 2023.

[349] Felix Wimbauer, Bichen Wu, Edgar Schoenfeld, Xiaoliang Dai, Ji Hou, Zijian He, Artsiom Sanakoyeu, Peizhao Zhang, Sam Tsai, Jonas Kohler, et al. Cache me if you can: Accelerating diffusion models through block caching. *arXiv preprint arXiv:2312.03209*, 2023.

[350] Enshu Liu, Xuefei Ning, Huazhong Yang, and Yu Wang. A unified sampling framework for solver searching of diffusion probabilistic models. *arXiv preprint arXiv:2312.07243*, 2023.

[351] Yunsung Lee, JinYoung Kim, Hyojun Go, Myeongho Jeong, Shinhyeok Oh, and Seungtaek Choi. Multi-architecture multi-expert diffusion models. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 38, pages 13427–13436, 2024.

[352] Junhyuk So, Jungwon Lee, Daehyun Ahn, Hyungjun Kim, and Eunhyeok Park. Temporal dynamic quantization for diffusion models. *Advances in Neural Information Processing Systems*, 36, 2024.

[353] Yefei He, Luping Liu, Jing Liu, Weijia Wu, Hong Zhou, and Bohan Zhuang. Ptqd: Accurate post-training quantization for diffusion models. *Advances in Neural Information Processing Systems*, 36, 2024.

[354] Nilesh Prasad Pandey, Marios Fournarakis, Chirag Patel, and Markus Nagel. Softmax bias correction for quantized generative models. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 1453–1458, 2023.

[355] Yuewei Yang, Xiaoliang Dai, Jialiang Wang, Peizhao Zhang, and Hongbo Zhang. Efficient quantization strategies for latent diffusion models. *arXiv preprint arXiv:2312.05431*, 2023.

[356] Xuewen Liu, Zhikai Li, Junrui Xiao, and Qingyi Gu. Enhanced distribution alignment for post-training quantization of diffusion models. *arXiv preprint arXiv:2401.04585*, 2024.

[357] Siao Tang, Xin Wang, Hong Chen, Chaoyu Guan, Zewen Wu, Yansong Tang, and Wenwu Zhu. Post-training quantization with progressive calibration and activation relaxing for text-to-image diffusion models. *arXiv preprint arXiv:2311.06322*, 2023.

[358] Huanpeng Chu, Wei Wu, Chengjie Zang, and Kun Yuan. Qncd: Quantization noise correction for diffusion models. *arXiv preprint arXiv:2403.19140*, 2024.

[359] Xiuyu Li, Yijiang Liu, Long Lian, Huanrui Yang, Zhen Dong, Daniel Kang, Shanghang Zhang, and Kurt Keutzer. Q-diffusion: Quantizing diffusion models. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 17535–17545, 2023.

[360] Sara Elkerdawy, Mostafa Elhoushi, Hong Zhang, and Nilanjan Ray. Fire together wire together: A dynamic pruning approach with self-supervised mask prediction. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 12454–12463, 2022.

[361] Ashish Kumar, Daneul Kim, Jaesik Park, and Laxmidhar Behera. Pick-or-mix: Dynamic channel sampling for convnets, 2024. URL `https://openreview.net/forum?id=Howb7fXB4V`.

[362] Tao Lin, Sebastian U. Stich, Luis Barba, Daniil Dmitriev, and Martin Jaggi. Dynamic model pruning with feedback. In *International Conference on Learning Representations*, 2020. URL https://openreview.net/forum?id=SJem8lSFwB.

[363] Yehui Tang, Yunhe Wang, Yixing Xu, Yiping Deng, Chao Xu, Dacheng Tao, and Chang Xu. Manifold regularized dynamic network pruning. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 5018–5028, 2021.

[364] Asano YM., Rupprecht C., and Vedaldi A. Self-labelling via simultaneous clustering and representation learning. In *International Conference on Learning Representations*, 2020. URL https://openreview.net/forum?id=Hyx-jyBFPr.

[365] Mathilde Caron, Ishan Misra, Julien Mairal, Priya Goyal, Piotr Bojanowski, and Armand Joulin. Unsupervised learning of visual features by contrasting cluster assignments. *Advances in neural information processing systems*, 33:9912–9924, 2020.

[366] Marco Cuturi. Sinkhorn distances: Lightspeed computation of optimal transport. *Advances in neural information processing systems*, 26, 2013.

[367] Emil Julius Gumbel. *Statistical theory of extreme values and some practical applications: a series of lectures*, volume 33. US Government Printing Office, 1954.

[368] Eric Jang, Shixiang Gu, and Ben Poole. Categorical reparameterization with gumbel-softmax. In *International Conference on Learning Representations*, 2017. URL https://openreview.net/forum?id=rkE3y85ee.

[369] Chris J. Maddison, Andriy Mnih, and Yee Whye Teh. The concrete distribution: A continuous relaxation of discrete random variables. In *International Conference on Learning Representations*, 2017. URL https://openreview.net/forum?id=S1jE5L5gl.

[370] Martin Heusel, Hubert Ramsauer, Thomas Unterthiner, Bernhard Nessler, and Sepp Hochreiter. Gans trained by a two time-scale update rule converge to a local nash equilibrium. *Advances in neural information processing systems*, 30, 2017.

[371] Jack Hessel, Ari Holtzman, Maxwell Forbes, Ronan Le Bras, and Yejin Choi. Clipscore: A reference-free evaluation metric for image captioning. *arXiv preprint arXiv:2104.08718*, 2021.

[372] Sadeep Jayasumana, Srikumar Ramalingam, Andreas Veit, Daniel Glasner, Ayan Chakrabarti, and Sanjiv Kumar. Rethinking fid: Towards a better evaluation metric for image generation. *arXiv preprint arXiv:2401.09603*, 2023.

[373] Luping Liu, Yi Ren, Zhijie Lin, and Zhou Zhao. Pseudo numerical methods for diffusion models on manifolds. In *The Tenth International Conference on Learning Representations, ICLR 2022, Virtual Event, April 25-29, 2022*. OpenReview.net, 2022. URL https://openreview.net/forum?id=PlKWVd2yBkY.

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

[375] Ilya Loshchilov and Frank Hutter. Decoupled weight decay regularization. In *7th International Conference on Learning Representations, ICLR 2019, New Orleans, LA, USA, May 6-9, 2019*. OpenReview.net, 2019. URL https://openreview.net/forum?id=Bkg6RiCqY7.

[376] Jonathan Ho and Tim Salimans. Classifier-free diffusion guidance. *CoRR*, abs/2207.12598, 2022. doi: 10.48550/ARXIV.2207.12598. URL https://doi.org/10.48550/arXiv.2207.12598.

[377] Rohit Gandikota, Joanna Materzynska, Tingrui Zhou, Antonio Torralba, and David Bau. Concept sliders: Lora adaptors for precise control in diffusion models. *arXiv preprint arXiv:2311.12092*, 2023.

[378] Tim Dettmers, Mike Lewis, Younes Belkada, and Luke Zettlemoyer. GPT3.int8(): 8-bit matrix multiplication for transformers at scale. In Alice H. Oh, Alekh Agarwal, Danielle Belgrave, and Kyunghyun Cho, editors, *Advances in Neural Information Processing Systems*, 2022. URL https://openreview.net/forum?id=dXiGWqBoxaD.

[379] Ji Lin, Jiaming Tang, Haotian Tang, Shang Yang, Wei-Ming Chen, Wei-Chen Wang, Guangxuan Xiao, Xingyu Dang, Chuang Gan, and Song Han. Awq: Activation-aware weight quantization for on-device llm compression and acceleration. *Proceedings of Machine Learning and Systems*, 6:87–100, 2024.

[380] Tianwei Yin, Michaël Gharbi, Richard Zhang, Eli Shechtman, Fredo Durand, William T Freeman, and Taesung Park. One-step diffusion with distribution matching distillation. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 6613–6623, 2024.