ROBUST EFFICIENT EDGE AI: NEW PRINCIPLES AND FRAMEWORKS FOR EMPOWERING ARTIFICIAL INTELLIGENCE ON EDGE DEVICES

A Dissertation
Presented to
The Academic Faculty

By

Rahul Duggal

In Partial Fulfillment
of the Requirements for the Degree
Doctor of Philosophy in Computer Science

School of Computational Science and Engineering
Georgia Institute of Technology

August 2022

ROBUST EFFICIENT EDGE AI: NEW PRINCIPLES AND FRAMEWORKS FOR EMPOWERING ARTIFICIAL INTELLIGENCE ON EDGE DEVICES

Approved by:

Duen Horng Chau, Advisor
School of Computational Science and Engineering
Georgia Institute of Technology

Ada Gavrilovska
School of Computer Science
Georgia Institute of Technology

Richard Vuduc
School of Computational Science and Engineering
Georgia Institute of Technology

Jimeng Sun
Department of Computer Science
University of Illinois at Urbana-Champaign

Callie Hao
School of Electrical and Computer Engineering
Georgia Institute of Technology

Date Approved: July 14, 2022

Logic will take you from A to B. Imagination will take you everywhere.
Albert Einstein

To my parents Manisha and Vijay Duggal,
for their everlasting support.

ACKNOWLEDGEMENTS

It would be remiss of me to say that this thesis is the fruit of solely my effort, for there are many individuals without whose contributions none of this would be possible.

First and foremost, I would like to thank my advisor Prof. Polo Chau for being an immovable rock of support and encouragement. I will forever be grateful to you for taking me on as your PhD student and the countless hours you've devoted on my journey. Your empathetic personality, an eye for detail, and overall positive attitude has left an indelible impact on my thinking. Although I won't be able to repay you in any way, I'll strive to carry forward and spread these learnings around me.

To my co-advisor Prof. Jimeng Sun, your work ethic has been my north star. Your early insights in the area of efficient deep learning helped me develop my research agenda and have significantly impacted this dissertation. My earliest collaborator at GT—Prof. Richard Vuduc, your clarity of thought, communication and infectious humor is inspiring. Also, your office in Klaus has to be the coolest one I've come across. My committee members Prof. Ada Gavrolovska and Prof. Callie Hao for graciously agreeing to be on my thesis committee. My undergraduate advisors Prof. Anubha Gupta and Prof. Shampa Chakraverty for recognizing and nurturing the young researcher in me.

I've been fortunate to be a part of two phenomenal research groups who provided invaluable feedback for many of my papers and contributed to a very positive work environment. I'd like to especially thank my academic siblings from Sunlab—Tianfan Fu, Siddharth Biswal, Iokeim Perros, Yanbo Xu, Monica Isgut, Sarah Wiegreffe, David Kartchner, Irfan Al-Husseini; and Polo Club—Fred Hohman, Nilaksh Das, Haekyu Park, Jay Wang, Austin Wright, Seongmin Lee, Ben Hoover, Anthony Peng, Omar Shaikh, Kevin Lee. Our labs cannot function without the incredible staff in the School of CSE. I draw special attention to Nirvana Edwards and Arlene Washington Capers who tirelessly resolved many of our administrative issues.

My two internships with the Amazon AWS AI Computer Vision group were instrumental in growing me as a computer vision researcher. My mentors Dr. Hao Zhou and Dr. Shuo Yang, thank you for being incredibly patient and providing exceptional feedback during our almost daily sync ups. My manager Dr. Wei Xia for orchestrating such a smooth internship experience in such extraordinary times. Prof. Stefano Soatto for providing impeccable feedback on my project. Dr. Yuanjun Xiong and Prof. Zhuowen Tu, for helping me develop and position our research ideas. My co-interns Sijie Yan, Yue Zhao, Eric Zhao and the larger AWS Rekognition team for challenging me with your amazing insights during our weekly presentations.

I feel incredibly lucky to be surrounded by an amazing set of friends who made feel at home in the US. Sandesh Adhikary, thank you for hosting me when I first moved to Atlanta. My flatmates Pramod Chunduri, Aditya Venkatraman, Ayush Kumar, Jan Bobek for the joint cooking sessions and fun times at home. Karan Samel, Sunny Dhamnani, Jyoti Narang, Samyak Datta and Devyani Choudhary for orchestrating the outings and road trips. Scott Freitas for the conversations about life, universe and everything. Suchita Jain for being like a little sister, always bubbling with energy. Ishaani for the weekly badminton sessions. Hitesh Dua, Shishir Sharma and Ankit Aggarwal for sharing your wisdom via never ending phone calls from North Carolina, Montreal and San Francisco. The Asha for Education and the Art of Living groups for the weekly run and yoga sessions.
Besides mentors, colleagues and friends, I'd like to specially thank my family.

To Madhvi Dua, my lovely wife, who moved to the US shortly after I started my PhD. Your unconditional support, acute planning skills and an empathetic outlook to life inspires me to improve every single day. Being an impeccable engineer yourself, I could rely on you for in-house design reviews, project/career planning and last minute paper reviews. I am indebted to you for enduring through so many project deadlines with me. So this PhD is as much yours as mine. To my elder brother Gaurav, growing up with you had perhaps unknowingly instilled in me a scientific temper. Looking back, this PhD seems only a natural outcome. To my mother Manisha and father Vijay, I could never be where I am without all your sacrifices, love and support. This PhD is dedicated to you. To our house cat Calcium, thank you for spreading happiness with your cute and funny antics.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

SUMMARY

Deep learning has revolutionised a breadth of industries by automating critical tasks while achieving superhuman accuracy. However, many of these benefits are driven by huge neural networks deployed on cloud servers that consume enormous energy. This thesis contributes two classes of novel frameworks and algorithms that extend the deployment frontier of deep learning models to tiny edge devices, which commonly operate in noisy environments with limited compute footprints:

(1) **New frameworks for efficient edge AI**. We introduce methods that reduce inference cost through filter pruning and efficient network design. CUP presents a new method for compressing and accelerating models, by clustering and pruning similar filters in each layer. CMP-NAS presents a new visual search framework that optimises a small and efficient edge model to work in tandem with a large server model to achieve high accuracy, achieving up to $80\times$ compute cost reduction.

(2) **New methods for robust edge AI**. We Introduce new methods that enable robustness to real-world noise while reducing inference cost. REST, extends the scope of pruning to obtain networks that are $9\times$ more efficient, run $6\times$ faster and robust to adversarial and gaussian noise. HAR generalises the idea of early exiting in multi-branch neural networks to the training phase leading to networks that obtain state-of-the-art accuracy under class imbalance while saving up to 20% inference compute. IMB-NAS optimises neural architectures on imbalanced datasets through super-network adaptation strategies that lead to $5\times$ compute savings compared to searching from scratch.

Our work makes a significant impact to industry and society: CMP-NAS enables the edge deployment use-case for fashion and face retrieval services, and was highlighted at Amazon company-wide to thousands of researchers and developers. REST enables at-home sleep monitoring through a mobile phone and was highlighted by several news media.

CHAPTER 1
INTRODUCTION

Deep learning has pervaded many aspects of our daily lives. Some applications include home health monitoring [1], threat monitoring through video surveillance [2], and social media content moderation [3]. The recent trend in deep learning suggests larger models achieve higher accuracies. Consequently, the field is moving towards larger models that require large amounts of compute resources [4, 5]. However, many of the next decade's research challenges such as in-home health monitoring, and augmented reality require low latency, high throughput prediction on edge devices such as mobile phones. This dichotomy naturally motivates the question: *How to achieve superhuman accuracy with deep neural networks deployed on edge devices?*

One part of the answer lies in making the neural networks *efficient* and amenable to deployment on low-power devices. The other part, making the networks *robust*, stems from the observation that edge deployment often entails *noisy data* arising from measurement errors (*e.g.*, from low-power sensors), the environment itself (*e.g.*, foggy weather), or class imbalance in the data distribution (*e.g.*, the long tail distribution of objects in the real-world). This thesis presents new principles and frameworks for developing robust and efficient neural networks for the edge.

1.1 Thesis Goal: Vision and Motivation

Through my research experience in problems ranging from healthcare, computer vision and security over the past five years, I realize the next decades' research challenge require a fundamental shift from large, accurate neural networks deployed on the cloud to small, efficient neural networks deployed on the edge. This realization stems from an extensive investigation of existing state-of-the-art neural networks. I observe that many of these networks are (1) prohibitively large for edge deployment and (2) susceptible to a large degradation in accuracy in the presence of noise.

A lot of previous research focuses on addressing either of these aspects in isolation. However, effective real-world deployment calls for a joint approach to address these challenges. Filling this research gap, my thesis investigates how to design efficient and robust neural networks for effective deployment in the real world.

1.2 Thesis Overview

Extending the deployment frontier of deep learning models to small edge devices poses two challenges. First, the typical consumer edge devices—mobile phones, smart watches —are resource limited towards compute (small/no GPUs) and energy (limited battery), which calls for developing energy efficient DNNs with a small compute footprint. Second, the edge devices often need to operate on noisy data such as that obtained from low-power sensors (*e.g.*, wearables), or operate on tasks that inherently suffer from class imbalance (*e.g.*, fall detection), which calls for making the DNN resilient to different kinds of real-world noises while ensuring a low compute footprint. My thesis formulates the above challenges into the following two research questions.

1. **How to enable efficiency in DNNs?** Modern DNNs are often too large to be deployable on edge devices. This calls for developing efficient DNNs that are operable in low resource environments. Towards this goal, we answer two questions: (1) Given an accurate DNN, how can we reduce its compute footprint? (2) How to design more efficient DNNs from scratch?

2. **How to jointly enable efficiency and robustness in DNNs?** In addition to efficiency, edge deployment adds the additional challenge of noisy data. What are the different forms of noises faced in the real world? How do we develop DNNs that are resilient to these noise types, while keeping the compute footprint low?

My thesis answers the above two research questions through two corresponding research thrusts. Specifically, I propose: **New Frameworks for Efficient Edge AI** as my answer to enabling efficiency in DNNs; and **New methods for robust edge AI** as my answer to jointly enabling efficiency and robustness in DNNs. Table 1.1 summarizes this mapping and Table 1.2 summarizes the publications arising from this research.

Table 1.1: The two main research questions of this thesis.

Research Question	**Answer** (Example Work)
1. How to tackle the efficiency of DNNs? (Subsection 1.2.1)	New frameworks for efficient edge AI. (CUP: Chapter 3, CMP-NAS: Chapter 4)
2. How to jointly tackle efficiency and robustness in DNNs? (Subsection 1.2.2)	New methods for robust edge AI. (REST: Chapter 5, HAR: Chapter 6), IMB-NAS: Chapter 7)

Table 1.2: Completed work mapped to chapters.

Part I: New frameworks for efficient edge AI

§ 3 CUP: Cluster Pruning for Compressing Deep Neural Networks.

Rahul Duggal, Cao Xiao, Richard Vuduc, Duen Horng Chau, Jimeng Sun.
IEEE International Conference on Big Data (Big Data), 2021.

§ 4 Compatibility Aware Heterogeneous Visual Search.

Rahul Duggal, Hao Zhou, Shuo Yang, Yuanjun Xiong, Wei Xia, Zhuowen Tu, Stefano Soatto.
In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2021.

Part II: New methods for robust edge AI

§ 5 REST: Robust and Efficient Neural Networks for Sleep Monitoring in the Wild.

Rahul Duggal, Scott Freitas, Cao Xiao, Duen Horng Chau, Jimeng Sun.
In *Proceedings of The Web Conference (WWW)*, 2020.

§ 6 HAR: Hardness Aware Reweighting for Imbalanced Datasets.

Rahul Duggal, Scott Freitas, Sunny Dhamnani, Duen Horng Chau, Jimeng Sun.
IEEE International Conference on Big Data (Big Data), 2021.

§ 7 IMB-NAS: Neural Architecture Search for Imbalanced Datasets.

Rahul Duggal, Sheng-Yun Peng, Hao Zhou, Duen Horng Chau.
In submission.

1.2.1 Part I: New Frameworks for Efficient Edge AI.

Deep Neural Networks are often over parameterized function approximators that contain many more weight parameters than the dataset samples they are trained over [6]. This causes the network to become prohibitively large for edge deployment. In this thrust we answer: how to obtain compact neural networks that are amenable to deployment on edge devices? There are two fundamental approaches: (1) *top-down* approach: pruning existing state-of-the-art (and often large) models which we explore in CUP (Chapter 3) in the context of image classification ; and (2) *bottom-up* approach: designing efficient light-weight models from scratch using neural architecture search which we explore in CMP-NAS (Chapter 4) for the task of visual search. Next, we summarize the key ideas and major results of CUP and CMP-NAS.

CUP: Cluster Pruning for Compressing Deep Neural Networks (Chapter 3). In this work, we propose a new method for compressing and accelerating state-of-the-art deep neural networks. At its core, CUP achieves compression by clustering similar filters in each layer. Post clustering, each cluster of filters is replaced by a single cluster representative. The overall workflow of our approach is described in Figure. 1.1. On Imagenet, CUP leads to a 2.47$\times$ FLOPS reduction on ResNet-50 with less than 1% drop in top-5 accuracy.

Notably, in the retrain-free setting, CUP saves over 10 hours of training time on 3 GPUs, in comparison to state-of-the-art methods. CUP, was the first to pose pruning as a filter clustering operation that scales to pruning large models (*e.g.*, ResNet-50) on large datasets.

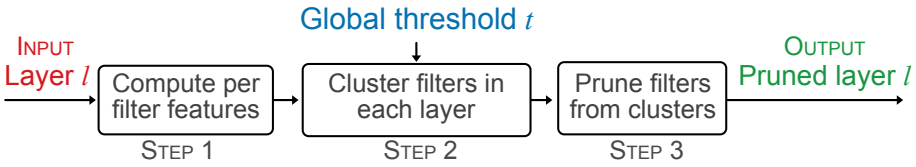


Figure 1.1: An overview of the three step cluster pruning method developed in CUP.

Compatibility-aware Heterogeneous Visual Search (Chapter 4). We tackle the problem of visual search under resource constraints. Existing systems use the same embedding model to compute representations (embeddings) for the query and gallery images. Such systems inherently face a hard accuracy-efficiency trade-off: the embedding model needs to be large enough to ensure high accuracy, yet small enough to enable query-embedding computation on resource-constrained platforms. This trade-off could be mitigated if gallery embeddings are generated from a large model and query embeddings are extracted using a compact model. The key to building such a system is to ensure representation compatibility between the query and gallery models. To this end, we address two forms of compatibility: (1) that enforced by modifying the parameters of each model that computes the embeddings; (2) that obtained by modifying the architectures that compute the embeddings, leading to compatibility-aware neural architecture search (CMP-NAS). Compared to ordinary (homogeneous) visual search using the largest embedding model (paragon), the edge models searched via CMP-NAS achieve 80-fold and 23-fold cost reduction while maintaining accuracy within 0.3% and 1.6% of the paragon for Fashion and Face retrieval.

1.2.2 Part II: New methods for robust edge AI.

Efficiency is one part of the story for edge deployment; The other is robustness. Neural networks deployed on the edge often operate in very noisy environments. How do we develop small models that are robust to these forms of noise? What are the different kinds of noise? We look at two kinds: that arising in the (1) *test data* distribution: due to inaccurate data measurements from low power sensors (*e.g.*, wearables), from the surrounding environment itself (*e.g.*, low lighting, fog, *etc.*). In REST, we tackle test time robustness for the task of sleep staging from EEG data, or (2) *training data* distribution: *e.g.*, class imbalance in the training data. In HAR, we tackle training time robustness manifesting as

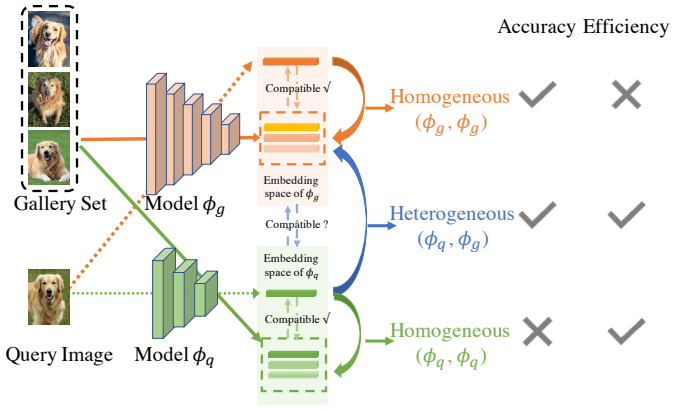


Figure 1.2: An overview of Heterogeneous Visual Search (blue) which uses a large model to compute embeddings for the gallery, and a small model for the query images. This allows high efficiency without sacrificing accuracy, provided that the green and orange embeddings are *compatible*.

class imbalance during image classification on long-tail datasets. In IMB-NAS, we search for efficient neural architectures on long tail datasets. Next, we summarize the key ideas and major results of REST, HAR and IMB-NAS.

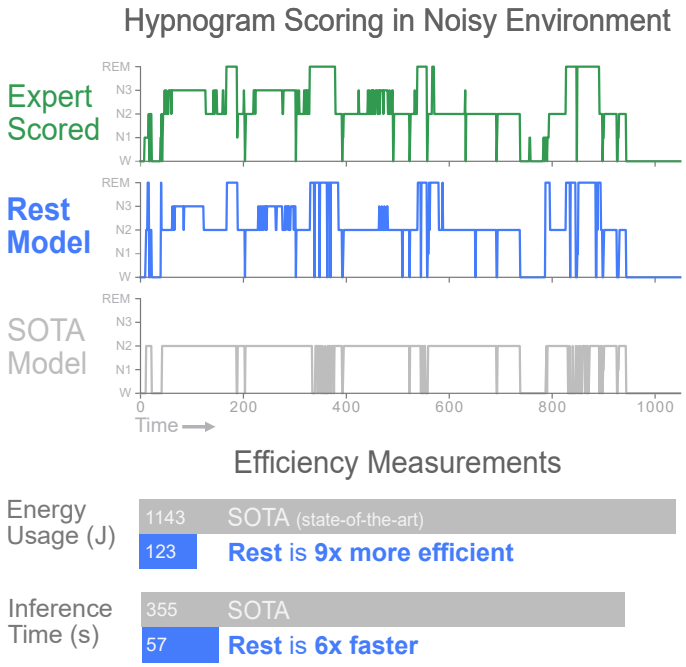


Figure 1.3: Top: we generate hypnograms for a patient in the SHHS test set. In the presence of Gaussian noise, our REST-generated hypnogram closely matches the contours of the expert-scored hypnogram. Hypnogram generated by a state-of-the-art (SOTA) model by Sors et al. [7] is considerably worse. Bottom: we measure energy consumed (in Joules) and inference time (in seconds) on a smartphone to score one night of EEG recordings. REST is 9X more energy efficient and 6X faster than the SOTA model.

REST: Robust and Efficient Neural Networks for Sleep Monitoring in the Wild (Chapter 5). A central tool in diagnosing sleep disorders is the hypnogram—which documents the progression of sleep stages (REM stage, Non- REM stages N1 to N3, and Wake stage) over an entire night (see Figure. 1.3, top). The process of acquiring a hypnogram from raw sensor data is called sleep staging, which is the focus of this work. Traditionally, to reliably obtain a hypnogram the patient has to undergo an overnight sleep study—called polysomnography (PSG)—at a sleep lab while wearing bio-sensors that measure physiological signals. This is both a costly, and invasive process which discourages many people from undergoing this exam. In REST, we develop tiny and robust deep neural networks that can generate a hypnogram (at home) on a mobile phone using EEG signals captured from a wearable sensor. With REST, we aim to overcome two challenges: (1) the model needs to be small enough to be deployable on a mobile phone and (2) robust to diverse forms of noise that arises due to low power EEG sensors and body movement. Indeed, as shown in Figure. 1.3 top, in the presence of noise, the REST generated hypnogram mimics the expert scored one while the sota accuracy degrades rapidly (in grey). Furthermore, as shown in Figure. 1.3 bottom, the REST model is $9\times$ more efficient and runs $6\times$ faster on a mobile.

HAR: Hardness Aware Reweighting for Imbalanced Datasets (Chapter 6). Class imbalance is a significant issue that causes neural networks to underfit to the rare classes. Traditional mitigation strategies include loss reshaping and data resampling which amount to increasing the loss contribution of minority classes and decreasing the loss contributed by the majority ones. However, by treating each example within a class equally, these methods lead to undesirable scenarios where hard-to-classify examples from the majority classes are down-weighted and easy-to-classify examples from the minority classes are up-weighted.We propose the Hardness Aware Reweighting (HAR) framework,which circumvents this issue by increasing the loss contribution of hard examples from both the majority and minority classes. This is achieved by augmenting a neural network with intermediate classifier branches to enable early-exiting during training (see Figure. 1.4). Experimental results on large-scale datasets demonstrate that HAR consistently improves state-of-the-art accuracy while saving upto 20% of inference FLOPS.

IMB-NAS: Neural Architecture Search for Imbalanced Datasets (Chapter 7). Class imbalance is a ubiquitous phenomenon occurring in real world data distributions. To overcome its detrimental effect on training accurate classifiers, existing work follows three major directions: class re-balancing, information transfer, and representation learning. In this paper, we propose a new and complementary direction for improving performance on long tailed datasets—optimizing the *backbone architecture* through neural architecture

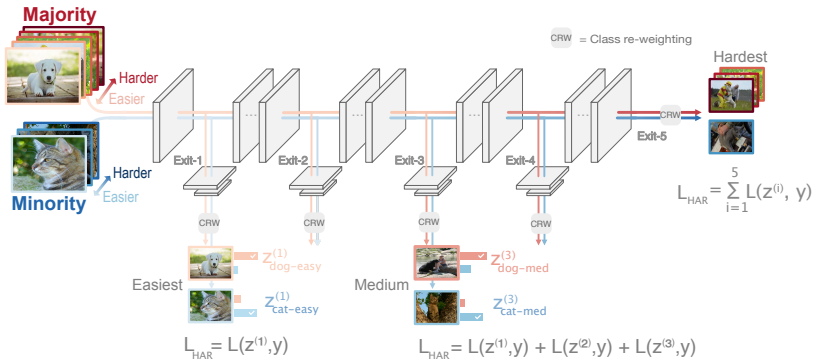


Figure 1.4: Hardness **A**ware **R**eweighting (**HAR**) framework augments a backbone network with auxiliary classifier branches. During training, an example accumulates loss at each branch, until either (a) it is *confidently* and *correctly* classified at a branch, or (b) it reaches the end. A harder example exits later in the network and accumulate higher overall loss.

Table 1.3: Motivation. We sample four architectures A1-A4 from the DARTS search space and train them on balanced (i.e. $1\times$) and imbalanced versions (i.e. $100\times$) of Cifar10 and Cifar100. (**Top**) Two similarly sized architectures (A1,A2) achieve similar accuracy on balanced Cifar10, but differ by 3% in presence of $100\times$ imbalance. (**Bottom**) The larger architecture (A3) outperforms the smaller on (A4) on balanced Cifar100, but under performs by 3.6% in the presence of $100\times$ imbalance. This suggests that an architecture's performance on balanced datasets is not indicative of its performance on imbalanced ones.

Dataset	Model	Flops	Accuracy (%)	
			bal($1\times$)	imbal($100\times$)
Cifar10	A1	410	94.6	77.3
	A2	407	94.7	74.1
Cifar100	A3	400	76.1	39.4
	A4	179	75.0	43.0

search (NAS). We find that an architecture's accuracy obtained on a balanced dataset is not indicative of good performance on imbalanced ones (see Table. 1.3). This poses the need for a full NAS run on long tailed datasets which can quickly become prohibitively compute intensive. To alleviate this compute burden, we aim to efficiently adapt a NAS super-network from a balanced source dataset to an imbalanced target one. Among several adaptation strategies, we find that the most effective one is to retrain the linear classification head with reweighted loss, while freezing the backbone NAS super-network trained on balanced source dataset. We perform extensive experiments on multiple datasets and provide concrete insights to optimize architectures for long tailed datasets.

1.3 Thesis Statement

Deep learning powered services can be deployed on small edge devices that operate in noisy environments with limited compute footprints through:

1. new frameworks that improve efficiency via bottom-up design with neural architecture search, and top-down design with pruning; and
2. new methods that improve test-time robustness with pruning, and training-time robustness with early exiting and neural architecture search.

1.4 Research Contributions & Impact

New algorithms to obtain efficient and robust models.

- Our CUP algorithm (Chapter 3) can prune large deep neural networks trained on ImageNet to more that $2\times$ while reducing training time by over 10 hours to obtain a compact DNN with less than 1% drop in Top-5 accuracy.
- Our CMP-NAS algorithm (Chapter 4) can search for efficient query models, which fitted into a heterogeneous visual search system leads to $80\times$ and $23\times$ compute cost reduction while maintaining accuracy within 0.3% and 1.6% of the paragon for fashion and face image retrieval respectively.
- Our REST algorithm (Chapter 5) produces highly-robust and efficient models that substantially outperform the original full-sized models in the presence of noise. For the sleep staging task over single-channel EEG, the REST model achieves a macro-F1 score of 0.67 vs. 0.39 achieved by a state-of-the-art model in the presence of Gaussian noise while obtaining $19\times$ parameter reduction and $15\times$ MFLOPS reduction on two large, real-world EEG datasets.
- Our HAR algorithm (Chapter 6) endows hardness awareness during the learning process thereby improving state-of-the-art accuracy of networks trained on large imabalanced datasets while saving upto 20% of inference FLOPS.
- Our IMB-NAS algorithm (Chapter 7) searches for efficient network architectures on long tail datasets while saving $5\times$ compute compared to searching from scratch. The searched networks improve accuracy on imbalanced datasets compared to architectures optimized on full balanced datasets.

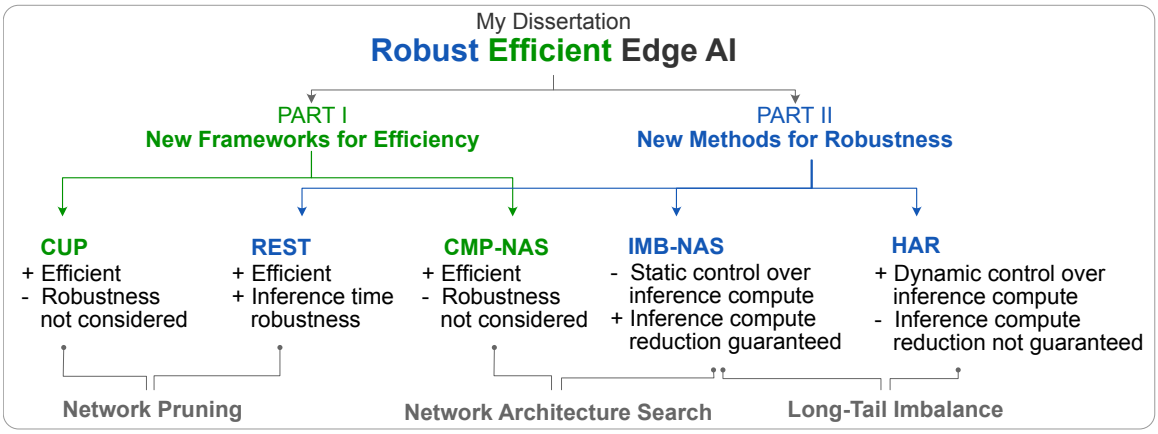


Figure 1.5: Summarizing all algorithmic contributions of this thesis.

Fig. 1.5 summarizes all the algorithmic contributions of this dissertation. The contributed algorithms span three research areas namely network pruning, network architecture search and long-tail imbalance. Additionally, we provide usage guidelines for choosing the appropriate algorithm to design edge models.

New insights and knowledge

- We are among the **first to demonstrate the viability of a home based sleep apnea monitoring and diagnosis system**. Through robust experiments, we show that a small neural network can be trained to be robust to real-world gaussian noise such that it can maintain a reasonable accuracy of 67% whereby the sota accuracy falls to 39%.
- We are the **first to develop a heterogeneous visual search system that extends real world applications** such as video based threat monitoring and image based product search onto edge devices.
- We are the **first to demonstrate** that architectures optimized on fully class balanced datasets are not the optimal ones for imbalanced datasets.

Industry & Media Coverage.

1. The CMP-NAS system, developed during my internship with Amazon was **showcased to thousands of developers** in the company-wide all-hands meeting. It was also the subject of a patent application.

2. CMP-NAS and its follow up work REG-NAS resulted in an **Amazon post-internship fellowship** to fund my research until graduation.

3. Our work on REST was **highlighted by several news media outlets** for its ability to accurately monitor sleep in the wild.

Open source software with ready implementations of baselines and proposed work.

- Our REST system was deployed to a Pixel 2 smartphone through an android application to demonstrate $17\times$ energy reduction and $9\times$ faster inference compared to the state of the art.

- The code packages for CUP and REST are publicly released through Github. Their modular construction supports easy extension to the latest pruning methods.

CHAPTER 2
RELATED WORK

This section briefly reviews related works. I focus on two areas relevant to this thesis: (1) Enabling efficiency in Deep Neural Networks; and (2) Making DNNs resilient to noise.

2.1 Enabling Efficiency in Deep Neural Networks.

At a high level, there are two approaches—*static* and *dynamic*—for enabling efficiency in neural networks. The latter differs from the former in that the compute footprint of the DNN can be dynamically varied. Within static methods, prior art can be classified into five directions: pruning [8], neural architecture search [9], quantization [10], low rank approximation [11] and knowledge distillation [12]. We cover the first two in the subsequent sections. Among dynamic methods, we review literature on early exiting through multi-branch neural networks.

2.1.1 Pruning Deep Neural Networks

At a high level, pruning methods can be categorized based on whether they lead to *unstructured* [13, 14, 15, 16] or *structured* [17, 18, 19, 20, 21] sparsity in the pruned network's weights. The latter reaps the benefits of pruning (*e.g.*, lower flops, faster inference) through a matrix reshaping operation completely avoiding the need of custom hardware as typically required by the former. Within structured pruning, a promising research direction is *channel pruning* where the aim is to prune entire filters. Existing channel pruning algorithms primarily differ in the criterion used for identifying pruneable filters. Examples of such criterions include pruning filters; with smallest L1 norm of incoming weights [17]; with largest average percentage of zeros in their activation maps [18]; using structured regularization [22, 19]; or with least discriminative power [21]. A drawback of these methods [17, 20] is that the number of filters to prune in each layer is a hyper-parameter leading to a combinatorial search space. We tackle this challenge in CUP.

2.1.2 Neural Architecture Search

Neural Architecture Search refers to the process of automatically searching for the network topology that maximizes accuracy, typically under a constrained compute budget. Most NAS strategies consist of three components: search space, search strategy and a search

reward. A search space refers to the design space of all architectures including operations (such as convolution, pooling, etc) and the network topoogy. Popular search spaces include the NasNet [23], DARTS [24], MobileNet [25], FBNet [26]. A recent comparison of the different search spaces is by Radovich et al [27]. The search strategy defines how to effectively explore the search space. This is important since most search space encode trillions of architectures, making random search infeasible. Popular search strategies include reinforcement learning [23] and evolutionary search [28]. The search reward refers to the optimization target e.g. accuracy of the neural network. Fast evaluation of the search reward is very important for the search to be feasible. Recent work in this area include low fidelity estimation [23, 28], extrapolation [29] and weight sharing [25, 26, 24].

2.1.3 Early Exiting with Multi-branch Neural Networks.

Research in this area aims to endow a neural network with auxiliary classifier branches (or early-exits) that allow for obtaining predictions from intermediate locations along the backbone DNN. This leads to advantages such as, saving inference time compute [30, 31, 32]; mitigating the vanishing gradient problem as in Inception networks [33]; while also affording a natural application to computing paradigms such as fog computing and 5G [34, 35]. The important research challenges for multi-branch DNNs (see review paper [36]) stem from questions such as: where to place the early-exits, what criterion to use for early-exiting and how to define the training objective. Many works place the early-exits after each block of layers which enables large savings of inference FLOPS [30, 31, 32]. The exit criteria range from early-exiting based on low entropy predictions as in BranchyNet [37] to early exiting based on prediction confidence as in MSDNet [30]. Training objectives include ones that distill knowledge from later exits to earlier ones [31, 38] or ensemble predictions from multiple branches to improve adversarial robustness [32].

2.2 Making Neural Networks Resilient to Noise.

Noise can arise in many forms. For this thesis, we focus on two types (1) *test time* noise that manifests as adversarial and gaussian noise; and (2) *training time* noise manifesting as class imbalance. We cover previous work on enabling robustness against these noise types.

2.2.1 Enabling Robustness to Adversarial and Gaussian Noise

Adversarial robustness seeks to ensure that the output of a neural network remains unchanged under a bounded perturbation of the input; or in other words, prevent an adveresary from maliciously perturbing the data to fool a neural network. Adversarial deep learning

was popularized by [39], where they showed it was possible to alter the class prediction of deep neural network models by carefully crafting an adversarially perturbed input. Since then, research suggests a strong link between adversarial robustness and noise robustness [40, 41, 42]. In particular, [40] found that by performing adversarial training on a deep neural network, it becomes robust to many forms of noise (e.g., Gaussian, blur, shot, etc.). In contrast, they found that training a model on Gaussian augmented data led to models that were less robust to adversarial perturbations. We build upon this finding of adversarial robustness as a proxy for noise robustness and improve upon it through the use of spectral regularization; while simultaneously compressing the model to a fraction of its original size for mobile devices.

2.2.2 Overcoming Class Imbalance

The techniques for overcoming class imbalance techniques fall into the following three categories. (1) *Loss rebalancing:* These methods reweight the loss contribution of each example such that the loss for minority classes is upweighted, while that of the majority classes is downweighted. The weighting scheme itself can be uniform across all the examples within a class [43, 44] or can be more fine-grained *i.e.*, specific to each example in consideration [45, 46, 47, 48]. The uniform reweighting techniques include reweighting based on inverse class frequency [43, 44] or based on the effective number of samples in each class [43]. On the other hand, fine-grained approaches include Focal loss [45], which reweights based on sample hardness or recent studies [46, 47, 48] that employ meta-learning to perform sample reweighting. (2) *Data resampling:* These methods either repeatedly sample examples from the minority class (over-sampling) [49, 50, 51, 52] or discard samples from the majority class (under-sampling) [53, 54, 55, 56]. Popular strategies include SMOTE that over-samples the minority class through linear interpolation [49]; or [53] that under-samples the majority class by clustering and replacing the majority class examples by a few anchor points. With neural networks, over-sampling generally creates redundancy and risks over-fitting to the rare classes, while, under-sampling is susceptible to losing information from the majority classes [57]. (3) *Training strategies:* These methods modify the training procedure to mitigate the problem of class imbalance. For instance, LDAM [58], introduces a delayed reweighting scheme wherein, class reweighting is applied after a few epochs of training. Kang et al. [59] show improvement through a two-step training process which decouples representation and classifier learning. Recently, BBN [60] show that gradually shifting emphasis from class sampling to reverse sampling helps improve accuracy.

Part I

New Frameworks for Efficient Edge AI

Overview

My thesis begins by tackling the fundamental challenge for edge deployment: *efficiency* of Deep Neural Networks. Our first scenario deals with the top-down approach, wherein given a source model, we'd like to obtain a more efficient target model that achieves the same accuracy as the source, but with lesser compute. In CUP, we develop a new approach to prune or delete redundant filters from the source model to obtain the target.

> **Chapter 3. CUP: Cluster Pruning for Compressing Deep Neural Networks.** Rahul Duggal, Cao Xiao, Richard Vuduc, Duen Horng Chau, Jimeng Sun. *IEEE International Conference on Big Data (Big Data), 2021* https://ieeexplore.ieee.org/document/9671980

The techniques developed in CUP are applicable only when we have an accurate source model, which may not always be the case. Such scenarios motivate the *bottom-up* design of efficient architectures using neural architecture search. In CMP-NAS, we search for the "most compatible" edge models for a heterogeneous visual search system. The resulting system achieves upto $80\times$ compute reduction with less than 0.3% accuracy drop.

> **Chapter 4. Compatibility-aware Heterogeneous Visual Search.**
> Rahul Duggal, Hao Zhou, Shuo Yang, Yuanjun Xiong, Wei Xia, Zhuowen Tu, Stefano Soatto. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2021. https://openaccess.thecvf.com/content/CVPR2021/papers/Duggal_Compatibility-Aware_Heterogeneous_Visual_Search_CVPR_2021_paper.pdf

CHAPTER 3
CUP: CLUSTER PRUNING FOR COMPRESSING DEEP NEURAL NETWORKS

We propose CUP, a new method for compressing and accelerating deep neural networks. At its core, CUP achieves compression by clustering and pruning similar filters in each layer. For clustering, CUP uses hierarchical clustering which allows for an elegant parameterization of model capacity through a single hyper-parameter t. We observe that by increasing t, CUP can dynamically reduce model capacity through non-uniform layer-wise pruning leading to two advantages. First, CUP can effectively compress a model to within the desired compute budget through a simple line-search on t. On Imagenet, CUP leads to a $2.47\times$ FLOPS reduction on Resnet-50 with less than 1% drop in top-5 accuracy. Second, through a simple extension, CUP can obtain the pruned model in a single training pass leading to large savings in training time. We call the retrain free version as CUP-RF. Notably, in the retrain-free setting, CUP-RF saves over 10 hours of training time on 3 GPUs, in comparison to state-of-the-art methods. The code for CUP is open sourced[1].

3.1 Introduction

Neural network compression is a critical enabler for the deployment of powerful deep neural networks (DNNs) on the edge. There are several ways to compress a DNN, including pruning [8, 17, 21], low rank approximation [61, 62], knowledge distillation [12, 63] and quantization [64, 65]. In this paper, we focus on the problem of channel pruning which, in a nutshell, aims to delete the "unimportant" filters of a neural network.

A typical channel pruning pipeline consists of three steps [8]: (1) train the target DNN for some task, *e.g.*, image classification; (2) identify and delete unimportant filters based on an importance criterion; (3) retrain the pruned network to recover accuracy lost due to pruning. This pipeline presents two challenges: *C1–non-uniform pruning*: determining the optimal layerwise pruning amount in step 2 is a combinatorial problem, and intractable for modern DNNs with hundreds of layers; and *C2–long runtime*: the retraining performed in step 3 slows down the pruning pipeline. To tackle *C1*, a line of work [17, 18, 19] prunes the network uniformly across each layer (*e.g.*, delete 50% filters in each layer), but this has shown to be sub-optimal [66]. Other works use heuristics that are either computationally expensive [67, 17] or involve many hyper-parameters, which are difficult to tune [66]. For tackling *C2*, recent methods [68, 69] modify the pruning pipeline by interleaving the

[1] `https://github.com/duggalrahul/CUP_Public`

pruning and training steps (*i.e.*, merging steps 1 and 2), thereby eliminating the need for retraining—we refer to these as *retrain-free* methods.

With CUP, we enable layer-wise non-uniform pruning (addressing *C1*) whilst introducing only a single hyper-parameter t. At its core, CUP employs hierarchical clustering to cluster similar filters in each layer. Pruning is then achieved by replacing each cluster by a representative filter. A key advantage of our method is that the clustering strategy used offers a principled way to determine the appropriate number of clusters in each layer. Empirically, we find that t allows for a smooth parameterization of the pruning amount. We leverage this observation to extend CUP to the retrain-free setting (addressing *C2*). Essentially, by gradually increasing t during the initial training phase, CUP-RF (RF for retrain-free) can incrementally prune a target model within one training pass. This leads to large savings in training time, *e.g.*, saving over 14 hours while training a ResNet-50 on ImageNet.

To summarize, our contributions in this paper are 1) We propose CUP as a method for compressing deep neural networks with the benefit of enabling non-uniform pruning through a single hyper-parameter *t*. 2) We extend CUP to CUP-RF whereby filters are pruned in the initial training pass itself resulting in large savings of time cost during pruning. 3) We comprehensively compare our methods to the state-of-the-art methods on large datasets (*e.g.*, Imagenet).

3.2 Related Work

Neural Network Compression is an active area of research wherein the goal is to reduce the memory, flops, or inference time of a neural network while retaining its performance. Prior work in this area can be broadly classified into the following four categories:

(1) *Low-Rank Approximation*: where the idea is to replace weight matrices (for DNN) or tensors (for CNNs) with their low-rank approximations obtained via matrix or tensor factorization [61, 62]. (2) *Quantization* wherein compression is achieved by using lower precision (and fewer bits) to store weights and activations of a neural network [64, 65]. (3) *Knowledge Distillation* wherein compression is achieved by training a small neural network to mimic the output (and/or intermediate) activations of a large network [12, 63]. (4) *Pruning* wherein compression is achieved by eliminating unimportant weights [8, 17, 21]. Methods in these categories are considered orthogonal and are often combined to achieve more compact models.

Within pruning, methods can either be structured or unstructured. Although the latter type i.e. unstructured pruning usually achieves higher compression, it seldom results in

flops and inference time reduction. Keeping this in mind, we adopt *structured pruning* for CUP. Within structured pruning, a promising research direction is *channel pruning* where the aim is to prune entire filters. Existing channel pruning algorithms primarily differ in the criterion used for identifying pruneable filters. Examples of such criterion include pruning filters, with smallest L1 norm of incoming weights [17], largest average percentage of zeros in their activation maps [18], using structured regularization [22, 19] or with least discriminative power [21]. A major drawback of these methods [17, 20] is that the number of filters to prune in each layer is a hyper-parameter. Determining the optimal hyper-parameters is an issue since modern neural networks can contain hundreds of layers. Recent works [68, 69] avoid the issue of per layer hyper-parameters by doing uniform pruning across layers e.g. Prune 50% neurons from each layer. This, however, can be a restrictive setting since some layers are less sensitive to pruning than other layers and can be pruned more aggressively [17]. CUP differs from prior work in that it enables *non uniform pruning* while introducing a *single hyper-parameter*.

Another distinguishing factor of CUP is that it can speed up training. Traditionally, channel pruning has employed a three-step regime involving (1) training the full model (2) pruning the full model to desired sparsity and (3) retraining the pruned model. Due to phases (1) and (3), the total time for obtaining a pruned model increases $1.5 - 2\times$. For example, the time to obtain a compressed Resnet-50 on Imagenet increases from 3 days to 5 days on 3 GPU's. Some recent works [68, 69, 70, 71] address the issue of increased training time by doing away with the retraining phase altogether. In this paper, we refer this as the **retrain-free setting**. In practice, the retrain-free setting significantly reduces the total time for obtaining a compressed model. With CUP-RF (for CUP retrain free), we are able to further reduce the training time by iteratively pruning the model during the training phase itself. The gradual reduction of model capacity during the initial training phase directly manifests in decreased training time. In particular, we demonstrate savings of up to 14 hours while training a Resnet-50 on Imagenet. In the next section, we introduce the notation we use in the rest of the paper.

3.3 Problem Setup

3.3.1 Notation

To maintain symbolic consistency, we use a capital symbol with a tilde, like $\widetilde{W}$, to represent tensors of rank 2 or higher (i.e. matrices and beyond). A capital symbol with a bar, like $\overline{F}$, represents a vector while a plain and small symbol like b represents a scalar. A lower subscript indexes a tensor, so, $\widetilde{F}_{i,j}$ indicates the element at position (i, j) while a superscript

with parenthesis like $\widetilde{W}^{(l)}$ maps the quantity $\widetilde{W}$ to layer l of the neural network.

3.3.2 Intuition

In this section we build the intuition behind clustering filters based on features derived from both incoming and outgoing weights. Consider one simple case of a fully connected neural network with n, m filters in layers $l-1$ and l, respectively. Here layer l is parameterized by weights $\widetilde{W}^{(l)} \in \mathbb{R}^{m \times n}$ and bias $b^{(l)} \in \mathbb{R}^m$. Within this layer, filter i performs the following nonlinear transformation.

$$\overline{O}_i^{(l)} = \sigma(\widetilde{W}_{i,:}^{(l)} \overline{O}^{(l-1)} + b_i^{(l)}) \tag{3.1}$$

Where $\overline{O}^{(l-1)}$ is the output from layer $l-1$, $\widetilde{W}_{i,:}^{(l)}, b_i^{(l)}$ are the i^{th} row and element of the weight matrix and bias vector respectively. These also constitute the set of incoming weights to filter i. Finally σ is a non linear function such as RELU.

Given this notation, a filter k in layer $l+1$ receives the combined contribution from filters i and j equal to $\widetilde{W}_{ki}^{(l+1)} \overline{O}_i^{(l)} + \widetilde{W}_{kj}^{(l+1)} \overline{O}_j^{(l)}$ where $\widetilde{W}_{ki}^{(l+1)}$ and $\widetilde{W}_{kj}^{(l+1)}$ are the weights from filter i to filter k and from j to k, respectively. This is illustrated in the top of figure 3.1.

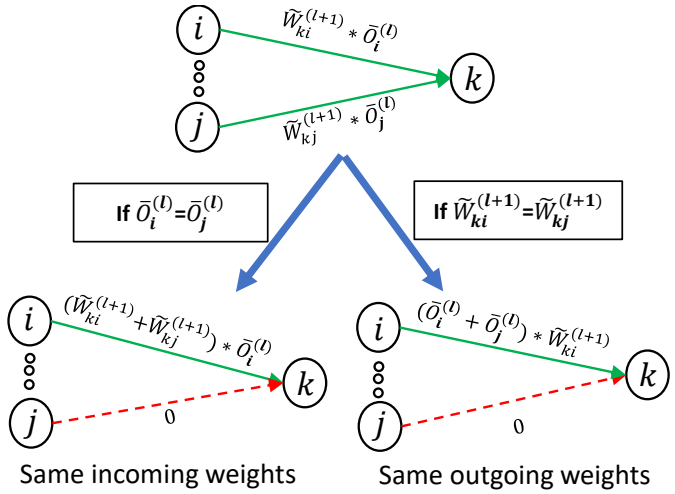


Figure 3.1: Intuition for using input and output weight connections. The top diagram illustrates original neural net structure. The bottom left diagram illustrates the case with same input weights. The bottom right diagram illustrates the case with same output weights

Inspired from these idealized scenarios, we propose to identify and prune similar filters based on features derived from both incoming and outgoing weights.

Depending on the input and output weight values, the following two cases can arise

- **If the input weights of filter i and j are same** [72] (i.e., $\widetilde{W}_{i,:}^{(l)} = \widetilde{W}_{j,:}^{(l)}$ and $b_i^{(l)} = b_j^{(l)}$) or equivalently $\overline{O}_i^{(l)} = \overline{O}_j^{(l)}$. Then the combined contribution can be provided by a

single filter with output connection weight $\widetilde{W}_{ki}^{(l+1)} + \widetilde{W}_{kj}^{(l+1)}$ and the other filter can be pruned. This case corresponds to clustering and pruning similar filters i and j based on input weights and is illustrated in figure 3.1 bottom left.

- **If the output weights of filters i and j are same** (i.e. $\widetilde{W}_{ki}^{(l+1)} = \widetilde{W}_{kj}^{(l+1)}$) Then combined contribution can be provided by an equivalent filter computing $\overline{O}_i^{(l)} + \overline{O}_j^{(l)}$ and the other filter can be pruned. This case corresponds to clustering and pruning similar filters i and j based on output weights and is illustrated in figure 3.1 bottom right.

3.4 The CUP framework

The CUP pruning algorithm is a three-step process and is outlined in Fig. 3.2. The first step computes features that characterize each filter. These features are specific to the layer type (fully connected or convolutional) and are computed from the incoming and outgoing weight connections. The second step clusters similar filters based on the features computed previously. The third and last step chooses a single representative filter from each cluster and prunes all others.

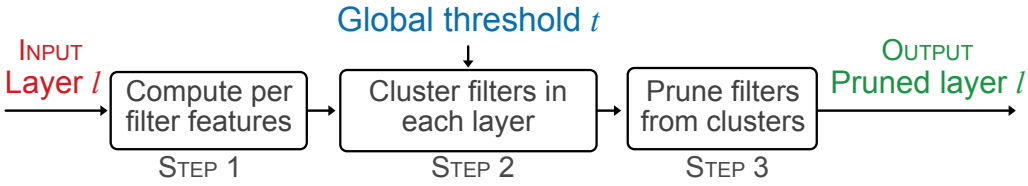


Figure 3.2: Three steps of the Cluster Pruning (CUP) algorithm.

3.4.1 Compute per-filter features (step 1)

The first step of CUP computes features that characterize each filter. These features are computed using both the *incoming* and *outgoing* connections of a filter. For further discussion, we assume layers $l-1, l$ and $l+1$ of the neural network contain n, m and p filters respectively.

Fully Connected Layers (Fig. 3.3a). The l^{th} fully connected layer is parameterized by weights $\widetilde{W}^{(l)} \in \mathbb{R}^{m \times n}$ and bias $\overline{B}^{(l)} \in \mathbb{R}^m$. For neuron i within this layer, we define it's

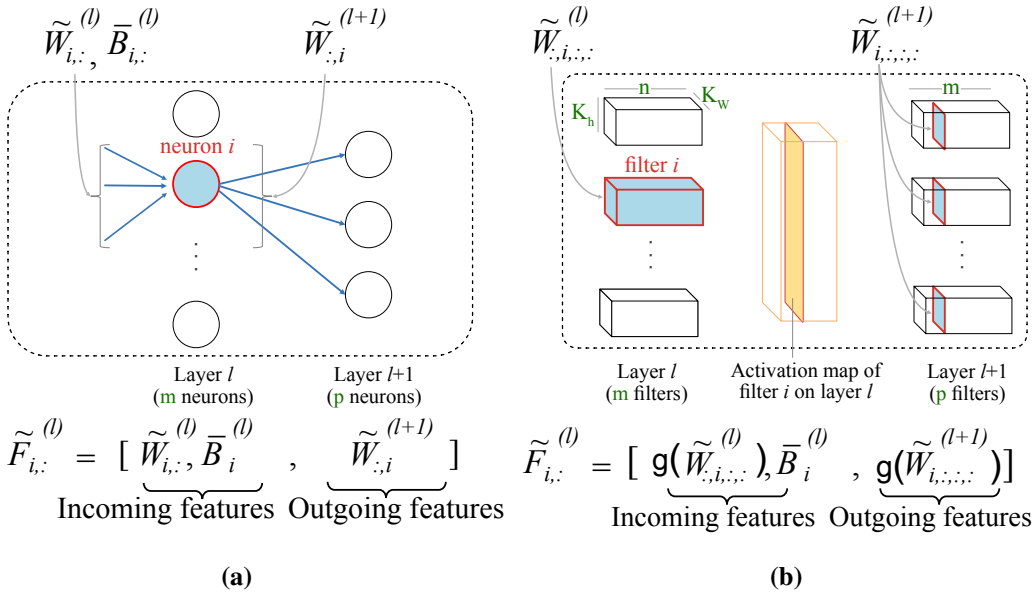


Figure 3.3: Computing features from the incoming and outgoing weights of (a) fully-connected layer, (b) convolution layer.

feature set $\widetilde{F}_{i,:}^{(l)} \in \mathbb{R}^{n+p+1}$ as

$$\widetilde{F}_{i,:}^{(l)} = \text{concat}(\underbrace{\widetilde{W}_{i,:}^{(l)}, \overline{B}_i^{(l)}}_{\text{Incoming features}} , \underbrace{\widetilde{W}_{:,i}^{(l+1)}}_{\text{Outgoing features}}), \tag{3.2}$$

where concat concatenates two vectors into one.

Convolutional Layers (ref. Fig. 3.3b). The l^{th} convolutional layer is completely parameterized by the 4-D weight tensor $\widetilde{W}^{(l)} \in R^{n \times m \times k_h \times k_w}$ and the bias vector $\overline{B}^{(l)} \in \mathbb{R}^m$. The four dimensions of $\widetilde{W}^{(l)}$ correspond to - number of input channels (n), number of filters in layer l (m), height of filter (k_h) and width of filter (k_w). For filter i within this layer, we define its feature set $\widetilde{F}_{i,:}^{(l)} \in \mathbb{R}^{n+p+1}$, as

$$\widetilde{F}_{i,:}^{(l)} = \text{concat}(\underbrace{g(\widetilde{W}_{:,i,:,:}^{(l)}), b_i^{(l)}}_{\text{Incoming features}}, \underbrace{\widetilde{W}_{i,:,:,:}^{(l+1)}}_{\text{Outgoing features}}), \tag{3.3}$$

$$\text{where } g(\widetilde{X}_{:,:,:}) = [\|\widetilde{X}_{1,:,:}\|_F, \ldots, \|\widetilde{X}_{C,:,:}\|_F]. \tag{3.4}$$

Here $g : \mathbb{R}^c \times \mathbb{R}^d \times \mathbb{R}^e \rightarrow \mathbb{R}^c$ computes the channel-wise frobenius norm of any arbitrary 3D tensor $\widetilde{X}$.

3.4.2 *Cluster filters in each layer (step 2)*

Given feature vectors $\widetilde{F}_{i,:}^{(l)}$ for each filter i in layer l, step 2 clusters filters within a layer using agglomerative hierarchical clustering[73, Chapter 15]. This specific choice of clustering affords a **key benefit**: The number of clusters in each layer can be *jointly* controlled using a single hyper-parameter *t*. In contrast some recent works use other clustering techniques such as K-means++ [74] or spectral clustering [75]. These works face the combinatorial challenge of deciding the appropriate number of clusters in each layer and are limited to uniform pruning.

With hierarchical clustering, the clustering operation for layer l begins by building a weighted binary tree representation (dendrogram) for that layer. Assume that layer l contains m filters. The algorithm starts off with m clusters $\mathbb{C}_i^{(l)} \forall i \in [1, m]$ i.e. each filter is a separate cluster. Then it iteratively builds the tree by merging two closest clusters as per the *Wards variance minimization* criterion. The criterion specifies to merge two clusters $\mathbb{C}_p^{(l)}, \mathbb{C}_q^{(l)}$ that lead to the least decrease in intra-cluster variance over all possible pairings of clusters in $\mathbb{C}^{(l)}$. The output of this phase is a weighted binary tree, or dendrogram, whose each non-leaf node specifies a cluster of filters while the edge weights encode the distance or dissimilarity between its children.

Determining the number of clusters: After constructing the dendrograms for each layer, we use edge weights to jointly determine the number of clusters in that layer. Specifically, the dendrograms for all layers are chopped at the same height t which is a hyper-parameter. The higher the value of t, fewer the number of clusters. Since CUP replaces each cluster with a filter (presented in the next section), increasing t ultimately leads to fewer remaining filters, or higher compression. To summarize, the output of step 2 is a set of $n^{(l)}$ clusters $\mathbb{C}^{(l)}$ for each layer l, such that $|\mathbb{C}^{(l)}| = n^{(l)}$.

3.4.3 *Prune filters from each cluster (step 3)*

The third and last step chooses the representative filter from each cluster and prunes all others. Given the set of filter clusters $\mathbb{C}^{(l)}$ for layer l, we formulate pruning as a subset selection problem. The idea is to select the most representative subset of filters $\mathbb{S}_r^{(l)}$ from each filter cluster $\mathbb{C}_r^{(l)} \in \mathbb{C}^{(l)}$. Pruning then corresponds to replacing all filters in $\mathbb{C}_r^{(l)}$ by $\mathbb{S}_r^{(l)}$. Motivated by prior work, several subset selection criterion can be formulated as below.

- **Norm based** : [17] prune filters based on the $l1$ norm of incoming weights. This criterion amounts to selecting the top $k\%$ filters having the highest feature norm as the cluster representative.

- **Zero activation based** : [18] prune filters based on the average percentage of zeros (ApoZ) in their activation map when evaluated over a held-out set. This criterion amounts to choosing the top $k\%$ filters having least ApoZ as the cluster representative.

- **Activation reconstruction based** : [20] prunes filters based on its contribution towards the next layer's activation. This criterion amounts to choosing the top $k\%$ filters having maximum contribution.

In our work, we use a norm based criterion to select a subset $\mathbb{S}_r^{(l)}$ from a cluster of filters $\mathbb{C}_r^{(l)}$ with filter i having features $\widetilde{F}_{i,:}^{(l)}$. This criterion is described through the equation

$$\mathbb{S}_r^{(l)} = \underset{i \in \mathbb{C}_r^{(l)}}{\text{argmax}} \| \widetilde{F}_{i,:}^{(l)} \|_2, \tag{3.5}$$

where argmax implies that we select a single representative neuron from each cluster. Thus, post pruning, the number of clusters in layer l equals the number of remaining filters.

*3.4.4 Extension to retrain free setting (*CUP-RF*)*

Similar to previous methods, CUP achieves compression through a three step pipeline involving training, pruning and retraining. However, with a slight modification, CUP can completely avoid any fine-tuning whatsoever. The modified algorithm is termed CUP-RF for CUP "Retrain-Free". The idea is to gradually reduce the model capacity during the initial training phase. This is achieved by calling CUP at the beginning of each epoch, with a monotonically increasing schedule for t. We find that the following linear schedule for $t(e)$ (value of t at epoch e) suffices for good performance.

$$t(e) = k.e + b \tag{3.6}$$

Here k, b are hyper-parameters controlling the slope and offset of the linear pruning schedule and are determined through a linesearch.

3.5 Experiments

In this section, we evaluate the proposed method CUP and it retraining free variation CUP-RF on three datasets of increasing complexity - MNIST, CIFAR and ImageNet. We compare our methods against recent work on several efficiency metrics. Then we perform ablation studies to demonstrate the robustness of CUP-RF to hyper-parameter choices. Finally, we present qualitative insights into why and how CUP generalizes prior work.

3.5.1 Datasets

We evaluate our methods against prior art on three datasets.

- **MNIST** [76] : This dataset consists of 60,000 training and 10,000 validation images of handwritten digits between 0-9. The images are greyscale and of spatial size 28×28.
- **CIFAR-10** [77] : These datasets consist of 50,000 training and 10,000 validation images from 10 classes. The images are 3 channel RGB, of spatial size 32×32.
- **Imagenet 2012** [78] : This dataset consists of 1.28 million images from 1000 classes in the training set and 50,000 validation images. The images are 3 channel RGB and are re-scaled to spatial size 224×224.

3.5.2 Training details, base models & evaluation metrics

For MNIST and CIFAR datasets, we reuse the training hyper-parameter settings from [19]. All networks are trained using SGD with batch size - 64, weight decay - 10^{-4}, initial learning rate - 0.1 which is decreased by a tenth at $1/2$ and $3/4$ the number of total epochs. On MNIST, we train for 30 epochs while on CIFAR 10 the networks are trained for 160 epochs. On Imagenet, we train with a batch size of 256 for 90 epochs. The initial learning is set to 0.1. This is reduced by a tenth at epochs 30 and 60. We use a weight decay of 10^{-4}. After compression, the model is retrained with a tenth of the initial learning rate while other hyper-parameters remain the same. The baseline models are described next.

- **ANN**: We use the multiple layer perceptron with 784-500-300-10 filter architecture in [19, 22]. This is trained upto 98.63% accuracy.
- **VGG-16** [79]: This the standard VGG architecture with batchnorm layers after each convolution layer. It is trained up to 93.64% on CIFAR-10.
- **Resnet-56** [80]: We use the standard Resnet model which is trained up to 93.67% on CIFAR-10.
- **Resnet-{18,34,50}** [80]: We use the official pytorch implementations for the three models. The baselines models have 69.87%, 73.59% and 75.86% Top-1 accuracies on Imagenet respectively. For Resnets, we only prune the first layer within each bottleneck.

We benchmark CUP against several recent state of the art compression methods. Whenever possible, relevant results are quoted directly from the referenced paper. For [22] we use our implementation. The compression metrics are reported as $M = \frac{P_{base}}{P_{compressed}}$ where P can be one of the following measured attributes (MA).

- Parameter reduction (PR) : The MA is number of weights.
- Flops reduction (FR) : The MA is number of multiply and add operations to score one input.
- Speedup: The MA is time to score one input on the CPU.

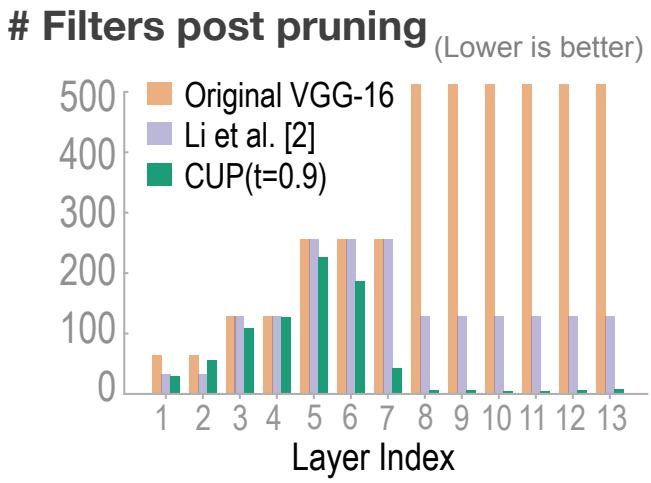


Figure 3.4: The number of filters remaining in each layer post pruning a VGG-16 on CIFAR-10.

3.5.3 Verifying two key benefits of pruning with CUP

Single hyper-parameter control. In Figure 3.4, we plot the number of remaining filters in each layer after pruning a VGG-16 on Cifar-10 with CUP ($t = 0.9$). Using a single hyper-parameter t, CUP accomplishes non-uniform pruning across the layers (see green bars). As t increases, CUP offers a desirable, largely monotonic effect on: (1) test accuracy reduction; (2) parameter reduction; (3) flops reduction; and (4) CPU wall-clock speedup, as show in Figures 3.5a–d.

Training time speedup. Table 3.1 shows our approach (bold font) achieves the shortest training time, best top-1 accuracy, and the most flop reduction, in both the *retrain-allowed* (marked with ✓) and the *retrain-free* settings (✗), for a ResNet-50 trained on ImageNet. In the *retrain-free* setting (Table 3.1, bottom row), CUP-RF saves 14 hours when compared to that of the uncompressed model (51.6 v.s. 66 hours). All the *retrain-free* methods generally

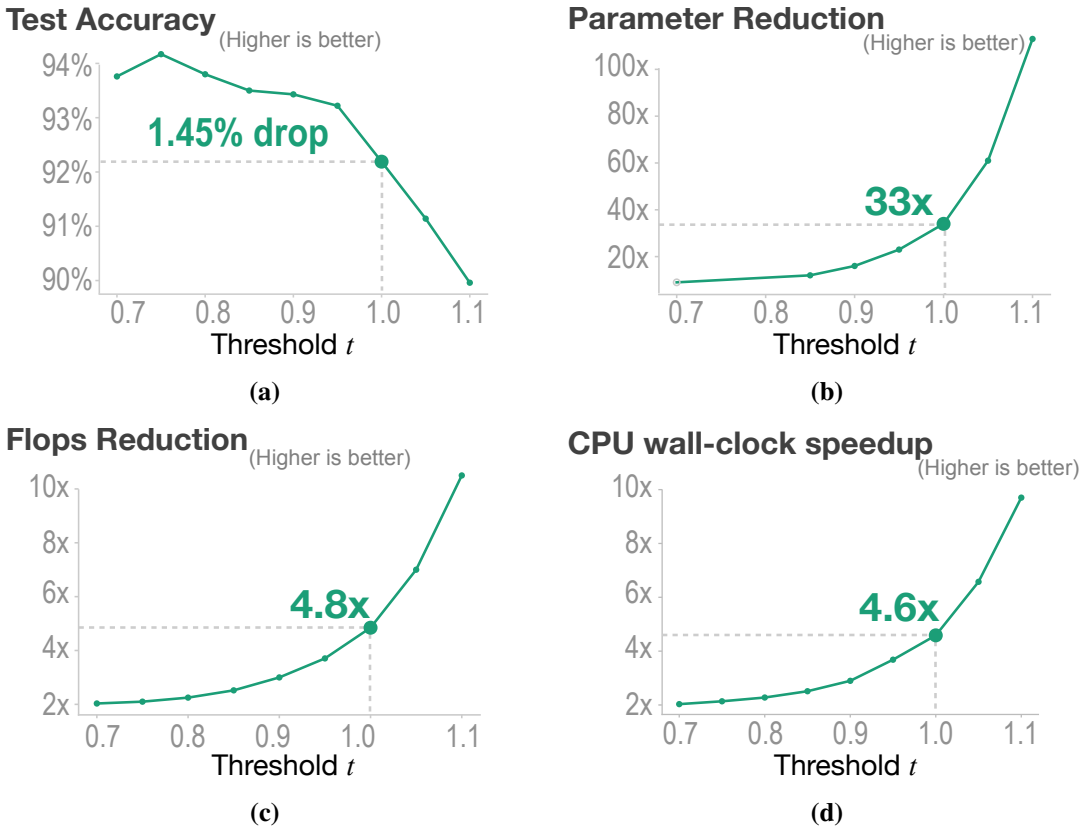


Figure 3.5: Compressing a VGG-16 on CIFAR-10 using CUP. We show the effect of pruning with a larger t on (a) test accuracy, (b) parameters, (c) flops, and (d) inference wall-clock speedup.

save over 60 GPU hours compared to their traditional 3-step *retrain-allowed* counterparts. However, *retrain-allowed* models are generally more compact and also achieve a better Top-1 accuracy.

Table 3.1: For a ResNet-50 trained on ImageNet, our approach (bolded) achieves the shortest training time, best top-1 accuracy, and most flop reduction, in both the *retrain-allowed* (✓) and the *retrain-free* settings (✗). FR means *flops reduction*.

Method	Retrain?	Top-1 (%)	FR (×)	Training Time (GPU Hours)
Resnet-50	-	75.86	1.00	66.0
SFP [68]	✓	62.14	2.39	122.4
GM [69]	✓	74.83	3.81	122.7
CUP (ours)	✓	**75.07**	**3.86**	**116.8**
SFP [68]	✗	74.01	1.73	61.8
GM [69]	✗	74.13	2.15	62.2
CUP-RF (ours)	✗	**74.34**	**2.21**	**51.6**

Method	Model	Accuracy change (%)	
		Without retraining	With retraining
Base	B	0	0
random	M2	-57.04%	-0.18%
L2	M2	-19.04%	-0.16%
L1 [17]	M2	-18.48%	-0.25%
SSL [22]	M1	-	-0.10%
Slimming [19]	M2	-	-0.06%
CUP	M2	**-13.26%**	**0.00%**

Table 3.2: Accuracy change while compressing base model B ($784 - 500 - 300 - 10$) to the compressed model M1 ($434 - 174 - 78 - 10$) and M2 ($784 - 100 - 60 - 10$). CUP achieves the best accuracy with and without retraining.

Table 3.3: For ResNet-56 and VGG-16 models trained on CIFAR-10, our approach (bolded) leads to highest flops reduction (FR), in both the *retrain-allowed* (✓) and the *retrain-free* settings (✗). FR means *flops reduction*.

Method	Retrain?	ResNet-56		VGG-16	
		FR ($\times$)	Acc ($\Delta\%$)	FR ($\times$)	Acc ($\Delta\%$)
L1 [17]	✓	1.37	-0.02	1.51	-0.15
CP [20]	✓	2.00	-1.00	2.00	-0.32
GM [69]	✓	2.10	-0.33	-	-
GAL [70]	✓	2.45	-1.68	1.82	-0.54
NS [19]	✓	-	-	2.04	-0.32
CUP (ours)	✓	**2.77**	-0.40	**3.70**	-0.70
SFP [68]	✗	2.10	-1.33	-	-
GM [69]	✗	2.10	-0.70	-	-
VCNP [71]	✗	1.25	-0.78	1.64	-0.07
GAL [70]	✗	1.59	-0.28	1.82	-3.18
CUP-RF (ours)	✗	**2.12**	-0.31	**3.15**	-0.40

3.5.4 Comparison via accuracy change on MNIST

Our goal in this subsection is to observe the change in accuracy while compressing baseline model B to compressed model M2 on the MNIST dataset. Here, model B is a four layer fully connected network consisting of 784-500-300-10 filters [22, 19]. In [22] model B is compressed to model M1: 434-174-78-10 which corresponds to 83.5% parameter reduction. In [19] it is compressed to model M2: $784 - 100 - 60 - 10$ which corresponds to 84.4% sparsity. In table 3.2 we compress baseline B to model M2 using CUP.

Notice that in the *without retraining* setting, CUP leads to minimum drop in accuracy. This signifies that our similarity based pruning criterion can better identify redundant filters compared to any of the magnitude based pruning criterion. Further, when we *allow*

Table 3.4: For ResNet-18/34/50 models trained on ImageNet, our approach (bolded) leads to highest flops reduction (FR) with minimal accuracy drop, in both the *retrain-allowed* (✓) and the *retrain-free* settings (✗). FR means *flops reduction*.

Model	Method	Retrain?	FR (×)	Acc. (Δ%)	
				Top-1	Top-5
ResNet-18	GM [69]	✓	1.71	-1.87	-1.15
	COP [81]	✓	**1.75**	-2.48	-
	CUP (Our)	✓	**1.75**	**-1.00**	**-0.79**
	SFP [68]	✗	1.71	-3.18	-1.85
	GM [69]	✗	1.71	-2.47	-1.52
	CUP-RF (ours)	✗	**1.75**	**-2.37**	**-1.40**
ResNet-34	L1 [17]	✓	1.31	-1.06	-
	GM [69]	✓	1.69	-1.29	-0.54
	CUP (ours)	✓	**1.78**	**-0.86**	**-0.53**
	SFP [68]	✗	1.69	-2.09	-1.29
	GM [69]	✗	1.69	-2.13	-0.92
	CUP-RF (ours)	✗	**1.71**	**-1.61**	**-0.89**
ResNet-50	SFP [68]	✓	2.15	-14.0	-8.20
	MP [82]	✓	2.05	-1.20	-
	CUP (ours)	✓	**2.47**	**-1.17**	**-0.81**
	SFP [68]	✗	1.71	-1.54	**-0.81**
	GM [69]	✗	2.15	-2.02	-0.93
	CUP-RF (ours)	✗	**2.20**	**-1.47**	-0.88

retraining, CUP can fully recover the original model's accuracy.

3.5.5 Comparison via flops and parameter reduction on CIFAR-10 and ImageNet

Tables 3.3 and 3.4 present the results for compressing ResNet-56, VGG-16 models on CIFAR-10 and ResNet-{18,34,50} models on ImageNet, under the *retrain-allowed* and *retrain-free* settings.

Retraining-allowed (rows with ✓). Under this traditional three-stage pipeline, our approach consistently leads to highest flops reduction with a lower drop in accuracy.

Retrain-free (rows with ✗). Even when retraining is not allowed, our approach leads to highest flops reduction with a comparable, or in many cases, lower drop in accuracy.

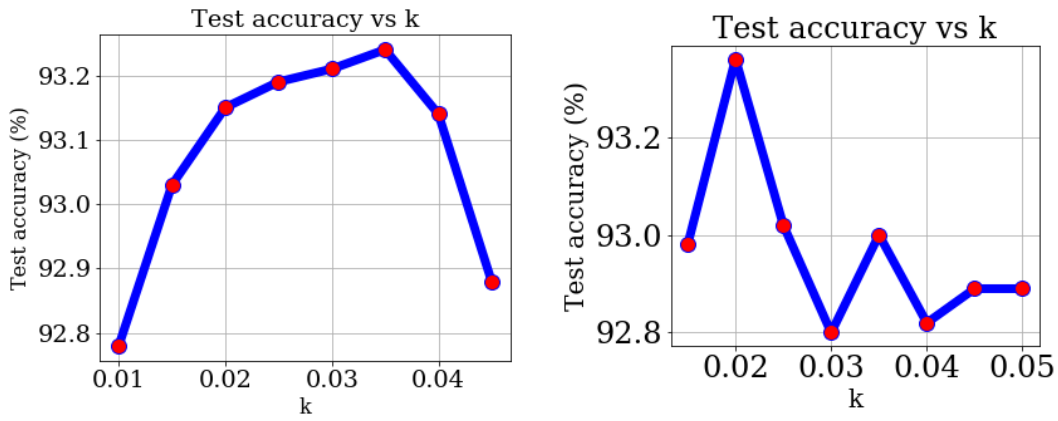
3.6 Ablation studies and hyper-parameter search

In this section, we perform ablation studies on the hyper-parameters introduced in CUP. Recall that CUP uses t to control the compression ratio while CUP-RF uses k and b. In

the following subsections, we present results on both CIFAR-10 and Imagenet.

3.6.1 Searching for t on CIFAR-10

The only hyper-parameter for compression using the CUP framework is the global threshold t. As a model designer, our goal is to identify the optimal t that yields a model that satisfies the flops budget of the deployment device. To study the effect of varying t in CUP, in figure 3.5, we apply CUP to 9 VGG-16 models on CIFAR-10 corresponding to $t \sim Uniform(0.7, 1.1)$. The final accuracy, parameters reduction, flops reduction and inference time speedup are plotted in fig 3.5a-d. Like previous studies, we see from fig 3.5a that mild compression first leads to an increase (94.17% from 93.61%, for $t = 0.75$) in validation accuracy. However, further compression ($t = [0.8 - 1.1]$ leads to a drop in accuracy.



(a) **(b)**

Figure 3.6: Validation accuracy on CIFAR-10 versus k for compressing (a) VGG-16 and (b) Resnet-56 using CUP-RF. As intuition suggests, A very fast pruning schedule (high k) damages the neural network whereas a very slow pruning rate leads to incomplete pruning.

3.6.2 Searching for k,b on CIFAR-10

The only hyper-parameters for compression using the CUP-RF framework are k, b. To study the effect of varying k, b in CUP-RF, in fig 3.6, we plot the final accuracy for VGG-16 and Resnet-56 trained on CIFAR-10 versus k. Recall that k controls the slope of the single-shot pruning schedule introduced in (3.6) while b is the offset. We constrain $b = 0$. From the figure, we see that for both VGG-16 (fig 3.6a) and Resnet-56 (fig 3.6b) the ideal value of k lies somewhere in the middle of the interval $[0.01, 0.05]$.

Key Takeaway: A very fast pruning schedule (high k) can damage the neural network whereas a very slow pruning rate may lead to incomplete pruning.

3.6.3 Searching for t on Imagenet

In table 3.5 we demonstrate the final accuracy achieved by various Resnet variants (trained on Imagenet) for increasing values of t. notice how the flops reduction metric (FR) gracefully increases with increasing t. This is similar to the observation for compressing VGG-16 on CIFAR-10 as noted in section 3.6 and fig 3.5c.

Table 3.5: Line search on t for compressing Resnet variants on Imagenet using CUP. Notice that increasing t leads to a graceful increase in flops reduction (FR).

t	FR (×)	Accuracy Change (%) Top-1 / Top-5
Resnet-18 on Imagenet		
Base	1×	69.88 / 89.26
0.8	1.75×	68.88 / 88.47
0.825	1.97 ×	68.29 / 88.15
Resnet-34 on Imagenet		
Base	1×	73.59 / 91.44
0.60	1.78×	72.73 / 90.91
0.65	2.08×	71.99 / 90.47
0.675	2.29×	71.65 / 90.21
0.70	2.55×	71.15 / 90.08
Resnet 50 on Imagenet		
Base	1×	75.86 / 92.87
0.65	2.18×	75.07 / 92.30
0.675	2.32×	74.73 / 92.14
0.70	2.47×	74.60 / 92.06
0.725	2.64×	74.42 / 91.74

3.6.4 Searching for k,b On Imagenet

From our experience on CIFAR-10, we realize that k needs to be neither too small nor too large. Consequently, we found $k = 0.03$ worked well and fixed it for all experiments. For our linesearch presented in table 3.6, we vary b over steps of 0.1. Notice that the reduction in training time for Resnet-50 is much larger than that for other models. This is due to the fact that the original model itself is much larger than its 18 and 34 layer variants. Thus

compressing it $2\times$ results in a much larger reduction in flops and training time. As a model designer, our goal is to identify a good k, b that leads to large training time reduction. Having set these values, the designer needs to specify the tgt_flops that satisfies the flops budget of the deployment device.

Table 3.6: Line search for k, b in compressing Resnet variants using CUP-RF. We desire a higher FR for an acceptable drop in accuracy. The key observation here is the increasingly larger savings in training time for Resnet 18/34/50.

k/b	FR ($\times$)	Accuracy Change (%) Top-1 / Top-5	Training Time (hrs)
Resnet-18 on Imagenet			
Base	$1\times$	69.88 / 89.26	38.75
0.03/0.3	$1.74\times$	67.38 / 87.86	38.6
0.03/0.4	$1.90\times$	66.86 / 87.37	38.8
0.03/0.5	$1.83\times$	67.24 / 87.59	38.4
Resnet-34 on Imagenet			
Base	$1\times$	73.59 / 91.44	44.91
0.03/0.3	$1.71\times$	71.98 / 90.42	39.7
0.03/0.4	$2.08\times$	71.69 / 90.28	39.4
Resnet 50 on Imagenet			
Base	$1\times$	75.86 / 92.87	66.00
0.03/0.3	$2.20\times$	74.40 / 91.99	54.48
0.03/0.4	$2.16\times$	74.31 / 92.10	52.45
0.03/0.5	$2.21\times$	74.39 / 91.94	51.58

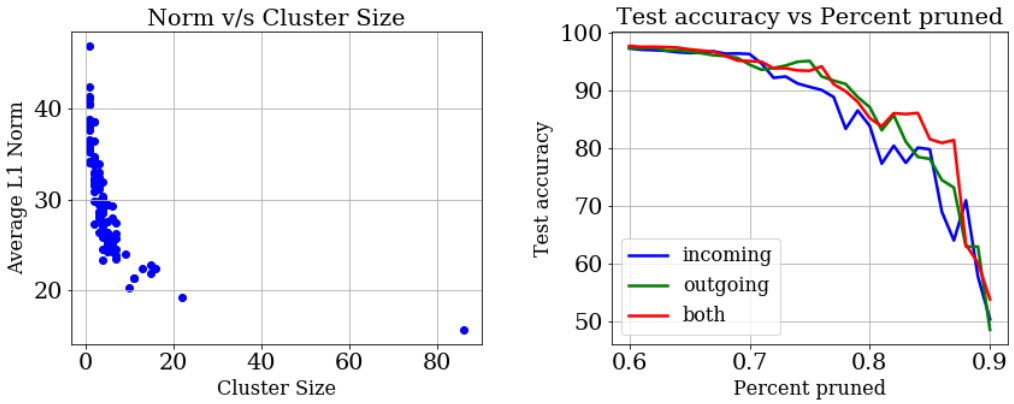
3.7 Discussion

We have seen thus far that CUP outperforms prior work quantitatively on several datasets and models. In this section, we draw connections with prior work and provide insights into how CUP generalizes some of these methods.

3.7.1 Filter Saliency & connection to magnitude pruning

An effective strategy for pruning entire filters is to prune based on the magnitude of filter weights. One of the earliest works along this direction, [8] proposed to prune individual weight connections based on magnitude thresholding. Later, [17] generalized this idea to prune entire filters having a low $L1$ norm of incoming weights. We observe that CUP generalizes magnitude based pruning further by using feature similarity as the pruning metric. This metric captures the notion of pruning based on weight magnitude as noted

in figure 3.7a where we plot the average $L1$ norm of features for filters in a cluster versus the cluster size. Magnitude pruning operates only at the right end of that figure wherein majority filters have small weights. However, CUP operates along the entire axes which means it additionally prunes similar filters that have high weight magnitudes. Hence CUP is able to prune a higher number of filters and thus encumbers a lower drop in accuracy for a sufficiently pruned network. This is observed in table 3.2 where $L1$ and $L2$ based magnitude pruning lead to 13.48% and 19.04% accuracy drop (without retraining) versus 8.26% for CUP.



(a) **(b)**

Figure 3.7: (a) shows the average $L1$ norm of cluster with size (b) shows the accuracy of the compressed model versus compression.

Figure 3.7b presents the accuracy for CUP using features computed from "incoming", "outgoing" and "both" type of weight connections. Each point along the x-axis notes the percentage of filters dropped from all layers of model B. This is varied from $[0.6, 0.9]$ in steps of 1. The y-axis notes the validation accuracy of the resulting model. It is observed that the combination of input/output features works best.

3.7.2 Comparison with training from scratch

Recently, [66] performed an empirical study showing that compression could be achieved by training a pre-determined (small) model from scratch. They argued against methods which prune a fixed number of filters from each layer. In the same spirit, in this subsection we replicate their main experiment—We compare the performance of the compressed VGG-16 model obtained through CUP ($t = 0.9$) versus training the same model from scratch in table 3.7. The scratch-160 model is obtained by training the compressed model from scratch for 160 epochs while the scratch-480 model is trained from scratch for $3\times$

epochs to compensate for the $3\times$ flop reduction in the compressed model. It is seen that the CUP compressed model outperforms both the scratch trained models. The margin may not seem large, however it must be noted the compressed model was itself discovered through CUP. Thus, the value of pruning lies also lies in discovering the compressed model [66].

Table 3.7: Benchmarking the VGG-16 compressed model discovered by CUP (T=0.9) when trained from scratch.

Model	Accuracy change
CUP (t=0.9)	**-0.09**%
scratch-160 epochs	-0.75%
scratch-480 epochs	-0.41%

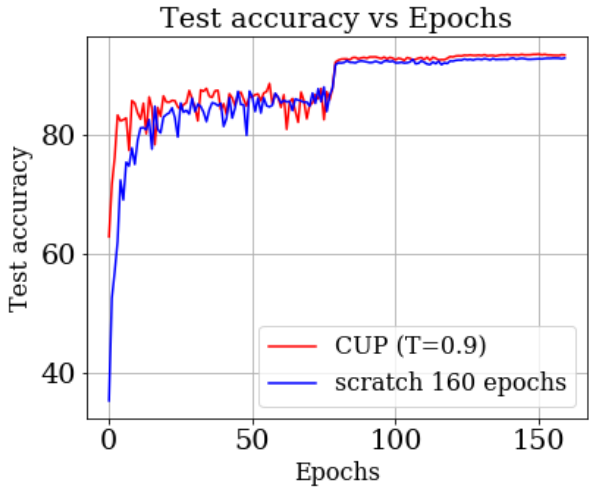


Figure 3.8: Comparison of test accuracy for the model obtained via CUP (T=0.9) versus the same model trained from scratch.

3.8 Conclusion

We proposed a new channel pruning based method for model compression that prunes entire filters based on similarity. We showed how hierarchical clustering can be used to enable layer-wise *non-uniform* pruning whilst introducing only a single hyper-parameter. Using multiple models and datasets, we demonstrated that CUP achieves the highest flops reduction with the least drop in accuracy. Further, CUP-RF leads to large savings in training time with only a small drop in performance. A limitation of the current work is that we used simple yet effective *linear trajectory* for setting t in CUP-RF.

CHAPTER 4
COMPATIBILITY-AWARE HETEROGENEOUS VISUAL SEARCH

We tackle the problem of visual search under resource constraints. Existing systems use the same embedding model to compute representations (embeddings) for the query and gallery images. Such systems inherently face a hard accuracy-efficiency trade-off: the embedding model needs to be large enough to ensure high accuracy, yet small enough to enable query-embedding computation on resource-constrained platforms. This trade-off could be mitigated if gallery embeddings are generated from a large model and query embeddings are extracted using a compact model. The key to building such a system is to ensure representation compatibility between the query and gallery models. In this paper, we address two forms of compatibility: One enforced by modifying the parameters of each model that computes the embeddings. The other by modifying the architectures that compute the embeddings, leading to compatibility-aware neural architecture search (CMP-NAS). We test CMP-NAS on challenging retrieval tasks for fashion images (DeepFashion2), and face images (IJB-C). Compared to ordinary (homogeneous) visual search using the largest embedding model (paragon), CMP-NAS achieves 80-fold and 23-fold cost reduction while maintaining accuracy within 0.3% and 1.6% of the paragon on DeepFashion2 and IJB-C respectively.

4.1 Introduction

A visual search system in an "open universe" setting is often composed of a gallery model ϕ_g and a query model ϕ_q, both mapping an input image to a vector representation known as *embedding*. The gallery model ϕ_g is typically used to map a set of gallery images onto their embedding vectors, a process known as indexing, while the query model extracts embeddings from query images to perform search against the indexed gallery. Most existing visual search approaches [83, 84, 85, 86, 87] use the same model architecture for both ϕ_q and ϕ_g. We refer to this setup as *homogeneous visual search*. An approach that uses different model architectures for ϕ_q and ϕ_g is referred to as *heterogeneous visual search* (HVS).

The use of the same $\phi_g = \phi_q$ trivially ensures that gallery and query images are mapped to the same vector space where the search is conducted. However, this engenders a hard accuracy-efficiency trade-off (Fig. 4.1)— choosing a large architecture ϕ_g for both query and gallery achieves high-accuracy at a loss of efficiency; choosing a small architecture ϕ_q

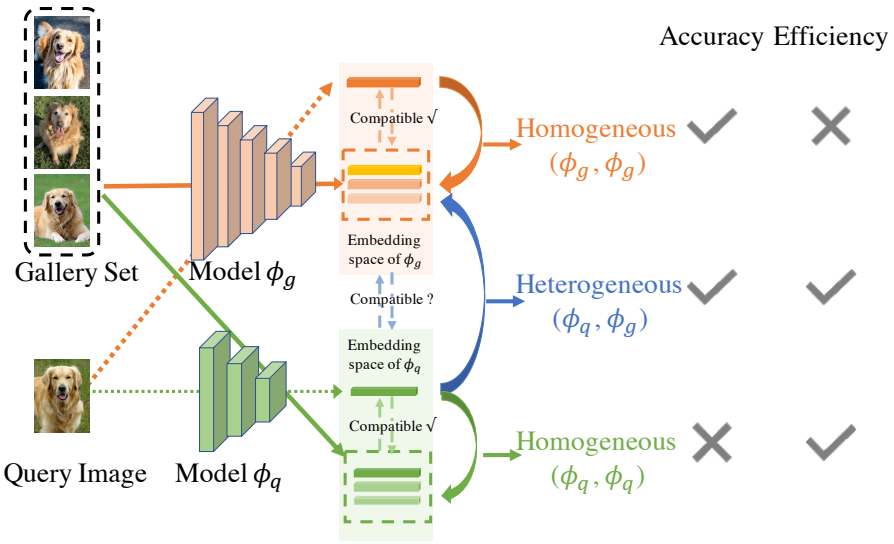


Figure 4.1: Homogeneous visual search uses the same embedding model, either large (orange) to meet performance specifications, or small (green) to meet cost constraints, forcing a dichotomy. Heterogeneous Visual Search (blue) uses a large model to compute embeddings for the gallery, and a small model for the query images. This allows high efficiency without sacrificing accuracy, provided that the green and orange embedding models are designed and trained to be *compatible*.

improves efficiency to the detriment of accuracy, which is compounded since in practice, indexing only happens sporadically while querying is performed continuously. This leads to efficiency being driven mainly by the query model. HVS allows the use of a small model ϕ_q for querying, and a large model ϕ_g for indexing, partly mitigating the accuracy-complexity trade-off by enlarging the trade space. The challenge in HVS is to ensure that ϕ_g and ϕ_q live in the same metric (vector) space. This can be done for given architectures ϕ_g, ϕ_q, by training the weights so the resulting embeddings are metrically compatible [88]. However, one can also enlarge the trade space by including the architecture in the design of metrically compatible models. Typically, ϕ_g is chosen to match the best current state-of-the-art (paragon) while the designer can search among query architectures ϕ_q to maximize efficiency while ensuring that performance remains close to the paragon.

In this work, we pursue compatibility by optimizing both the model parameters (weights) as well as the model architecture. We show that (1) weight inheritance [89] and (2) backward-compatible training (BCT) [88] can achieve compatibility through weight optimization. Among these, the latter is more general in that it works with *arbitrary* embedding functions ϕ_g and ϕ_q. We expand beyond BCT to neural architecture search (NAS) [9, 90, 91, 92] with our proposed compatibility-aware NAS (CMP-NAS) strategy that searches for a query model ϕ_q that is maximally efficient while being compatible with ϕ_g. We hypothesize that CMP-NAS can simultaneously find the architecture of query model and its weights that achieve efficiency similar to that of the smallest (query) model, and ac-

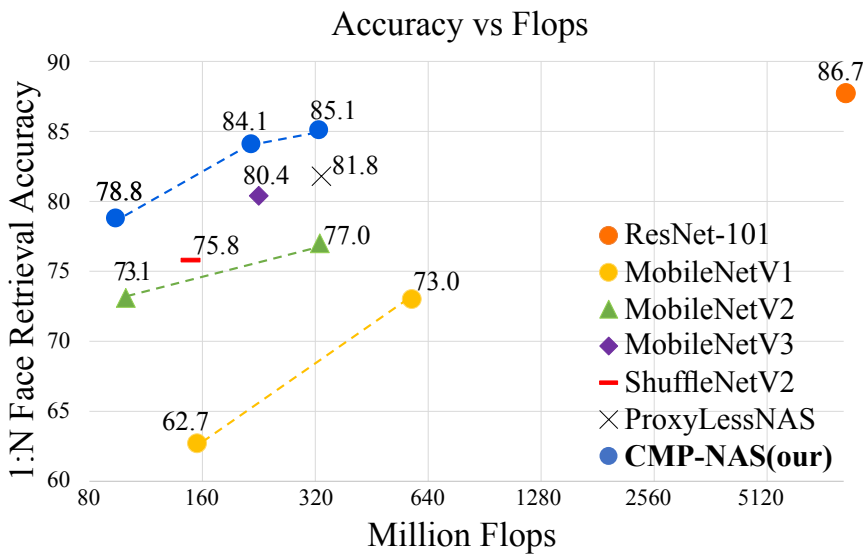


Figure 4.2: The trade-off between accuracy and efficiency for a heterogeneous system performing 1:N Face retrieval on DeepFashion2. We use a ResNet-101 as the gallery model and compare different architectures as query models. For MobileNetV1 and V2, we provide results with width $0.5\times$ and $1\times$.

curacy close to that of the paragon (gallery model). Indeed the results in Fig. 4.2 shows that CMP-NAS outperforms all of the state-of-the-art off-the-shelf architectures designed for mobile platforms with resource constraints. Compared with paragon (state-of-the-art high-compute homogeneous visual search) methods, HVS reduce query model flops by $23\times$ with only 1.6% in loss of search accuracy for the task of face retrieval.

Our *contributions* can be summarized as follows: 1) we demonstrate that an HVS system allows to better trade off accuracy and complexity, by optimizing over both model parameters and architecture. 2) We propose a novel CMP-NAS method combining weight-based compatibility with a novel reward function to achieve compatibility-aware architecture search for HVS. 3) We show that our CMP-NAS can reduce model complexity many-fold with only a marginal drop in accuracy. For instance, we achieve $23\times$ reduction in flops with only 1.6% drop in retrieval accuracy on face retrieval using standard benchmarks.

4.2 Related Work

Visual search: Most prior visual search systems construct embedding vectors either by aggregating hand-crafted features [93, 94, 95, 96, 97, 98], or through feature maps extracted from a convolutional neural network [84, 99, 85, 100, 84, 86, 101, 87, 102]. The latter, being more prevalent in recent times, differ from us in that they follow the homogeneous visual search setting and suffer from a hard accuracy and efficiency trade-off. Recently, [103] discusses the asymmetric testing task which is similar to our heterogeneous setting.

However, their method is unable to ensure that the heterogeneous accuracy supersedes the homogeneous one (compatibility rule in Sec. 4.3.1). Such a system is not practically useful since the homogeneous deployment achieves both a higher accuracy and a higher efficiency.

Cross-model compatibility: The broad goal of this area is to ensure embeddings generated by different models are compatible. Some recent works ensure cross-model compatibility by learning transformation functions from the query embedding space to the gallery one [104, 105, 106]. Different from these works, our approach directly optimizes the query model such that its metric space aligns with that of the gallery. This leads to more flexibility in designing the query model and allows us to introduce architecture search in the metric space alignment process. Our idea of model compatibility, as metric space alignment, is similar to the one in backward-compatible training (BCT) [88]. However, [88] only considers compatibility through model weights, whereas, we generalize this concept to the model architecture. Additionally, [88] targets for compatibility between an updated model and its previous (less powerful) version, the application scenarios of which are different from this work.

Architecture Optimization: Recent progress demonstrates the advantages of automated architecture design over manual design through techniques such as neural architecture search (NAS) [9, 92, 91, 90]. Most existing NAS algorithms however, search for architectures that achieve the best accuracy when used independently. In contrast, our task necessitates a deployment scenario with two models: one for processing the query images and another for processing the gallery. Recently, [107, 108] propose to use a large teacher model to guide the architecture search process for a smaller student which is essentially knowledge distillation in architecture space. However, our experiments show that knowledge distillation cannot guarantee compatibility and thus these methods may not succeed in optimizing the architecture in that aspect. To the best of our knowledge, CMP-NAS is the first to consider the notion of compatibility during architecture optimization.

4.3 Methodology

We use ϕ to denote an embedding model in a visual search system and κ to denote the classifier that is used to train ϕ. We further assume ϕ is determined by its architecture a and weights w. For our visual search system, a gallery model is first trained on a training set $\mathcal{T}$ and then used to map each image x in the gallery set $\mathcal{G}$ onto an embedding vector $\phi_g(x) \in \mathbb{R}^K$. Note this mapping process only uses the embedding portion ϕ_g. During test time, we use the query model (trained previously on $\mathcal{T}$) to map the query image x' onto an embedding vector $\phi_q(x') \in \mathbb{R}^K$. The closest match is then obtained through a nearest

neighbor search in the embedding space. Typically, visual search accuracy is measured through some metric, such as top-10 accuracy, which we denote by $M(\phi_q, \phi_g; \mathcal{Q}, \mathcal{G})$. This is calculated by processing query image set $\mathcal{Q}$ with ϕ_q and processing the gallery set $\mathcal{G}$ with ϕ_g. For simplification, we omit the image sets and adopt the notation $M(\phi_q, \phi_g)$ to denote our accuracy metric.

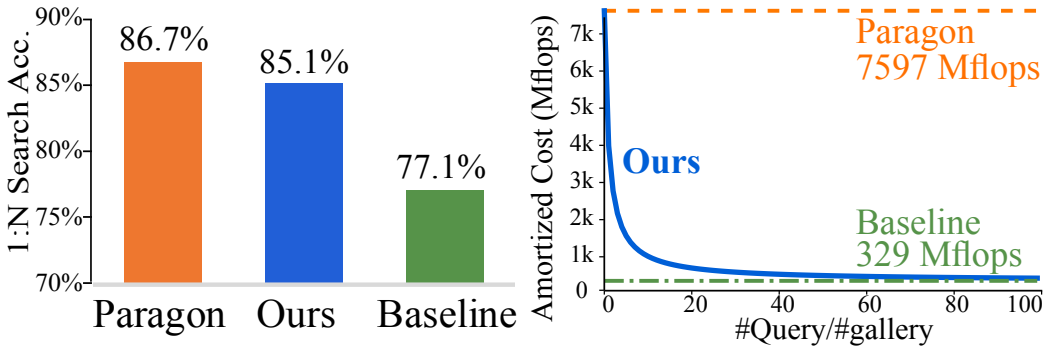
4.3.1 Homogeneous vs heterogeneous visual search

Assuming ϕ_q and ϕ_g are different models and ϕ_q is smaller than ϕ_g, we define two kinds of visual search:

- **Homogeneous visual search** uses the same embedding model to process the gallery and query images, and is denoted by (ϕ_q, ϕ_q) or (ϕ_g, ϕ_g).
- **Heterogeneous visual search** uses different embedding models to process the query and gallery images, respectively, and is denoted by (ϕ_q, ϕ_g).

We illustrate the accuracy-efficiency trade-off faced by visual search systems in Fig. 4.3. A homogeneous system with a larger embedding model (*e.g.*, ResNet-101 [80], denoted as paragon) achieves a higher accuracy due to better embeddings (orange bar in Fig. 4.3(a)) but also consumes more flops during query time (orange line in Fig. 4.3(b)). On the other hand, a smaller embedding model (*e.g.*, MobileNetV2 [109], denoted as baseline) in the homogeneous setting achieves the opposite end of the trade-off (green bar and line in Fig. 4.3(a),(b)). Our heterogeneous system (blue bar and line in Fig. 4.3(a),(b)) achieves accuracy within 1.6% of the paragon and efficiency of the baseline.

When computing the cost of a visual search method, one has to take into account both the cost of indexing, which happens sporadically, and the cost of querying, which occurs continuously. While large, the indexing cost is amortized through the lifetime of the system. To capture both, in Fig. 4.3(b) we report the amortized cost of embedding the query and gallery images, as a function of the ratio of queries to gallery images processed. In most practical systems, the number of queries exceeds the number of indexed images by orders of magnitude, so the relevant cost is the asymptote, but we report the entire curve for completeness. The initial condition for that curve is the cost of the paragon. Our goal is to design a system that has a cost approaching the asymptote (b), with performance approaching the paragon (a).



(a) Accuracy on IJB-C. **(b)** Amortized cost analysis

Figure 4.3: Accuracy-efficiency trade-off for visual search. In (a) we compare the 1:N face retrieval accuracy (TPIR@FPIR=10^{-1}) on IJB-C. We denote the homogeneous system with ResNet-101 and MobileNetV2 as the paragon and baseline respectively. In (b) we observe that, as the size of the query set increases, the complexity of our heterogeneous system converges to that of the baseline.

Notion of Compatibility

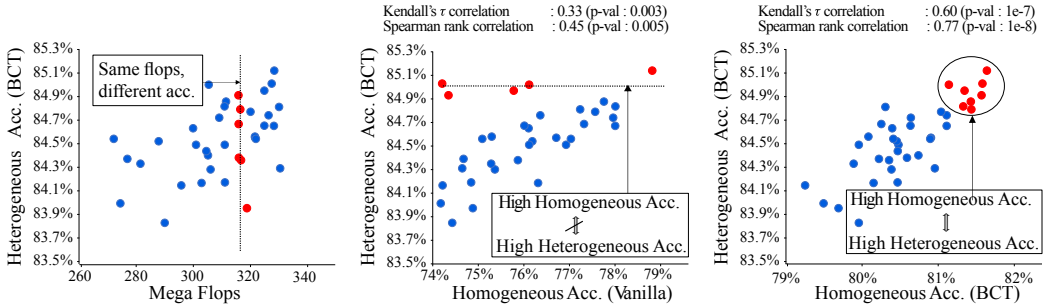
A key requirement of a heterogeneous system is that the query and gallery models should be compatible. We define this notion through the **compatibility rule** which states that:

> A *smaller* model ϕ_q is compatible with a *larger* model ϕ_g if it satisfies the inequality $M(\phi_q, \phi_g) > M(\phi_q, \phi_q)$.

We note that satisfying this rule is a necessary condition for heterogeneous search. A heterogeneous system violating this condition, *i.e.*, $M(\phi_q, \phi_g) < M(\phi_q, \phi_q)$, is not practically useful since the homogeneous system $M(\phi_q, \phi_q)$ achieves both higher efficiency and accuracy. Additionally, a practically useful heterogeneous system should also satisfy $M(\phi_q, \phi_g) \approx M(\phi_g, \phi_g)$. In subsequent sections, we study how to achieve both these goals through weight and architecture compatibility.

4.3.2 Compatibility for Heterogeneous Models

In this section, we discuss different ways to obtain compatible query and gallery models ϕ_q, ϕ_g that satisfy the compatibility rule. While a general treatment may optimize ϕ_q and ϕ_g jointly, in this paper, we consider the simpler case when ϕ_g is fixed to a standard large model (ResNet-101) while we optimize the query model ϕ_q. For the subsequent discussion, we assume the gallery model ϕ_g has an architecture a_g and is parameterized by weights w_g. Corresponding quantities for the query model are ϕ_q with architecture a_q and parameterized by weights w_q. To train the query and gallery models ϕ_q, ϕ_g we use the classification-based training [88, 110] with the query and gallery classifiers denoted by κ_q and κ_g respectively. In what follows, we discuss two levels of compatibility—weight level and architecture level.



(a) **(b)** **(c)**

Figure 4.4: NAS Motivation. We randomly sample 40 architectures from the ShuffleNet search space of [91] and train them from scratch. Observe that (a) Architectures with same flops (shown with red circles) can have different heterogeneous accuracies proving that architecture has a measurable impact on compatibility. (b) Architectures (shown in red) achieving the highest heterogeneous accuracy with BCT training are not the ones achieving the highest homogeneous accuracy with vanilla training. This means that traditional NAS (which optimizes for homogeneous accuracy while using vanilla training) may fail to find the most compatible models. (c) When trained with BCT, the architectures achieving the highest heterogeneous accuracy also achieve the highest homogeneous accuracy. This means simply equipping traditional NAS with BCT will aid the search for compatible architectures.

Weight-level compatibility

Given the gallery model ϕ_g and its classifier κ_g, weight-level compatibility aims to learn the weights w_q of query model ϕ_q such that the compatibility rule is satisfied. To this end, the optimal query weights w_q^*, and its corresponding classifier κ_q^* can be learned by minimizing a composite loss over the training set $\mathcal{T}$.

$$w_q^*, \kappa_q^* = \underset{w_q, \kappa_q}{\operatorname{argmin}} \{\lambda_1 \mathcal{L}_1(w_q, \kappa_q; \mathcal{T}) + \lambda_2 \mathcal{L}_2(w_q, \kappa_q, w_g, \kappa_g; \mathcal{T})\}, \quad (4.1)$$

where $\mathcal{L}_1$ is a classification loss such as the Cosine margin [111], Norm-Softmax [112] and $\mathcal{L}_2$ is the additional term which promotes compatibility. We consider four training methods which can be described using Eq.4.1 as follows:

1. Vanilla training: Considers $\lambda_2 = 0$.
2. Knowledge Distillation [12]: $\mathcal{L}_2$ is the temperature smoothed cross-entropy loss between the logits of query and gallery model.
3. Fine-tuning: Initializes w_q using w_g and κ_q using κ_g and considers $\lambda_2 = 0$.
4. Backward-compatible training (BCT) [88]: Uses $\mathcal{L}_2 = \mathcal{L}_1(w_q, \kappa_g; \mathcal{T})$. This ensures that the query embedding model learns a representation that is compatible with the gallery classifier.

We compare these methods in Tab. 4.4, and find that only the last two succeed in ensuring compatibility. Among these two, fine-tuning is more restrictive since it makes a stronger assumption about the query architecture—it requires the query model to have a similar network structure, kernel size, layer configuration as the gallery model. In contrast, BCT poses no such restriction and can be used to train any query architecture. Thus we use [88] as our default method to ensure weight-level compatibility. Recently [103] proposed to learn the weights of a query model by minimizing the L_2 distance between query and gallery embeddings, however, both [103] and [88] observe that the resulting query model does not satisfy the compatibility rule.

Architecture-level compatibility

Given the gallery model ϕ_g and classifier κ_g, the problem of architecture-level compatibility aims to search for an architecture a_q for the query model ϕ_q that is most compatible with a fixed gallery model. The need of architecture level compatibility is motivated by two questions:

Q1 How much does architecture impact compatibility?

Q2 Can traditional NAS find compatible architectures?

To answer these questions, we randomly sample 40 architectures from the ShuffleNet search space [91], with each having roughly 300 Million flops.

A1 We train these architectures with BCT ($\lambda_1 = 1, \lambda_2 = 1$ in Eq. 4.1) and plot the heterogeneous accuracy *vs*. flops in Fig. 4.4(a). There are two observations: (1) heterogeneous accuracy is not correlated with flops and (2) architectures with similar flops can achieve different accuracy, which indicates architecture indeed has a measurable impact on accuracy.

A2 We plot the homogeneous accuracy of models with vanilla training (target of traditional NAS) *vs*. heterogeneous accuracy of the same models trained with BCT (our target) in Fig. 4.4(b). We observe that: (1) The correlation between the two accuracy is low and (2) The architectures with the highest heterogeneous accuracy are not those with highest homogeneous accuracy. This indicates traditional NAS may not be successful in searching for compatible architectures.

We further investigate the correlation between homogeneous (with BCT) and heterogeneous accuracy (with BCT) in Fig. 4.4(c) and discover that the correlation of these two accuracies is much higher than that in Fig. 4.4(b). This offers a key insight that equipping traditional NAS with BCT may help in searching compatible architectures.

Architecture optimization with CMP-NAS Based on the intuition developed previously, we develop CMP-NAS using the following notation. Denote by $\phi_q(a_q, w_q)$ a candidate query embedding model with architecture a_q and weights w_q. We further denote κ_q as its corresponding classifier. With CMP-NAS, we solve a two-step optimization problem where the first step amounts to learning the best set of weights— w_q^* (for the embedding model ϕ_q) and κ_q^* (for the common classifier)–by minimizing a classification loss $\mathcal{L}$ over the training set $\mathcal{T}$ as below

$$w_q^*, \kappa_q^* \quad = \quad \underset{w_q, \kappa_q}{\operatorname{argmin}} \{\lambda_1 \mathcal{L}(\phi_q(a_q, w_q), \kappa_q; \mathcal{T}) \quad + \quad \lambda_2 \mathcal{L}(\phi_q(a_q, w_q), \kappa_g; \mathcal{T})\}, \quad (4.2)$$

where $\mathcal{L}$ can be any classification loss such as Cosine margin [111], Norm-Softmax [112]. Similar to BCT, the second term $\mathcal{L}(\phi_q(a_q, w_q), \kappa_g; \mathcal{T})$ ensures that the candidate query embedding model $\phi_q(a_q, w_q^*)$ learns a representation that is compatible with the gallery classifier.

Using weights w_q^* and κ_q^* from above, the second step amounts to finding the best query architecture a_q^* in a search space Ω, by maximizing a reward $\mathcal{R}$ evaluated on the validation set as below

$$a_q^* = \underset{a_q \in \Omega}{\operatorname{argmax}} \, \mathcal{R}\left(\phi_q(a_q, w_q^*), \kappa_q^*\right). \quad (4.3)$$

We consider three candidate rewards presented in Tab 4.1. Similar to traditional NAS, homogeneous accuracy $M(\phi_q(a_q, w_q), \phi_q(a_q, w_q))$ is our baseline reward $\mathcal{R}_1$. Recall that however, we are interested in searching for the architecture which achieves the best heterogeneous accuracy. With this aim, we design rewards $\mathcal{R}_2$ and $\mathcal{R}_3$ which include the heterogeneous accuracy in their formulation.

Table 4.1: Different rewards considered with CMP-NAS. The rewards $\mathcal{R}_1$, $\mathcal{R}_2$ prioritize either the symmetric or asymmetric accuracy while ignoring the other. $\mathcal{R}_3$ prioritizes *both* accuracies and consistently outperforms other rewards.

Reward	Formulation
$\mathcal{R}_1$	$M(\phi_q(a_q, w_q), \phi_q(a_q, w_q)$
$\mathcal{R}_2$	$M(\phi_q(a_q, w_q), \phi_g)$
$\mathcal{R}_3$	$M(\phi_q(a_q, w_q), \phi_q(a_q, w_q)) \times M(\phi_q(a_q, w_q), \phi_g)$

Our CMP-NAS formulation in Eq. 4.2 and rewards in Tab. 4.1 is general and can work with any NAS method. For demonstration, we test our idea with the single path one shot NAS [91] and consists of the following two components:

Search Space: Similar to popular weight sharing methods [90, 91, 24], the search space

of our query model consists of a shufflenet-based super-network. The super-network consists of 20 sequentially stacked choice blocks. Each choice block can select one of four operations: $k \times k$ convolutional blocks ($k \in 3, 5, 7$) inspired by ShuffleNetV2 [113] and a 3×3 Xception [114] inspired convolutional block. Additionally, each choice block can also select from 10 different channel choices $0.2 - 2.0\times$. During training we use the loss formulation in Eq. 4.2 to train this super-network whereby, in each batch a new architecture is sampled uniformly [91] and only the weights corresponding to it are updated.

Search Strategy: To search for the most compatible architecture, CMP-NAS uses evolutionary search [91] fitted with the different rewards outlined in Tab. 4.1. The search is fast because each architecture inherits the weights from the super-network. In the end, we obtain the five best architectures and re-train them from scratch with BCT.

4.4 Experiments

We evaluate the efficacy of our heterogeneous system on two tasks: face retrieval, as it is one of the "open-universe" problems with the largest publicly available datasets; and fashion retrieval which necessitates an open-set treatment due to the constant evolution of fashion items. We use face retrieval as the main benchmark for our ablation studies.

4.4.1 Datasets, Metrics and Gallery Model

Face Retrieval: We use the IMDB-Face dataset [115] to train the embedding model for the face retrieval task. The IMDB-Face dataset contains over 1.7M images of about 59k celebrities. If not otherwise specified, we use 95% of the data as training set to train our embedding model and use the remaining 5% as a validation dataset to compute the rewards for architecture search. For testing, we use the widely used IJB-C face recognition benchmark dataset [116]. The performance is evaluated using the true positive identification rate at a false positive identification rate of 10^{-1} (TPIR@FPIR=10^{-1}). Throughout the evaluation, we use a ResNet-101 as the fixed gallery model ϕ_g.

Fashion Retrieval: We evaluate the proposed method on Commercial-Consumer Clothes Retrieval task on DeepFashion2 dataset [117]. It contains 337K commercial-consumer clothes pairs in the training set, from which 90% of the data is used for training the embedding and the rest 10% is used for computing the rewards in architecture search. We report the test accuracy using Top-10 retrieval accuracy on the original validation set, which contains 10,844 consumer images with 12,377 query items, and 21,309 commercial images with 36,961 items in the gallery. ResNet-101 is used as the fixed gallery model ϕ_g.

4.4.2 *Implementation Details*

Our query and gallery models take a 112×112 image as input and output an embedding vector of 128 dimensions.

Face retrieval: We use mis-alignment and color distortion for data augmentation [88]. Following recent state-of-the-art [111], we train our gallery ResNet-101 model using the cosine margin loss [111] with margin set to 0.4. We use the SGD optimizer with weight decay 5×10^{-4}. The initial learning rate is set to 0.1 which decreases to 0.01, 0.001 and 0.0001 after 8, 12 and 14 epochs. Our gallery model is trained for 16 epochs with a batch size 320. We train the query models for 32 epochs with a cosine learning rate decay schedule [118]. The initial learning rate is set to 1.3 all query models except MobileNetV1($1\times$) which uses 0.1.

Fashion retrieval: The original fashion retrieval task with DeepFashion2 [117] requires to first detect and then retrieve fashion items. Since we only tackle the retrieval task, we construct our retrieval-only dataset by using ground truth bounding box annotations to extract the fashion items. To train the gallery model, we follow [110] in using normalized cross entropy loss with temperature 0.5. The gallery model is trained for 40 epochs using an initial learning rate of 3.0 with cosine decay. The weight decay is set to 10^{-4}. Our query models are trained with BCT for 80 epochs using an initial learning rate of 10 with cosine decay.

Runtime: On a system containing 8 Tesla V100 GPUs, the entire pipeline for the face (and fashion) retrieval takes roughly 100 (45) hours. This includes roughly 8 (8) hours to train the gallery model, 32 (14) hours to train the query super-network, 48 (20) hours for evolutionary search and 2 (2) hours to train the final query architecture.

For additional implementation details specific to CMP-NAS, please refer to the supplementary material.

Table 4.2: Comparison with baseline and paragon for 1:N face retrieval on IJB-C. Accuracy is reported as TPIR(%)@FPIR=10^{-1}. All the models except the paragon are trained with BCT loss using ResNet-101 as the "teacher".

	Gallery Model	Query Model	Acc. (%)	Query Flops (M)
Paragon	ResNet-101	ResNet-101	86.7	7597
Proposed	ResNet-101	CMP-NAS	85.1	327
Baseline	MobileNetV2	MobileNetV2	77.1	329

Table 4.3: Comparison with baseline and paragon for fashion retrieval on DeepFashion2. Accuracy is reported as Top-10 retrieval accuracy. All the models except the paragon are trained with BCT loss using ResNet-101 as the "teacher".

	Gallery Model	Query Model	Acc. (%)	Query Flops (M)
Paragon	ResNet-101	ResNet-101	65.2	7597
Proposed	ResNet-101	CMP-NAS	64.9	211
Baseline	MobileNetV3	MobileNetV3	62.7	226

4.4.3 Baseline and paragon for visual search

Since gallery features can be pre-computed and there is no computational constraint on the gallery side, we fix the gallery model to a ResNet-101. In terms of visual search accuracy, the paragon is achieved by the (ResNet-101, ResNet-101) system on both the face and fashion retrieval tasks. On the other hand, we use (MobileNetV2, MobileNetV2) and (MobileNetV3, MobileNetV3) as the baseline for face and fashion retrieval respectively, since they achieve the highest accuracy among the MobileNet family.

Tab. 4.2 shows almost a 10% gap in accuracy between the paragon and baseline for face retrieval. On the other end, the baseline consumes $23\times$ fewer query flops than the paragon. This establishes the goal of our heterogeneous system: To achieve accuracy similar to the paragon while consuming query flops similar to the baseline. Indeed, the middle rows of Tab. 4.2 shows this goal is achieved by our proposed heterogeneous system (ResNet-101, CMP-NAS) which consumes similar query flops as the baseline with only 1.6% accuracy drop compared to the paragon. Tab. 4.3 reveals a similar observation on DeepFashion2.

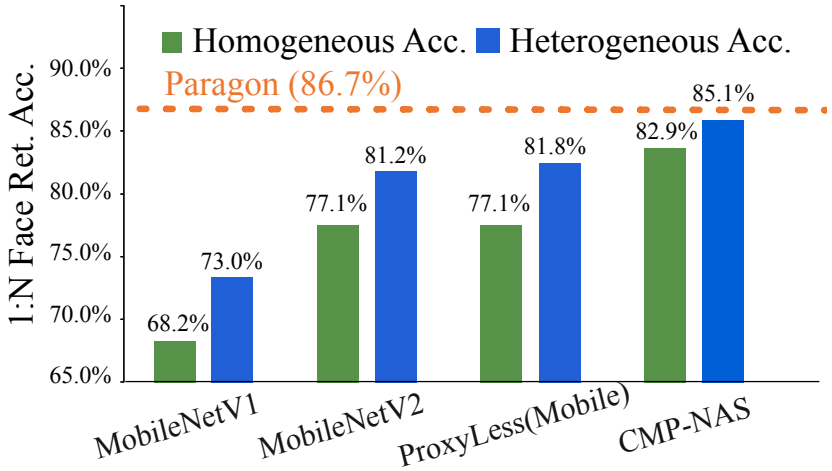
4.4.4 Dissecting the performance of CMP-NAS

In this subsection, we break down the accuracy achieved by our heterogeneous system in terms of the improvement due to (1) weights and (2) architecture compatibility.

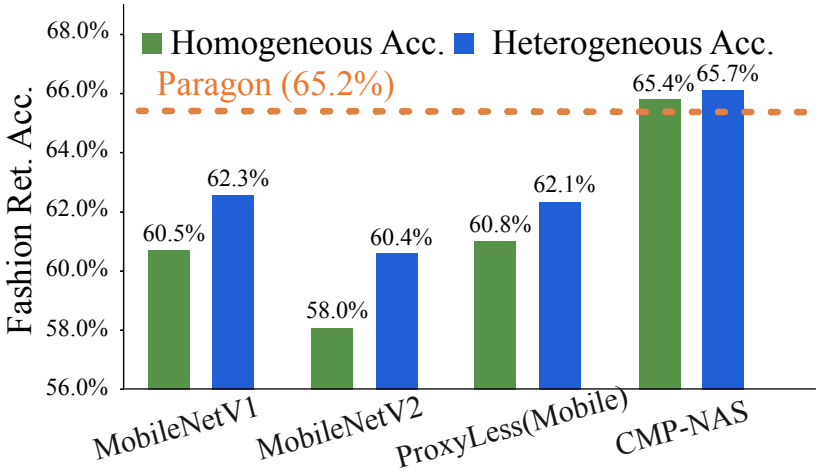
Improvement due to weight-compatibility: To observe the improvement due to weight compatibility, Fig. 4.5 (a),(b) shows the homogeneous and heterogeneous accuracy obtained by three state-of-the-art query models with *static* architectures in the 300 Million flops range trained with BCT (Eq. 4.1). We observe that heterogeneous system outperforms homogeneous system on average by 3.95% and 1.45% for face and fashion retrieval respectively. This indicates that considering weight-compatibility alone is beneficial.

Improvement due to architecture compatibility: To see the additional benefits due to architecture compatibility, in Fig. 4.5 we compare the heterogeneous accuracy achieved

by the query models obtained via CMP-NAS and trained with BCT. On IJB-C and DeepFashion2 datasets, CMP-NAS outperforms the second-best model ProxyLess(Mobile) by 3.3%, 3.6% in terms of heterogeneous accuracy. This shows architecture compatibility can improve accuracy by a large margin. Additionally, we also observe gains of 5.8% and 4.8% in terms of homogeneous accuracy.



(a) 1:N Face retrieval accuracy (TPIR@FPIR=10^{-1}) on IJB-C.



(b) Fashion retrieval accuracy (top-10) on DeepFashion2.

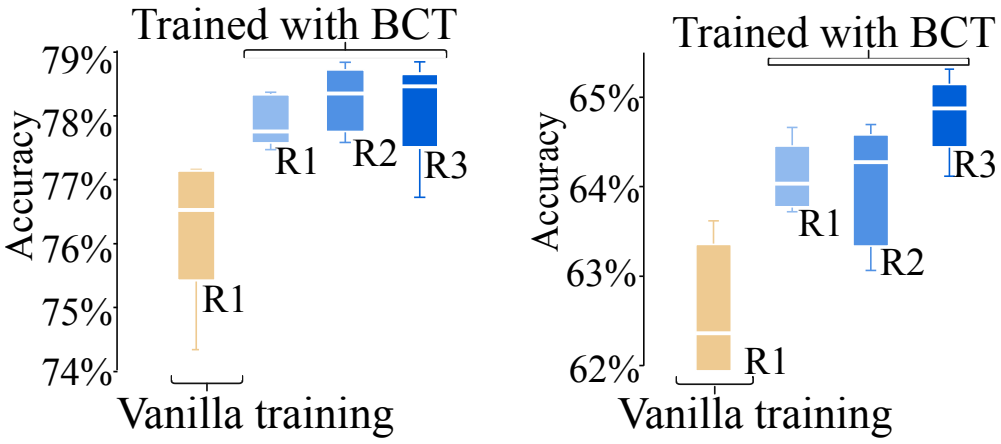
Figure 4.5: Evaluating the heterogeneous and homogeneous search accuracy for face and fashion retrieval tasks using different query models. CMP-NAS outperforms other baselines and achieves accuracy close to the paragon.

Comparing different methods for weight-compatibility: In Sec. 4.3.2, we discussed four ways to achieve weight-compatibility; (1) vanilla training, (2) knowledge distillation [12], (3) fine-tuning, and (4) BCT [88]. To quantitatively compare these methods, we obtain the query network by pruning the gallery ResNet-101 model using magnitude pruning [17] and channel pruning [20]. To obtain the (pruned) query model, we prune 90% of filters from

the first two layers in each residual block of the gallery network. The query model obtained is trained with each of the four methods and Tab. 4.4 shows both the homogeneous and heterogeneous search accuracy. We observe only fine-tuning and BCT can achieve weight-compatibility wherein accuracy of the heterogeneous system supersedes that of the homogeneous system. Training from scratch and knowledge distillation on the other hand, cannot ensure compatibility and obtain 0.0% accuracy for heterogeneous search. Among fine-tuning and BCT, we prefer BCT to ensure weight compatibility for two reasons: (1) fine-tuning is restrictive: it poses a strong requirement on the query architecture *e.g.*, query model is obtained by pruning the gallery model and (2) the model trained with BCT performs better.

Table 4.4: Comparing techniques for achieving weight-level compatibility on the 1:N face retrieval task. The query model ϕ_q is obtain by pruning 90% of filters in the first two layers of each residual block of the gallery model. We see that for both pruning methods, training the query model with BCT loss leads to the highest heterogeneous accuracy.

Gallery model	Query model	Train Scratch	Finetune	BCT	KD
Magnitude prune	Magnitude prune	84.4	84.9	86.4	86.8
ResNet-101	Magnitude prune	0.0	86.5	87.2	0.0
Channel prune	Channel prune	84.2	85.2	86.5	87.0
ResNet-101	Channel prune	0.0	86.3	87.4	0.0



(a) 1:N search on IJB-C.

(b) Top-10 on DeepFashion2.

Figure 4.6: Ablating on training strategies (vanilla, BCT) and rewards ($\mathcal{R}_1 - \mathcal{R}_3$) for CMP-NAS These plots show the heterogeneous accuracy of the best 5 models (under 100 Mflops) discovered by each method and trained from scratch with BCT. Observe that the ingredients of CMP-NAS *i.e.*, BCT training + reward $\mathcal{R}_3$ perform the best.

Comparing CMP-NAS with baseline NAS[91]: To measure the gains relative to baseline

NAS, in Fig. 4.6, we present a barplot of the heterogeneous accuracy achieved by the best 5 architectures obtained by vanilla NAS (yellow bar) and CMP-NAS (blue bars) when trained from scratch using BCT. The baseline considers the vanilla loss ($\lambda_2 = 0$ in Eq. 4.1) to train the super-network and searches using reward $\mathcal{R}_1$ while CMP-NAS uses BCT to train the super-network and can search using rewards $\mathcal{R}_1, \mathcal{R}_2, \mathcal{R}_3$. On both datasets, CMP-NAS outperforms the baseline by $2 - 2.5\%$.

Comparing reward choices for CMP-NAS: In Fig. 4.6, the performance of different reward choices (of Tab. 4.1) are shown by blue plots. As expected, the baseline reward ($\mathcal{R}_1$) performs worst since its target (homogeneous accuracy) is misaligned with our target (heterogeneous accuracy). The second reward ($\mathcal{R}_2$) is much better since it directly optimizes the target while the composite reward ($\mathcal{R}_3$) works best with especially large gains observed on DeepFashion2.

Table 4.5: Evaluating architectures searched with CMP-NAS for fashion retrieval (denoted as fashion) and face retrieval (denoted as face) tasks. We search models for three different complexity tiers: 100, 230 and 330 Mflops and use the best architecture to report the results. The searched models outperform other architectures by $3 \sim 5\%$ on both the tasks.

Gallery model	Query model	Query MFlops	Fashion retrieval top-10 search	Face retrieval 1:N search
ResNet-101	ResNet-101	7597	65.1	86.7
ResNet-101	MobileNetV1(1x)	579	62.3	73.0
	MobileNetV2(1x)	329	60.4	77.0
	ProxyLess(mobile)	332	62.1	81.8
	CMP-NAS-a(Face)	**327**	65.4	**85.1**
	CMP-NAS-a(Fashion)	**314**	**65.7**	84.2
ResNet-101	MobileNetV3	226	63.0	80.4
	CMP-NAS-b(Face)	**216**	64.4	**84.1**
	CMP-NAS-b(Fashion)	**211**	**64.9**	81.5
ResNet-101	MobileNetV1(0.5x)	155	60.3	62.7
	ShuffleNetV2(1x)	149	63.3	75.8
	ShuffleNetV1(1x,g=1)	148	62.6	76.0
	MobileNetV2(0.5x)	100	62.0	73.1
	CMP-NAS-c(Face)	**94**	62.4	**78.8**
	CMP-NAS-c(Fashion)	**93**	**64.8**	77.8

4.4.5 Generalization performance of CMP-NAS

In this section, we investigate the performance of CMP-NAS under different compute constraints, application scenarios and tasks. Inspired by state-of-the-art architectures for

mobile deployment we select three computational tiers: 330 million flops (similar to MobileNetV2), 230 Mflops (similar to MobileNetV3), and 100 Mflops (similar to ShuffleNetV2). For each computational tier, we implement a heterogeneous system using the models searched by CMP-NAS. These models are denoted by CMP-NAS-a (330 Mflops), CMP-NAS-b (230 Mflops), and CMP-NAS-c (100 Mflops). Additionally, we append "(Face)"/"(Fashion)" to the model name, *e.g.*, "CMP-NAS-a(Face)"/"CMP-NAS-a(Fashion)", to denote the architecture searched on the face or fashion datasets respectively.

CMP-NAS for different resource constraints: We compare the performance of architectures searched by CMP-NAS for each computational tier in Tab. 4.5. For each task, we look at the model searched on the same task *e.g.*, for face retrieval we look at CMP-NAS-a(Face) *etc*. On both datasets the models searched by CMP-NAS consistently outperform other state-of-the-art baselines. For 330 Mflops tier, CMP-NAS-a outperforms the second best (ProxyLess(Mobile) [90]) by 3.6% and by 3.1% on the fashion and face retrieval tasks respectively. Similarly, CMP-NAS-b outperforms the second best network MobileNetV3 [119] by 1.9% and 3.7% on the corresponding tasks. Finally, for the 100M tier, CMP-NAS-c achieves 1.5% and 3.0% improvement over the second best network ShuffleNetV2($1\times$) while consuming 33% fewer flops. These results establish the generalization ability of the CMP-NAS for different computation constraints.

CMP-NAS across different applications: For this experiment, we evaluate the architectures searched on the face dataset for the fashion retrieval task and vice versa. The results are shown in the Tab. 4.5. We observe that the architectures optimized for the face tasks CMP-NAS-a/b/c(Face) also outperform the baselines for the fashion retrieval task. Moreover, these models only lose $1-2\%$ accuracy compared to the best model searched on the fashion dataset. We make similar observations for CMP-NAS-a/b/c/(Fashion) evaluated

Table 4.6: Evaluating the models CMP-NAS-a,b,c(Face) on the 1:1 face verification task using IJB-C. Accuracy metric is TAR@FAR=10^{-4}. The searched models outperform the baselines indicating they can generalize across tasks.

Gallery model	Query model	Query MFlops	Homogeneous accuracy	Heterogeneous accuracy
ResNet-101	ResNet-101	7597	85.4	-
ResNet-101	ProxyLess(mobile)	332	75.5	80.3
	CMP-NAS-a(Face)	**327**	**81.6**	**84.5**
ResNet-101	MobileNetV3	226	74.3	79.9
	CMP-NAS-b(Face)	**216**	**79.0**	**82.8**
ResNet-101	ShuffleNetV2(1x)	149	66.8	74.8
	CMP-NAS-c(Face)	**94**	**71.5**	**78.3**

on the face retrieval task. This shows that the architectures searched by CMP-NAS can generalize across application scenarios.

CMP-NAS for face verification: In table Tab. 4.6, we use the CMP-NAS-a/b/c(Face) models for another "open universe" problem: 1:1 face verification. The results show that the models searched by CMP-NAS outperform state-of-the art architectures by $3-5\%$ in the homogeneous and heterogeneous settings. Importantly, the compatibility rule is also achieved. This indicates the searched models can generalize across different tasks.

4.5 Discussion and Conclusion

We have presented a heterogeneous visual search system that achieves high accuracy with low computational cost. Key to building this system is ensuring the query and gallery models are compatible. We achieve this through joint weight and architecture compatibility optimization with CMP-NAS. There are, however, some limitations of our method: (1) Our method is limited to classification-based embedding training and does not directly work with metric learning based approaches; (2) We consider the simplified use-case for architecture optimization wherein the gallery model is fixed. A more general treatment of model compatibility may optimize both the gallery and query models. These limitations show that there is scope for improving our HVS system which can be tackled by future work.

Part II

New methods for robust edge AI

Overview

In the previous part, we focused on efficiency of Deep Neural Networks. While efficiency necessary for edge deployment, it is not sufficient. Since edge scenarios often entail noisy environments, it is critical to ensure the neural networks are also resilient to noise. In REST, we jointly tackle the challenges of *efficiency* and *robustness* to adversarial and gaussian noise for the task of home based sleep monitoring.

> **Chapter 5: REST:Robust and Efficient Neural Networks for Sleep Monitoring in the Wild.** Rahul Duggal, Scott Freitas, Cao Xiao, Duen Horng Chau, Jimeng Sun. In *Proceedings of The Web Conference (WWW)*, 2020. `https://dl.acm.org/doi/10.1145/3366423.3380241`

Noise can be of many types. In REST we tackle test time noise, while in HAR we tackle training time noise that manifests as class imbalance in the data distribution. We develop a hardness aware re-weighting strategy that works with existing loss functions to improve accuracy while leading to upto 20% inference flops savings.

> **Chapter 6: HAR:Hardness Aware Reweighting for Imbalanced Datasets.** Rahul Duggal, Scott Freitas, Sunny Dhamnani, Duen Horng Chau, Jimeng Sun. *IEEE International Conference on Big Data (Big Data).*, 2021. `https://ieeexplore.ieee.org/document/9671807`

In IMB-NAS, we jointly tackle class imbalance and efficiency by searching for optimal backbone architectures via neural architecture search on imbalanced datasets.

> **Chapter 7: IMB-NAS:Neural Architecture Search for Imbalanced Datasets.** Rahul Duggal, Sheng-Yun Peng, Hao Zhou, Duen Horng Chau. *In submission.*, 2022. `N/A`

CHAPTER 5
REST:ROBUST AND EFFICIENT NEURAL NETWORKS FOR SLEEP MONITORING IN THE WILD

In recent years, significant attention has been devoted towards integrating deep learning technologies in the healthcare domain. However, to safely and practically deploy deep learning models for home health monitoring, two significant challenges must be addressed: the models should be (1) robust against noise; and (2) compact and energy-efficient. We propose REST, a new method that simultaneously tackles both issues via 1) *adversarial training* and controlling the Lipschitz constant of the neural network through *spectral regularization* while 2) enabling neural network compression through *sparsity regularization*. We demonstrate that REST produces highly-robust and efficient models that substantially outperform the original full-sized models in the presence of noise. For the sleep staging task over single-channel electroencephalogram (EEG), the REST model achieves a macro-F1 score of 0.67 vs. 0.39 achieved by a state-of-the-art model in the presence of Gaussian noise while obtaining $19\times$ parameter reduction and $15\times$ MFLOPS reduction on two large, real-world EEG datasets. By deploying these models to an Android application on a smartphone, we quantitatively observe that REST allows models to achieve up to $17\times$ energy reduction and $9\times$ faster inference. We open source the code repository with this paper: `https://github.com/duggalrahul/REST`.

5.1 Introduction

As many as 70 million Americans suffer from sleep disorders that affects their daily functioning, long-term health and longevity. The long-term effects of sleep deprivation and sleep disorders include an increased risk of hypertension, diabetes, obesity, depression, heart attack, and stroke [120]. The cost of undiagnosed sleep apnea alone is estimated to exceed 100 billion in the US [121].

A central tool in identifying sleep disorders is the **hypnogram**—which documents the progression of sleep stages (**REM** stage, **Non-REM** stages **N1** to **N3**, and **Wake** stage) over an entire night (see Fig. 5.1, top). The process of acquiring a hypnogram from raw sensor data is called **sleep staging**, which is the focus of this work. Traditionally, to reliably obtain a hypnogram the patient has to undergo an overnight sleep study—called *polysomnography* (PSG)—at a sleep lab while wearing bio-sensors that measure physiological signals, which include electroencephalogram (EEG), eye movements (EOG), muscle activity or skeletal

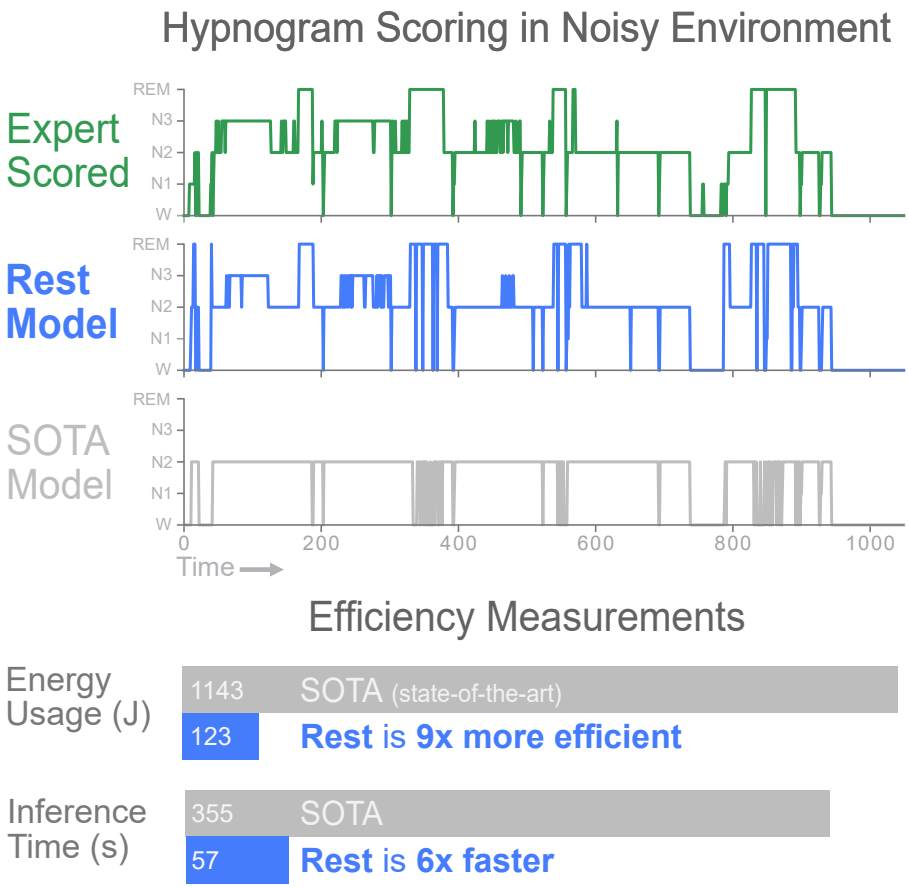


Figure 5.1: Top: we generate hypnograms for a patient in the SHHS test set. In the presence of Gaussian noise, our REST-generated hypnogram closely matches the contours of the expert-scored hypnogram. Hypnogram generated by a state-of-the-art (SOTA) model by Sors et al. [7] is considerably worse. Bottom: we measure energy consumed (in Joules) and inference time (in seconds) on a smartphone to score one night of EEG recordings. REST is 9X more energy efficient and 6X faster than the SOTA model.

muscle activation (EMG), and heart rhythm (ECG). The PSG data is then analyzed by a trained sleep technician and a certified sleep doctor to produce a PSG report. The hypnogram plays an essential role in the PSG report, where it is used to derive many important metrics such as sleep efficiency and apnea index. Unfortunately, manually annotating this PSG is both costly and time consuming for the doctors. Recent research has proposed to alleviate these issues by automatically generating the hypnogram directly from the PSG using deep neural networks [122, 123]. However, the process of obtaining a PSG report is still *costly* and *invasive* to patients, reducing their participation, which ultimately leads to undiagnosed sleep disorders [124].

One promising direction to reduce undiagnosed sleep disorders is to enable sleep monitoring at the home using commercial wearables (e.g., Fitbit, Apple Watch, Emotiv) [125]. However, despite significant research advances, a recent study shows that wearables using

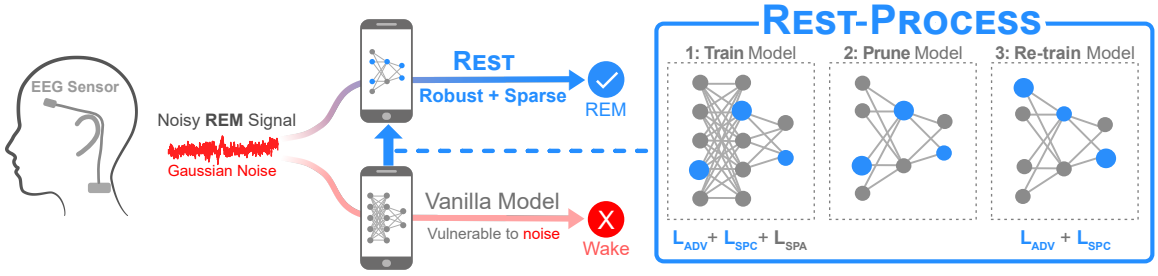


Figure 5.2: REST Overview: (from left) When a noisy EEG signal belonging to the REM (rapid eye movement) sleep stage enters a traditional neural network which is vulnerable to noise, it gets wrongly classified as a Wake sleep stage. On the other hand, the same signal is correctly classified as the REM sleep stage by the REST model which is both robust and sparse. (From right) REST is a three step process involving (1) training the model with adversarial training, spectral regularization and sparsity regularization (2) pruning the model and (3) re-training the compact model.

a single sensor (e.g., single lead EEG) often have lower performance for sleep staging, indicating a large room for improvement [126].

5.1.1 Contributions

Our contributions are two-fold—(i) we identify emerging research challenges for the task of sleep monitoring in the wild; and (ii) we propose REST, a novel framework that addresses these issues.

I. New Research Challenges for Sleep Monitoring.

- **C1. Robustness to Noise.** We observe that state-of-the-art deep neural networks (DNN) are highly susceptible to environmental noise (Fig. 5.1, top). In the case of wearables, noise is a serious consideration since bioelectrical signal sensors (e.g., electroencephalogram "EEG", electrocardiogram "ECG") are commonly susceptible to *Gaussian* and *shot* noise, which can be introduced by electrical interferences (e.g., power-line) and user motions (e.g., muscle contraction, respiration) [127, 128, 129, 130]. This poses a need for noise-tolerant models. In this paper, we show that adversarial training and spectral regularization can impart significant noise robustness to sleep staging DNNs (see top of Fig 5.1).

- **C2. Energy and Computational Efficiency.** Mobile deep learning systems have traditionally offloaded compute intensive inference to cloud servers, requiring transfer of sensitive data and assumption of available Internet. However, this data uploading process is difficult for many healthcare scenarios because of—(1) **privacy**: individuals are often reluctant to share health information as they consider it highly sensitive; and (2)

accessibility: real-time home monitoring is most needed in resource-poor environments where high-speed Internet may not be reliably available. Directly deploying a neural network to a mobile phone bypasses these issues. However, due to the constrained computation and energy budget of mobile devices, these models need to be fast in speed and parsimonious with their energy consumption.

II. Noise-robust and Efficient Sleep Monitoring. Having identified these two new research challenges, we propose **REST**, the first framework for developing noise-robust and efficient neural networks for home sleep monitoring (Fig. 5.2). Through REST, our major contributions include:

- "**R**obust and **E**fficient Neural Networks for **S**leep Moni**t**oring" By integrating a novel combination of three training objectives, REST endows a model with noise robustness through (1) *adversarial training* and (2) *spectral regularization*; and promotes energy and computational efficiency by enabling compression through (3) *sparsity regularization*.

- **Extensive evaluation** We benchmark the performance of REST against competitive baselines, on two real-world sleep staging EEG datasets—Sleep-EDF from Physionet and Sleep Heart Health Study (SHHS). We demonstrate that REST produces highly compact models that substantially outperform the original full-sized models in the presence of noise. REST models achieves a macro-F1 score of 0.67 vs. 0.39 for the state-of-the-art model in the presence of Gaussian noise, with $19\times$ parameter and $15\times$ MFLOPS reduction.

- **Real-world deployment.** We deploy a REST model onto a Pixel 2 smartphone through an Android application performing sleep staging. Our experiments reveal REST achieves $17\times$ energy reduction and $9\times$ faster inference on a smartphone, compared to uncompressed models.

5.2 Related Work

In this section we discuss related work from three areas—(1) the task of sleep stage prediction, (2) robustness of deep neural networks and (3) compression of deep learning models.

5.2.1 Sleep-Stage Prediction

Sleep staging is the task of annotating a polysomnography (PSG) report into a hypnogram, where 30 second sleep intervals are annotated into one of five sleep stages (W, N1, N2, N3, REM). Recently, significant effort has been devoted towards automating this annotation

process using deep learning [7, 122, 131, 132, 133, 134], to name a few. While there exists a large body of research in this area—two works in particular look at both single channel [122] and multi-channel [131] deep learning architectures for sleep stage prediction on EEG. In [122], the authors develop a deep learning architecture (SLEEPNET) for sleep stage prediction that achieves expert-level accuracy on EEG data. In [131], the authors develop a multi-modal deep learning architecture for sleep stage prediction that achieves state-of-the-art accuracy. As we demonstrate later in this paper (Section 5.4.5), these sleep staging models are frequently susceptible to noise and suffer a large performance drop in its presence (see Figure 5.1). In addition, these DNNs are often overparameterized (Section 5.4.6), making deployment to mobile devices and wearables difficult. Through REST, we address these limitations and develop noise robust and efficient neural networks for edge computing.

5.2.2 Noise & Adversarial Robustness

Adversarial robustness seeks to ensure that the output of a neural network remains unchanged under a bounded perturbation of the input; or in other words, prevent an adveresary from maliciously perturbing the data to fool a neural network. Adversarial deep learning was popularized by [39], where they showed it was possible to alter the class prediction of deep neural network models by carefully crafting an adversarially perturbed input. Since then, research suggests a strong link between adversarial robustness and noise robustness [40, 41, 42]. In particular, [40] found that by performing adversarial training on a deep neural network, it becomes robust to many forms of noise (e.g., Gaussian, blur, shot, etc.). In contrast, they found that training a model on Gaussian augmented data led to models that were less robust to adversarial perturbations. We build upon this finding of adversarial robustness as a proxy for noise robustness and improve upon it through the use of spectral regularization; while simultaneously compressing the model to a fraction of its original size for mobile devices.

5.2.3 Model Compression

Model compression aims to learn a reduced representation of the weights that parameterize a neural network; shrinking the computational requirements for memory, floating point operations (FLOPS), inference time and energy. Broadly, prior art can be classified into four directions—pruning [8], quantization [10], low rank approximation [11] and knowledge distillation [12]. For REST, we focus on structured (channel) pruning thanks to its performance benefits (speedup, FLOP reduction) and ease of deployment with regular hardware.

In structured channel pruning, the idea is to assign a measure of importance to each filter of a convolutional neural network (CNN) and achieve desired sparsity by pruning the least important ones. Prior work demonstrates several ways to estimate filter importance—magnitude of weights [17], structured sparsity regularization [22], regularization on activation scaling factors [19], filter similarity [135] and discriminative power of filters [21]. Recently there has been an attempt to bridge the area of model compression with adversarial robustness through connection pruning [136] and quantization [137]. Different from previous work, REST aims to compress a model by pruning whole filters while imparting noise tolerance through adversarial training and spectral regularization. REST can be further compressed through quantization [137].

5.3 REST: Noise-Robust & Efficient Models

REST is a new method that simultaneously compresses a neural network while developing both noise and adversarial robustness.

5.3.1 Overview

Our main idea is to enable REST to endow models with these properties by integrating three careful modifications of the traditional training loss function. (1) The *adversarial training* term, which builds noise robustness by training on adversarial examples (Section 5.3.2); (2) the *spectral regularization* term, which adds to the noise robustness by constraining the Lipschitz constant of the neural network (Section 5.3.3); and (3) the sparsity regularization term that helps to identify important neurons and enables compression (Section 5.3.4). Throughout the paper, we follow standard notation and use capital bold letters for matrices (e.g., $\widetilde{A}$), lower-case bold letters for vectors (e.g., $\widetilde{A}$).

5.3.2 Adversarial Training

The goal of adversarial training is to generate noise robustness by exposing the neural network to adversarially perturbed inputs during the training process. Given a neural network $f(\widetilde{X}; \widetilde{W})$ with input $\widetilde{X}$, weights $\widetilde{W}$ and corresponding loss function $L(f(\widetilde{X}; \widetilde{W}), y)$, adversarial training aims at solving the following min-max problem:

$$\min_{\widetilde{W}} \left[\mathop{\mathbb{E}}_{\widetilde{X}, y \sim D} \left(\max_{\delta \in S} \; L(f(\widetilde{X} + \delta; \widetilde{W}), y) \right) \right] \tag{5.1}$$

Here D is the unperturbed dataset consisting of the clean EEG signals $\widetilde{X} \in \mathbb{R}^{K_{in} \times K_L}$ (K_{in} is the number of channels and K_L is the length of the signal) along with their cor-

responding label y. The inner maximization problem in (5.1) embodies the goal of the adversary—that is, produce adversarially perturbed inputs (i.e., $\widetilde{X} + \delta$) that maximize the loss function L. On the other hand, the outer minimization term aims to build robustness by countering the adversary through minimizing the expected loss on perturbed inputs.

Maximizing the inner loss term in (5.1) is equivalent to finding the adversarial signal $\widetilde{X}_p = \widetilde{X} + \delta$ that maximally alters the loss function L within some bounded perturbation $\delta \in S$. Here S is the set of allowable perturbations. Several choices exist for such an adversary. For REST, we use the iterative Projected Gradient Descent (PGD) adversary since it's one of the strongest first order attacks [138]. Its operation is described below in Equation 5.2.

$$\widetilde{X}_p^{(t+1)} = \widetilde{X}_p^{(t)} + \Pi_\tau \Big[\epsilon \cdot sign \Big\{ \nabla_{\widetilde{X}_p^{(t)}} L \left(f(\widetilde{X}_p^{(t)}; \widetilde{W}), y \right) \Big\} \Big] \tag{5.2}$$

Here $\widetilde{X}_p^{(0)} = \widetilde{X}$ and at every step t, the previous perturbed input $\widetilde{X}_p^{(t-1)}$ is modified with the sign of the gradient of the loss, multiplied by ϵ (controls attack strength). Π_τ is a function that clips the input at the positions where it exceeds the predefined L_∞ bound τ. Finally, after n_{iter} iterations we have the REST adversarial training term L_{adv} in Equation 5.3.

$$L_{adv} = L(f(\widetilde{X}_p^{(n_{iter})}; \widetilde{W}), y) \tag{5.3}$$

5.3.3 Spectral Regularizer

The second term in the objective function is the spectral regularization term, which aims to constrain the change in output of a neural network for some change in input. The intuition is to suppress the amplification of noise as it passes through the successive layers of a neural network. In this section we show that an effective way to achieve this is via constraining the Lipschitz constant of each layer's weights.

For a real valued function $f : \mathbb{R} \to \mathbb{R}$ the Lipschitz constant is a positive real value C such that $|f(x_1) - f(x_2)| \leq C|x_1 - x_2|$. If $C > 1$ then the change in input is magnified through the function f. For a neural net, this can lead to input noise amplification. On the other hand, if $C < 1$ then the noise amplification effect is diminished. This can have the unintended consequence of reducing the discriminative capability of a neural net. Therefore our goal is to set the Lipschitz constant $C = 1$. The Lipschitz constant for the l^{th} fully connected layer parameterized by the weight matrix $\widetilde{W}^{(l)} \in \mathbb{R}^{K_{in} \times K_{out}}$ is equivalent to its spectral norm $\rho(\widetilde{W}^{(l)})$ [139]. Here the spectral norm of a matrix $\widetilde{W}$ is the square root of the largest singular value of $\widetilde{W}^T \widetilde{W}$. The spectral norm of a 1-D convolutional layer

Algorithm 1: Noise Robust & Efficient Neural Network Training (REST)

Input: Model weights $\widetilde{W}$, EEG signal $\widetilde{X}$ and label y from dataset $\widetilde{D}$, spectral regularization λ_o, sparsity regularization λ_g, learning rate α, perturbation strength ϵ, maximum PGD iterations n_{iter} and model sparsity s

Output: Noise robust, compressed neural network

(1) **Train the full model with REST loss L_R:**

for *epoch = 1 to N* **do**
  for *minibatch* $B \subset D$ **do**
    for $\widetilde{X} \in B$ **do**
      $\widetilde{X}_p^{(1)} = \widetilde{X}$
      for *k=1 to* n_{iter} **do**
        $\widetilde{X}_p^{(k+1)} = \widetilde{X}_p^{(k)} + \Pi_\tau(\epsilon \cdot sign(\nabla_{\widetilde{X}_p^{(k)}} L(f(\widetilde{X}_p^{(k)}; \widetilde{W}), y)))$
      $\widetilde{W}_{grad} \leftarrow \mathop{\mathbb{E}}_{\widetilde{X}, y \sim D} |\nabla_{\widetilde{W}} L_R(\widetilde{X}_p, y; \widetilde{W})|$
      where $L_R =$

$$\underbrace{L(f(\widetilde{X}_p; \widetilde{W}), y)}_{\text{adversarial training}} + \lambda_o \underbrace{\sum_{\text{layer l=1}}^{N} \|(\widetilde{W}^{(l)})^T \widetilde{W}^{(l)} - \widetilde{I}\|_2}_{\text{spectral regularization}} + \lambda_g \underbrace{\sum_{\text{layer l=1}}^{N} \|\gamma^{(l)}\|_1}_{\text{sparsity regularization}}$$

      $\widetilde{W} \leftarrow \widetilde{W} - \alpha \cdot \widetilde{W}_{grad}$

(2) **Prune the trained model:**

Globally prune filters from $\widetilde{W}$ having smallest γ values until $\frac{n_f(\widetilde{W'})}{n_f(\widetilde{W})} \leq s$.

Constrain layerwise sparsity so $\frac{n_f(\widetilde{W'^{(l)}})}{n_f(\widetilde{W}^{(l)})} \geq 0.1$.

(3) **Re-train the pruned model:**

Retrain compressed network $f(\widetilde{X}; \widetilde{W'})$ using *adversarial training* and *spectral regularization* (no sparsity regularization).

parameterized by the tensor $\widetilde{W}^{(l)} \in \mathbb{R}^{K_{out} \times K_{in} \times K_l}$ can be realized by reshaping it to a matrix $\widetilde{W}^{(l)} = \mathbb{R}^{K_{out} \times (K_{in} K_l)}$ and then computing the largest singular value.

A neural network of N layers can be viewed as a function $f(\cdot)$ composed of N sub-functions $f(x) = f_1(\cdot) \circ f_2(\cdot) \circ ... f_N(x)$. A loose upper bound for the Lipschitz constant of f is the product of Lipschitz constants of individual layers or $\rho(f) \leq \prod_{i=1}^{N} \rho(f_i)$ [139]. The overall Lipschitz constant can grow exponentially if the spectral norm of each layer is greater than 1. On the contrary, it could go to 0 if spectral norm of each layer is between 0 and 1. Thus the ideal case arises when the spectral norm for each layer equals 1. This can be achieved in several ways [140, 139, 141], however, one effective way is to encourage orthonormality in the columns of the weight matrix $\widetilde{W}$ through the minimization of $\|\widetilde{W}^T \widetilde{W} - \widetilde{I}\|$ where $\widetilde{I}$ is the identity matrix. This additional loss term helps regulate

the singular values and bring them close to 1. Thus we incorporate the following spectral regularization term into our loss objective, where λ_o is a hyperparameter controlling the strength of the spectral regularization.

$$L_{Spectral} = \lambda_o \sum_{i=1}^{N} \|(\widetilde{W}^{(i)})^T \widetilde{W}^{(i)} - \widetilde{I}\|_2 \tag{5.4}$$

5.3.4 Sparsity Regularizer & REST *Loss Function*

The third term of the REST objective function consists of the sparsity regularizer. With this term, we aim to learn the important filters in the neural network. Once these are determined, the original neural network can be pruned to the desired level of sparsity.

The incoming weights for filter i in the l^{th} fully connected (or 1-D convolutional) layer can be specified as $\overline{W}^{(l)}_{i,:} \in \mathbb{R}^{K_{in}}$ (or $\overline{W}^{(l)}_{i,:,:} \in \mathbb{R}^{K_{in} \times K_L}$). We introduce a per filter multiplicand $\gamma^{(l)}_i$ that scales the output activation of the i^{th} neuron in layer l. By controlling the value of this multiplicand, we realize the importance of the neuron. In particular, zeroing it amounts to dropping the entire filter. Note that the L_0 norm on the multiplicand vector $\|\boldsymbol{\gamma}^{(l)}\|_0$, where $\boldsymbol{\gamma}^{(l)} \in \mathbb{R}^{K_{out}}$, can naturally satisfy the sparsity objective since it counts the number of non zero entries in a vector. However since the L_0 norm is a nondifferentiable function, we use the L_1 norm as a surrogate [142, 22, 19] which is amenable to backpropagation through its subgradient.

To realize the per filter multiplicand $\boldsymbol{\gamma}^{(l)}_i$, we leverage the per filter multiplier within the batch normalization layer [19]. In most modern networks, a batchnorm layer immediately follows the convolutional/linear layers and implements the following operation.

$$\boldsymbol{B}^{(l)}_i = \left(\frac{\widetilde{A}^{(l)} - \boldsymbol{\mu}^{(l)}_i}{\boldsymbol{\sigma}^{(l)}_i} \right) \boldsymbol{\gamma}^{(l)}_i + \boldsymbol{\beta}^{(l)}_i \tag{5.5}$$

Here $\widetilde{A}^{(l)}_i$ denotes output activation of filter i in layer l while $\widetilde{B}^{(l)}_i$ denotes its transformation through batchnorm layer l; $\boldsymbol{\mu}^{(l)} \in R^{K_{out}}$, $\boldsymbol{\sigma}^{(l)} \in R^{K_{out}}$ denote the mini-batch mean and standard deviation for layer l's activations; and $\boldsymbol{\gamma}^{(l)} \in R^{K_{out}}$ and $\boldsymbol{\beta}^{(l)} \in R^{K_{out}}$ are learnable parameters. Our sparsity regularization is defined on $\boldsymbol{\gamma}^{(l)}$ as below, where λ_g is a hyperparameter controlling the strength of sparsity regularization.

$$L_{Sparsity} = \lambda_g \sum_{i=1}^{N} \|\boldsymbol{\gamma}^{(l)}\|_1 \tag{5.6}$$

The sparsity regularization term (5.6) promotes learning a subset of important filters while training the model. Compression then amounts to globally pruning filters with the

smallest value of multiplicands in (5.5) to achieve the desired model compression. Pruning typically causes a large drop in accuracy. Once the pruned model is identified, we fine-tune it via retraining.

Now that we have discussed each component of REST, we present the full loss function in (5.7) and the training process in Algorithm 1. A pictorial overview of the process can be seen in Figure 5.2.

$$L_R = \underbrace{L(f(\widetilde{X}_p; \widetilde{W}), y)}_{\text{adversarial training}} + \underbrace{\lambda_o \sum_{i=1}^{N} \|(\widetilde{W}^{(i)})^T \widetilde{W}^{(i)} - \widetilde{I}\|_2}_{\text{spectral regularization}} + \underbrace{\lambda_g \sum_{i=1}^{N} \|\boldsymbol{\gamma}^{(l)}\|_1}_{\text{sparsity regularization}} \tag{5.7}$$

5.4 Experiments

We compare the efficacy of REST neural networks to four baseline models (Section 5.4.2) on two publicly available EEG datasets—Sleep-EDF from Physionet [143] and Sleep Heart Health Study (SHHS) [144]. Our evaluation focuses on two broad directions—**noise robustness** and **model efficiency**. Noise robustness compares the efficacy of each model when EEG data is corrupted with three types of noise: *adversarial*, *Gaussian* and *shot*. Model efficiency compares both static (e.g., model size, floating point operations) and dynamic measurements (e.g., inference time, energy consumption). For dynamic measurements which depend on device hardware, we deploy each model to a Pixel 2 smartphone.

5.4.1 Datasets

Our evaluation uses two real-world sleep staging EEG datasets.

- **Sleep-EDF**: This dataset consists of data from two studies—age effect in healthy subjects (SC) and Temazepam effects on sleep (ST). Following [123], we use whole-night

Table 5.1: Dataset summary outlining the number of 30 second EEG recordings belonging to each sleep stage class.

Dataset	**W**	**N1**	**N2**	**N3**(N4)	**REM**	**Total**
Sleep-EDF	8,168	2,804	17,799	5,703	7,717	42,191
SHHS	28,854	3,377	41,246	13,409	13,179	100,065

polysomnographic sleep recordings on 40 healthy subjects (one night per patient) from SC. It is important to note that the SC study is conducted in the subject's homes, not a sleep center and hence this dataset is inherently noisy. However, the sensing environment is still relatively controlled since sleep doctors visited the patient's home to setup the wearable EEG sensors. After obtaining the data, the recordings are manually classified into one of eight classes (W, N1, N2, N3, N4, REM, MOVEMENT, UNKNOWN); we follow the steps in [123] and merge stages N3 and N4 into a single N3 stage and exclude MOVEMENT and UNKNOWN stages to match the five stages of sleep according to the American Academy of Sleep Medicine (AASM) [145]. Each single channel EEG recording of 30 seconds corresponds to a vector of dimension 1×3000. Similar to [7], while scoring at time i, we include EEG recordings from times $i-3, i-2, i-1, i$. Thus we expand the EEG vector by concatenating the previous three time steps to create a vector of size 1×12000. After pre-processing the data, our dataset consists of 42,191 EEG recordings, each described by a 12,000 length vector and assigned a sleep stage label from Wake, N1, N2, N3 and REM using the Fpz-Cz EEG sensor (see Table 5.1 for sleep stage breakdown). Following standard practice [123], we divide the dataset on a *per-patient, whole-night* basis, using 80% for training, 10% for validation, and 10% for testing. That is, a single patient is recorded for one night and can only be in one of the three sets (training, validation, testing). The final number of EEG recordings in their respective splits are 34,820, 5,345 and 3,908. While the number of recordings appear to differ from the 80-10-10 ratio, this is because the data is split over the total number of *patients*, where each patient is monitored for a time period of variable length (9 hours $\pm$ few minutes.)

- **Sleep Heart Health Study (SHHS)**: The Sleep Heart Health Study consists of two rounds of polysomnographic recordings (SHHS-1 and SHHS-2) sampled at 125 Hz in a sleep center environment. Following [7], we use only the first round (SHHS-1) containing 5,793 polysomnographic records over two channels (C4-A1 and C3-A2). Recordings are manually classified into one of six classes (W, N1, N2, N3, N4 and REM). As suggested in [145], we merge N3 and N4 stages into a single N3 stage (see Table 5.1 for sleep stage breakdown). We use 100 distinct patients randomly sampled from the original dataset (one night per patient). Similar to [7], we look at three previous time steps in order to score the EEG recording at the current time step. This amounts to concatenating the current EEG recording of size 1×3750 (equal to 125 Hz $\times$ 30 Hz) to generate an EEG recording of size 1×15000. After this pre-processing, our dataset consists of 100,065 EEG recordings, each described by a 15,000 length vector and assigned a sleep stage label from the same 5 classes using the Fpz-Cz EEG sensor. We use the same 80-

10-10 data split as in Sleep-EDF, resulting in 79,940 EEG recordings for training, 9,999 for validation, and 10,126 for testing.

5.4.2 Model Architecture and Configurations

We use the sleep staging CNN architecture proposed by [7], since it achieves state-of-the-art accuracy for sleep stage classification using single channel EEG. We implement all models in PyTorch 0.4. For training and evaluation, we use a server equipped with an Intel Xeon E5-2690 CPU, 250GB RAM and 8 Nvidia Titan Xp GPUs. Mobile device measurements use a Pixel 2 smartphone with an Android application running Tensorflow Lite[1]. With [7] as the architecture for all baselines below, we compare the following 6 configurations:

1. **Sors** [7]: Baseline neural network model trained on unperturbed data. This model contains 12 1-D convolutional layers followed by 2 fully connected layers and achieves state-of-the-art performance on sleep staging using single channel EEG.

2. **Liu** [19]: We train on unperturbed data and compress the Sors model using sparsity regularization as proposed in [19].

3. **Blanco** [146]: We use same setup from Liu above. During test time, the noisy test input is filtered using a bandpass filter with cutoff 0.5Hz-40Hz This technique is commonly used for removing noise in EEG analysis [146].

4. **Ford** [40]: We train and compress the Sors model with sparsity regularization on input data perturbed by Gaussian noise. Gaussian training parameter c_g = 0.2 controls the perturbation strength during training; identified through a line search in Section 5.4.4.

5. **REST (A)**: Our compressed Sors model obtained through adversarial training and sparsity regularization. We use the hyperparameters: ϵ = 10, n_{iter}= 5/10 (SHHS/Sleep-EDF), where ϵ is a *key* variable controlling the strength of adversarial perturbation during training. The optimal ϵ value is determined through a line search described in Section 5.4.4.

6. **REST (A+S)**: Our compressed Sors model obtained through adversarial training, spectral and sparsity regularization. We set the spectral regularization parameter $\lambda_o = 3 \times 10^{-3}$ and sparsity regularization parameter $\lambda_g = 10^{-5}$ based on a grid search in Section 5.4.4.

[1]TensorFlow Lite: `https://www.tensorflow.org/lite`

All models are trained for 30 epochs using SGD. The initial learning rate is set to 0.1 and multiplied by 0.1 at epochs 10 and 20; the weight decay is set to 0.0002. All compressed models use the same compression method, consisting of weight pruning followed by model re-training. The sparsity regularization parameter $\lambda_g = 10^{-5}$ is identified through a grid search with λ_o (after determining ϵ through a line search). Detailed analysis of the hyperparameter selection for ϵ, λ_o and λ_g can be found in Section 5.4.4. Finally, we set a high sparsity level s = 0.8 (80% neurons from the original networks were pruned) after observation that the models are overparametrized for the task of sleep stage classification.

5.4.3 Evaluation Metrics

Noise robustness metrics To study the noise robustness of each model configuration, we evaluate macro-F1 score in the presence of three types of noise: adversarial, Gaussian and shot. We select macro-F1 since it is a standard metric for evaluating classification performance in imbalanced datasets. Adversarial noise is defined at three strength levels through $\epsilon = 2/6/12$ in Equation 5.2; Gaussian noise at three levels through $c_g = 0.1/0.2/0.3$ in Equation 5.8; and shot noise at three levels through $c_s = 5000/2500/1000$ in Equation 5.9. These parameter values are chosen based on prior work [138, 41] and empirical observation. For evaluating robustness to adversarial noise, we assume the white box setting where the attacker has access to model weights. The formulation for Gaussian and shot noise is in Equation 5.8 and 5.9, respectively.

$$\widetilde{X}_{gauss} = \widetilde{X} + N(0, c_g \cdot \sigma_{train}) \tag{5.8}$$

In Equation 5.8, σ_{train} is the standard deviation of the training data and N is the normal distribution. The noise strength—low, medium and high—corresponds to $c_g = 0.1/0.2/0.3$.

$$\begin{aligned} \widetilde{X}_{norm} &= \frac{\widetilde{X} - x_{min}}{x_{max} - x_{min}} \\ \widetilde{X}' &= clip_{0,1}\left(\frac{Poisson(\widetilde{X}_{norm}.c_s)}{c_s}\right) \\ \widetilde{X}_{shot} &= \widetilde{X}'.(x_{max} - x_{min}) + x_{min} \end{aligned} \tag{5.9}$$

In Equation 5.9, x_{min}, x_{max} denote the minimum and maximum values in the training data; and $clip_{0,1}$ is a function that projects the input to the range [0,1].

Model efficiency metrics To evaluate the efficiency of each model configuration, we use the following measures:

- **Parameter Reduction**: Memory consumed (in KB) for storing the weights of a model.
- **Floating point operations (FLOPS)**: Number of multiply and add operations performed by the model in one forward pass. Measurement units are Mega (10^6).
- **Inference Time**: Average time taken (in seconds) to score one night of EEG data. We assume a night consists of 9 hours and amounts to 1,080 EEG recordings (each of 30 seconds). This is measured on a Pixel 2 smartphone.
- **Energy Consumption**: Average energy consumed by a model (in Joules) to score one night of EEG data on a Pixel 2 smartphone. To measure consumed energy, we implement an infinite inference loop over EEG recordings until the battery level drops from 100% down to 85%. For each unit percent drop (i.e., 15 levels), we log the number of iterations N_i performed by the model. Given that a standard Pixel 2 battery can deliver 2700 mAh at 3.85 Volts, we use the following conversion to estimate energy consumed E (in Joules) for a unit percent drop in battery level $E = \frac{2700}{1000} \times 3600 \times 3.85$. The total energy for inferencing over an entire night of EEG recordings is then calculated as $\frac{E}{N_i} \times 1080$ where N_i is the number of inferences made in the unit battery drop interval. We average this for every unit battery percentage drop from 100% to 85% (i.e., 15 intervals) to calculate the average energy consumption

5.4.4 Hyperparameter Selection

Optimal hyper-parameter selection is crucial for obtaining good performance with both baseline and REST models. We systematically conduct a series of line and grid searches to determine ideal values of ϵ, c_g, λ_o and λ_g using the validation sets.

Selecting ϵ This parameter controls the perturbation strength of adversarial training in Equation 5.2. Correctly setting this parameter is critical since a small ϵ value will have no effect on noise robustness, while too high a value will lead to poor benign accuracy. We follow standard procedure and determine the optimal ϵ on a per-dataset basis [138], conducting a line search across $\epsilon \in [0,30]$ in steps of 2. For each value of ϵ we measure benign and adversarial validation macro-F1 score, where adversarial macro-F1 is an average of three strength levels: low (ϵ=2), medium (ϵ=6) and high (ϵ=12). We then select the ϵ with highest macro-F1 score averaged across the benign and adversarial macro-F1. Line search results are shown in Figure 5.3; we select $\epsilon = 10$ for both dataset since it's the value with highest average macro-F1.

Selecting c_g This parameter controls the noise perturbation strength of Gaussian training in Equation 5.8. Similar to ϵ, we determine c_g on a per-dataset basis, conducting a line

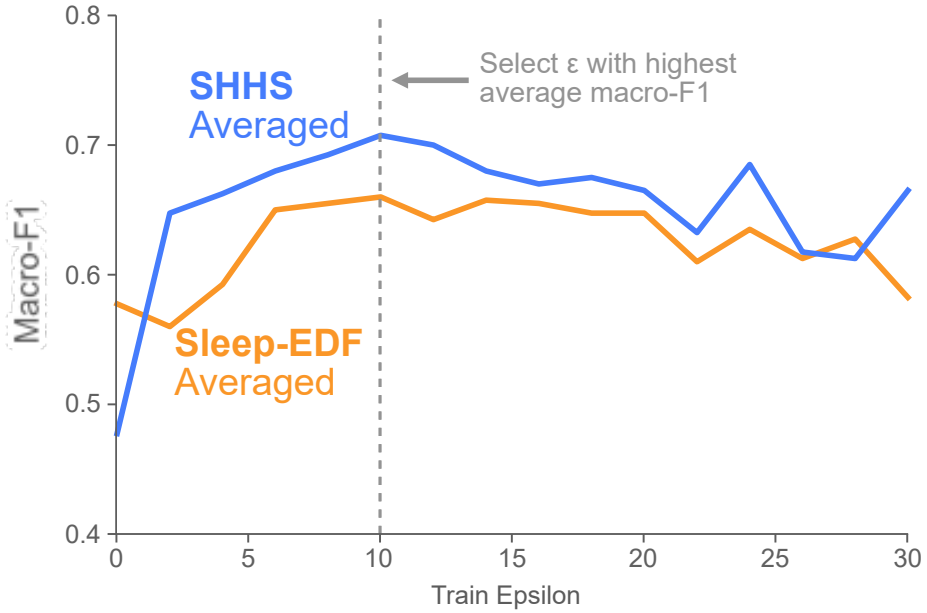


Figure 5.3: Line search results for ϵ on Sleep-EDF and SHHS datasets. We select ϵ=10, since it provides the best average macro-F1 score on both datasets.

Table 5.2: Line search results for identifying optimal c_g on Sleep-EDF and SHHS datasets. Macro-F1 is abbreviated F1 in table; average macro-F1 is the mean of all macro-F1 scores. We select c_g=0.2 for both datasets as it represents a good trade-off between benign and Gaussian macro-F1.

			Guassian F1			
	c_g	**Benign F1**	**Low**	**Med**	**High**	**Average F1**
EDF	0.1	0.75	0.76	0.7	0.5	0.68
	0.2	0.7	0.72	0.75	0.64	**0.70**
	0.3	0.67	0.68	0.71	0.75	0.7025
SHHS	0.1	0.69	0.74	0.45	0.21	0.52
	0.2	0.68	0.69	0.68	0.43	**0.62**
	0.3	0.55	0.57	0.65	0.74	0.63

search across c_g values: 0.1 (low), 0.2 (medium) and 0.3 (high). Based on results from Table 5.2, we select c_g=0.2 for both datasets since it provides the best average macro-F1 score while minimizing the drop in benign accuracy.

Selecting λ_o and λ_g These parameters determine the strength of spectral and sparsity regularization in Equation 5.7. We determine the best value for λ_o and λ_g through a grid search across the following parameter values $\lambda_o = [0.001, 0.003, 0.005]$ and $\lambda_g = [1E-04, 1E-05]$. Based on results from Table 5.3, we select $\lambda_o = 0.003$ and $\lambda_g = 1E-05$. Since these are model dependent parameters, we calculate them once on the Sleep-EDF dataset and re-use them for SHHS.

Table 5.3: Grid search results for λ_o and λ_g on Sleep-EDF dataset. Macro-F1 is abbreviated as F1 in table; average macro-F1 is the mean of all macro-F1 scores. We select λ_o and λ_g with highest average macro-F1 score.

			Adversarial F1			
λ_o	λ_g	**Benign F1**	**Low**	**Med**	**High**	**Avg. F1**
0.001	1E-04	0.73	0.66	0.65	0.61	0.66
0.003	1E-04	0.72	0.64	0.63	0.59	0.65
0.005	1E-04	0.72	0.65	0.64	0.62	0.66
0.001	1E-05	0.73	0.66	0.65	0.62	0.67
0.003	1E-05	0.73	0.67	0.66	0.62	**0.67**
0.005	1E-05	0.73	0.64	0.64	0.62	0.66

Table 5.4: Meta Analysis: Comparison of macro-F1 scores achieved by each model. The models are evaluated on Sleep-EDF and SHHS datasets with three types and strengths of noise corruption. We bold the compressed model with the best performance (averaged over 3 runs) and report the standard deviation of each model next to the macro-F1 score. REST performs better in *all* noise test measurements.

				Adversarial			Gaussian			Shot		
Data	**Method**	**Compress**	**No noise**	Low	Med	High	Low	Med	High	Low	Med	High
Sleep-EDF	Sors [7]	✗	0.67 ± 0.02	0.57 ± 0.02	0.51 ± 0.04	0.19 ± 0.06	0.66 ± 0.03	0.60 ± 0.03	0.39 ± 0.08	0.58 ± 0.04	0.42 ± 0.08	0.11 ± 0.03
	Liu [19]	✓	**0.69** ± 0.02	0.52 ± 0.07	0.41 ± 0.07	0.09 ± 0.02	0.67 ± 0.02	0.53 ± 0.02	0.28 ± 0.04	0.52 ± 0.03	0.31 ± 0.04	0.06 ± 0.01
	Blanco [146]	✓	0.68 ± 0.01	0.51 ± 0.06	0.40 ± 0.06	0.09 ± 0.02	0.65 ± 0.02	0.54 ± 0.04	0.31 ± 0.10	0.53 ± 0.04	0.34 ± 0.09	0.08 ± 0.02
	Ford [40]	✓	0.64 ± 0.01	0.59 ± 0.01	0.60 ± 0.02	0.31 ± 0.08	0.65 ± 0.01	0.67 ± 0.02	0.57 ± 0.03	0.67 ± 0.02	0.60 ± 0.02	0.10 ± 0.01
	REST (A)	✓	0.66 ± 0.02	0.64 ± 0.02	0.64 ± 0.02	0.61 ± 0.02	0.66 ± 0.02	0.67 ± 0.01	0.66 ± 0.01	0.67 ± 0.01	0.66 ± 0.01	**0.42** ± 0.06
	REST (A+S)	✓	**0.69** ± 0.01	**0.67** ± 0.02	**0.66** ± 0.01	**0.61** ± 0.03	**0.69** ± 0.01	**0.68** ± 0.01	**0.67** ± 0.02	**0.68** ± 0.01	**0.67** ± 0.02	**0.42** ± 0.08
SHHS	Sors [7]	✗	**0.78** ± 0.01	0.62 ± 0.03	0.46 ± 0.03	0.33 ± 0.00	0.64 ± 0.03	0.43 ± 0.02	0.35 ± 0.04	0.69 ± 0.02	0.59 ± 0.03	0.45 ± 0.01
	Liu [19]	✓	0.77 ± 0.01	0.61 ± 0.02	0.49 ± 0.04	0.34 ± 0.03	0.66 ± 0.05	0.45 ± 0.05	0.34 ± 0.04	0.70 ± 0.04	0.62 ± 0.04	0.47 ± 0.05
	Blanco [146]	✓	0.77 ± 0.01	0.60 ± 0.03	0.47 ± 0.04	0.33 ± 0.02	0.64 ± 0.07	0.43 ± 0.05	0.34 ± 0.04	0.67 ± 0.06	0.59 ± 0.05	0.46 ± 0.04
	Ford [40]	✓	0.62 ± 0.02	0.59 ± 0.01	0.62 ± 0.00	0.59 ± 0.05	0.66 ± 0.00	0.75 ± 0.04	0.47 ± 0.10	0.65 ± 0.00	0.68 ± 0.01	0.74 ± 0.04
	REST (A)	✓	0.70 ± 0.01	0.68 ± 0.00	0.70 ± 0.01	0.67 ± 0.01	0.72 ± 0.01	0.76 ± 0.01	0.58 ± 0.03	0.72 ± 0.01	0.74 ± 0.01	0.76 ± 0.01
	REST (A+S)	✓	0.72 ± 0.01	**0.69** ± 0.01	**0.70** ± 0.01	**0.69** ± 0.02	**0.74** ± 0.01	**0.77** ± 0.01	**0.62** ± 0.03	**0.73** ± 0.01	**0.75** ± 0.01	**0.78** ± 0.00

5.4.5 Noise Robustness

To evaluate noise robustness, we ask the following questions—(1) what is the impact of REST on model accuracy with and without noise in the data? and (2) how does REST training compare to baseline methods of benign training, Gaussian training and noise filtering? In answering these questions, we analyze noise robustness of models at three scales: (i) meta-level macro-F1 scores; (ii) meso-level confusion matrix heatmaps; and (iii) granular-level single-patient hypnograms.

I. Meta analysis: Macro-F1 Scores In Table 5.4, we present a high-level overview of model performance through macro-F1 scores on three types and strength levels of noise corruption. The Macro-F1 scores and standard deviation are reported by averaging over

three runs for each model and noise level. We identify multiple key insights as described below:

1. **REST Outperforms Across All Types of Noise** As demonstrated by the higher macro-F1 scores, REST outperforms all baseline methods in the presence of noise. In addition, REST has a low standard deviation, indicating model performance is not dependent on weight initialization.
2. **Spectral Regularization Improves Performance** REST $(A+S)$ consistently improves upon REST (A), indicating the usefulness of spectral regularization towards enhancing noise robustness by constraining the Lipschitz constant.
3. **SHHS Performance Better Than Sleep-EDF** Performance is generally better on the SHHS dataset compared to Sleep-EDF. One possible explanation is due to the SHHS dataset being less noisy in comparison to the Sleep-EDF dataset. This stems from the fact that the SHHS study was performed in the hospital setting while Sleep-EDF was undertaken in the home setting.
4. **Benign & Adversarial Accuracy Trade-off** Contrary to the traditional trade-off between benign and adversarial accuracy, REST performance matches Liu in the no noise setting on sleep-EDF. This is likely attributable to the noise in the Sleep-EDF dataset, which was collected in the home setting. On the SHHS dataset, the Liu model outperforms REST in the no noise setting, where data is captured in the less noise prone hospital setting. Due to this, REST models are best positioned for use in noisy environments (e.g., at home); while traditional models are more effective in controlled environments (e.g., sleep labs).

II. Meso Analysis: Per-class Performance We visualize and identify class-wise trends using confusion matrix heatmaps (Fig. 5.4). Each confusion matrix describes a model's performance for a given level of noise (or no noise). A model that is performing well should have a dark diagonal and light off-diagonal. We normalize the rows of each confusion matrix to accurately represent class predictions in an imbalanced dataset. When a matrix diagonal has a value of 1 (dark blue, or dark green) the model predicts every example correctly; the opposite occurs at 0 (white). Analyzing Figure 5.4, we identify the following key insights:

1. **REST Performs Well Across All Classes** REST accurately predicts each sleep stage (W, N1, N2, N3, REM) across multiple types of noise (Fig. 5.4, bottom 3 rows), as evidenced by the dark diagonal. In comparison, each baseline method has considerable performance degradation (light diagonal) in the presence of noise. This is particularly

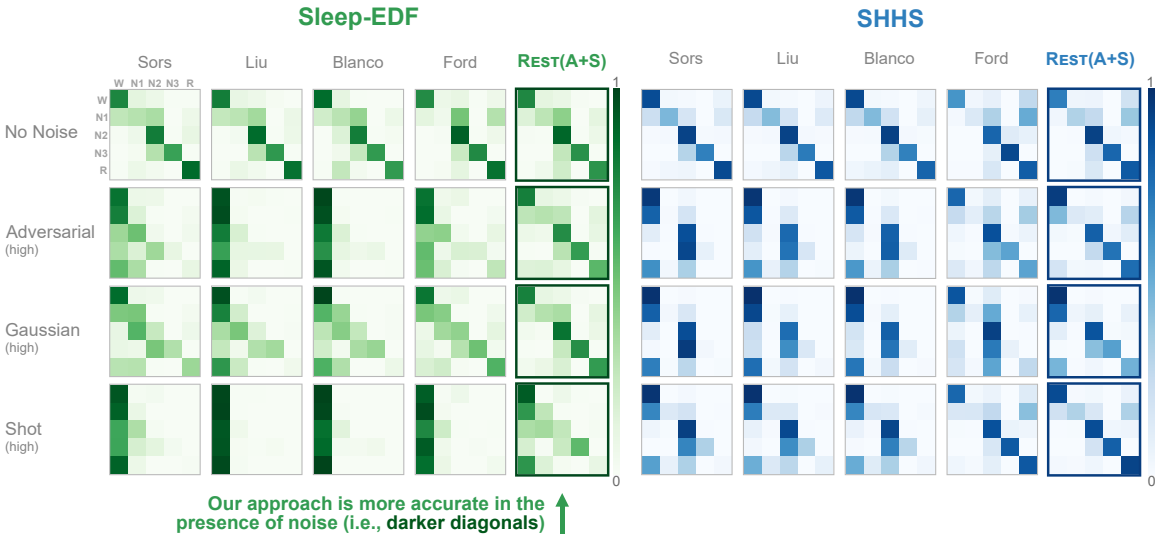


Figure 5.4: Meso Analysis: Class-wise comparison of model predictions. The models are evaluated over the SHHS test set perturbed with different noise types. In each confusion matrix, rows are ground-truth classes while columns are predicted classes. The intensity of a cell is obtained by normalizing the score with respect to the class membership. When a cell has a value of 1 (dark blue, or dark green) the model predicts every example correctly, the opposite occurs at 0 (white). A model that is performing well would have a dark diagonal and light off-diagonal. REST has the darkest cells along the diagonal on both datasets.

evident on the Sleep-EDF dataset (left half) where data is collected in the noisier home environment.

2. **N1 Class Difficult to Predict** When no noise is present (Fig. 5.4, top row), each method performs well as evidenced by the dark diagonal, except on the N1 sleep stage class. This performance drop is likely due to the limited number of N1 examples in the datasets (see Table 5.1).
3. **Increased Misclassification Towards "Wake" Class** On the Sleep-EDF dataset, shot and adversarial noise cause the baseline models to mispredict classes as Wake. One possible explanation is that the models misinterpret the additive noise as evidence for the wake class which has characteristically large fluctuations.

III. Granular Analysis: Single-patient Hypnograms We want to more deeply understand how our REST models counteract noise at the hypnogram level. Therefore, we select a test set patient from the SHHS dataset, and generate and visualize the patient's overnight hypnograms using the Sors and REST models on three levels of Gaussian noise corruption (Figure 5.5). Each of these hypnograms is compared to a trained technicians hypnogram (expert scored in Fig. 5.5), representing the ground-truth. We inspect a few more test set patients using the above approach, and identify multiple key representative insights:

1. **Noisy Environments Require Robust Models** As data noise increases, Sors performance degrades. This begins at the low noise level, further accelerates in the medium level and reaches nearly zero at the high level. In contrast, REST effectively handles all levels of noise, generating an accurate hypnogram at even the highest level.
2. **Low Noise Environments Give Good Performance** In the no noise setting (top row) both the Sors and REST models generate accurate hypnograms, closely matching the contours of expert scoring (bottom).

5.4.6 Model Efficiency

We measure model efficiency along two dimensions—(1) *static metrics*: amount of memory required to store weights in memory and FLOPS; and (2) *dynamic metrics*: inference time and energy consumption. For dynamic measurements that depend on device hardware, we deploy each model to a Pixel 2 smartphone.

Analyzing Static Metrics: Memory & Flops Table 5.5 describes the size (in KB) and computational requirements (in MFlops) of each model. We identify the following key insights:

1. **REST Models Require Fewest FLOPS** On both datasets, REST requires the least number of FLOPS.
2. **REST Models are Small** REST models are also smaller (or comparable) to baseline

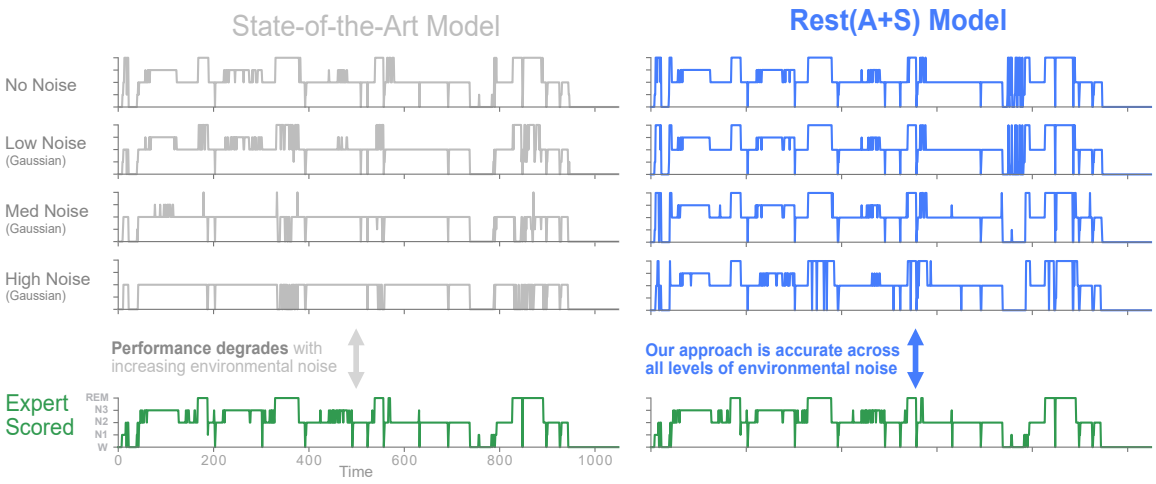


Figure 5.5: Granular Analysis: Comparison of the overnight hypnograms obtained for a patient in the SHHS test set. The hypnograms are generated using the Sors (left) and REST (right) models in the presence of increasing strengths of Gaussian noise. When no noise is present (top row), both models perform well, closely matching the ground truth (bottom row). However, with increasing noise, Sors performance rapidly degrades, while REST continues to generate accurate hypnograms.

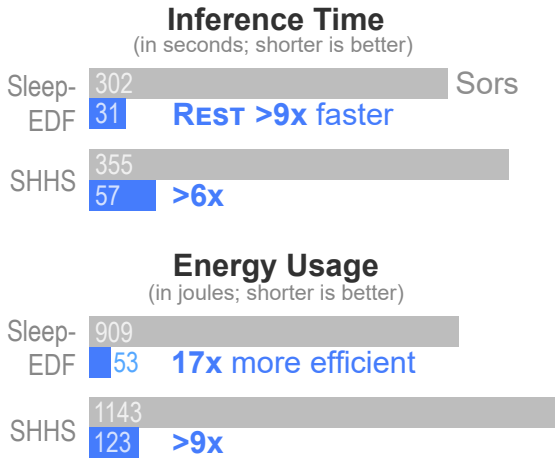


Figure 5.6: Time and energy consumption for scoring a single night of EEG recordings. REST(A+S) is significantly faster and more energy efficient than the state-of-the-art Sors model. Evaluations were done on a Pixel 2 smartphone.

Table 5.5: Comparison on model size and the FLOPS required to score a single night of EEG recordings. REST models are significantly smaller and comparable in size/compute to baselines.

Data	Model	**Size** (KB)	**MFlops**
Sleep-EDF	Sors [7]	8,896	1451
	Liu [19]	**440**	127
	Blanco [146]	440	127
	Ford [40]	448	144
	REST (A)	464	98
	REST (A+S)	449	**94**
SHHS	Sors [7]	8,996	1815
	Liu [19]	**464**	211
	Blanco [146]	464	211
	Ford [40]	478	170
	REST (A)	476	160
	REST (A+S)	496	**142**

compressed models while achieving significantly better noise robustness.

3. **Model Efficiency and Noise Robustness** Combining the insights from Section 5.4.5 and the above, we observe that REST models have significantly better noise robustness while maintaining a competitive memory footprint. This suggests that robustness is more dependent on the the training process, rather than model capacity.

Analyzing Dynamic Metrics: Inference Time & Energy In Figure 5.6, we benchmark the inference time and energy consumption of a Sors and REST model deployed on a Pixel 2 smartphone using Tensorflow Lite. We identify the following insights:

1. **REST Models Run Faster** When deployed, REST runs $9\times$ and $6\times$ faster than the uncompressed model on the two datasets.
2. **REST Models are Energy Efficient** REST models also consume $17\times$ and $9\times$ less energy than an uncompressed model on the Sleep-EDF and SHHS datasets, respectively.
3. **Enabling Sleep Staging for Edge Computing** The above benefits demonstrate that model compression effectively translates into faster inference and a reduction in energy consumption. These benefits are crucial for deploying on the edge.

5.5 Conclusion

We identified two key challenges in developing deep neural networks for sleep monitoring in the home environment—*robustness to noise* and *efficiency*. We proposed to solve these challenges through REST—a new method that simultaneously tackles both issues. For the sleep staging task over electroencephalogram (EEG), REST trains models that achieve up to $19\times$ parameter reduction and $15\times$ MFLOPS reduction with an increase of up to 0.36 in macro-F-1 score in the presence of noise. By deploying these models to a smartphone, we demonstrate that REST achieves up to $17\times$ energy reduction and $9\times$ faster inference.

CHAPTER 6
HAR:HARDNESS AWARE REWEIGHTING FOR IMBALANCED DATASETS

Class imbalance is a significant issue that causes neural networks to underfit to the rare classes. Traditional mitigation strategies include loss reshaping and data resampling which amount to increasing the loss contribution of minority classes and decreasing the loss contributed by the majority ones. However, by treating each example within a class equally, these methods lead to undesirable scenarios where hard-to-classify examples from the majority classes are down-weighted and easy-to-classify examples from the minority classes are up-weighted. We propose the *Hardness Aware Reweighting* (HAR) framework, which circumvents this issue by increasing the loss contribution of hard examples from both the majority and minority classes. This is achieved by augmenting a neural network with intermediate classifier branches to enable early-exiting during training. Experimental results on large-scale datasets demonstrate that HAR consistently improves state-of-the-art accuracy while saving up to 20% of inference FLOPS.

6.1 Introduction

Class imbalance is a ubiquitous phenomenon commonly observed in naturally occurring data distributions [147, 148, 1]. However, despite it's ubiquity, popular datasets (e.g., CIFAR-10/100 [77], ImageNet [149]) are often artificially balanced leading to a mismatch between reality and practice. Realizing this mismatch, recent research aims to account for class imbalance by evolving the traditional datasets to their *long-tailed* (LT) versions, e.g., CIFAR LT [43], ImageNet LT [59]. Here, a long tail signifies that the majority of the classes constitute only a minority of the overall data and vice versa. Our work focuses on training accurate neural networks on datasets exhibiting a long-tail class imbalance.

Overcoming a long tail imbalance is hard, since most classifiers tend to favor the majority classes that constitute the bulk of the training set [147, 150]. This is especially true for convolutional neural networks (CNNs), which are known to suffer under class imbalance [57]. An effective mitigating strategy is loss reweighting, which follows the simple rule: *increase the loss contributed by the minority classes, and decrease the loss contributed by the majority ones*. This idea is realized in popular reweighting methods, such as class reweighting based on inverse frequency [151, 152], inverse-square-root frequency [153] and the effective number of samples [43]. These methods, however, uniformly reweight all examples within each class, leading to the undesirable scenario

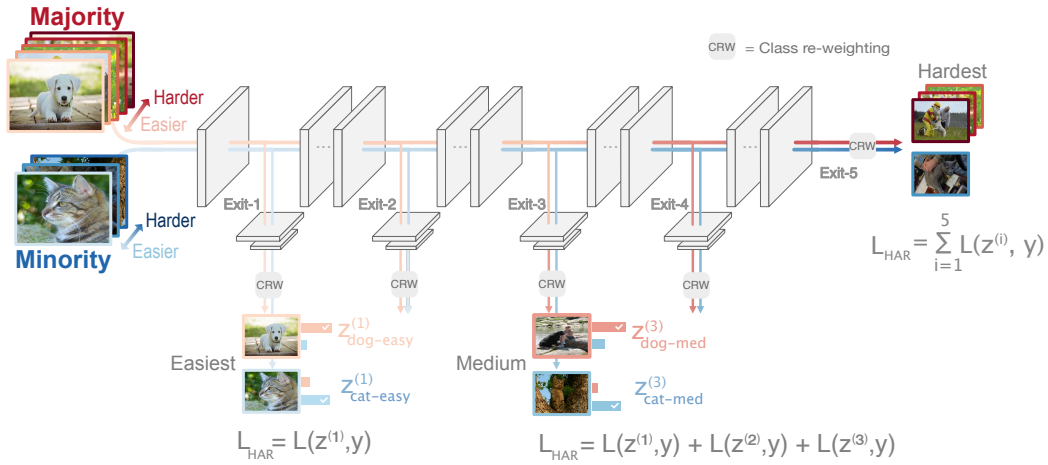


Figure 6.1: **H**ardness **A**ware **R**eweighting (**HAR**) framework augments a backbone network with auxiliary classifier branches. During training, an example accumulates loss at each branch, until either (a) it is *confidently* and *correctly* classified at a branch, or (b) it reaches the end. A harder example exits later in the network and accumulate higher overall loss.

wherein hard-to-classify examples from the majority classes are downweighted, and easy-to-classify examples from the minority classes are upweighted. This is undesirable because hard examples are known to provide stronger learning signals [154, 155] and should avoid being downweighted.

One way to avoid the above scenario is to develop a more fine-grained, instance-specific reweighting strategy. This is the motivation behind recent methods that use meta-learning [46, 47] and domain-adaptation [48]. However, these methods do not encode the key notion of example hardness, which means that hard examples are still susceptible to being downweighted, causing their learning signals to be diminished [154, 155]. On the other end, classical hardness-aware methods such as ADASYN [51] and SMOTEBoost [156] do not scale to modern CNNs [57]. This presents a need for developing a *hardness-aware*, instance-specific reweighting strategy suitable for training deep neural networks.

We propose the **H**ardness **A**ware **R**eweighting (**HAR**) framework that incorporates the notion of example hardness during training. HAR is premised around the idea of increasing the loss contribution of hard examples in both the majority and minority classes. It augments a backbone neural network with auxiliary classifier branches (illustrated in Fig. 6.1) which enable the backbone to learn a notion of hardness by conditionally exiting easy-to-classify examples during training. Once an example exits, it no longer incurs additional loss. A natural outcome of this process is that harder examples exit towards the end and accumulate a higher overall loss, thus realizing HAR's goal.

6.1.1 Our Contributions

1. Example Hardness as Key to Unlock Generalization. We contribute the key idea that accounting for *example hardness* during class reweighting can improve accuracy of modern CNNs trained under long-tail class imbalance. (Sec. 6.3.1)

2. HAR: General Framework to Endow Hardness Awareness. Our proposed HAR framework is a general approach to endow any loss function with hardness awareness and offers three benefits:

1. **State-of-the-art Accuracy**. By increasing the loss contribution of hard examples, HAR shifts the focus of learning to harder examples, leading to state-of-the-art accuracies on standard, large-scale imbalanced benchmark datasets, such as ImageNet LT [59] and iNaturalist '18 [157]. (Sec. 6.4.4)

2. **Compute Savings**. HAR enables *dynamic* inference wherein the inference cost of a model can be varied in realtime. The blue dots (•) in the Fig. 6.2 plots a HAR model's accuracies at eight different compute budgets. The curve formed envelopes all other approaches, showing that HAR enables inference FLOPS savings while still achieving state-of-the-art accuracy. (Sec. 6.3.3)

3. **Plug-and-play Support for Existing Loss functions.** HAR can endow any loss functions with hardness awareness by using it at the auxiliary branches. We observe large accuracy improvements when using the weighted cross-entropy [43] or the weighted LDAM loss [58] at the exits. (Sec. 6.3.2)

3. Extensive Evaluation on Large-scale Datasets. We comprehensively evaluate HAR using modern CNNs by training them on four standard imbalanced benchmarks, including the large-scale ImageNet LT [59] and iNaturalist '18 [157] datasets. Additionally, we ablate on the different modelling choices of HAR to uncover why and how it leads to a higher accuracy with compute savings. (Sec. 6.4.6)

Conceptually, HAR shares its motivation with the seminal work of focal loss [45] which is the first to demonstrate the benefit of enabling hardness awareness in the class imbalance setting. There are however two major differences: (1) focal loss uses prediction confidence as a measure of hardness, while HAR uses the idea of early exiting. This couples focal loss to cross entropy, whereas HAR can work with many loss types; (2) unlike HAR, focal loss cannot save compute on easy examples, as early-exiting is not an option. Our experiments in Sec. 6.4 show that a model trained with HAR significantly outperforms a model trained with focal loss.

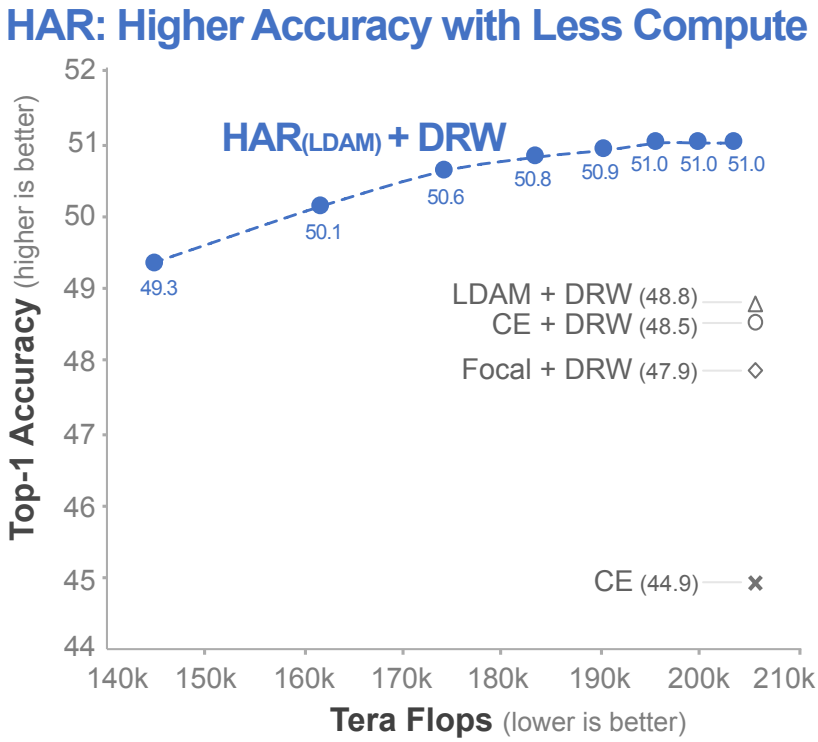


Figure 6.2: The HAR framework leads to higher top-1 accuracies while saving compute, for a ResNet-50 model trained on the ImageNet LT dataset. Additionally, HAR supports *dynamic* inference that offers a favorable top-1 accuracy *vs*. efficiency trade-off under different compute budgets (shown as blue dots •). In contrast, traditional methods lead to *static* models with *fixed* compute costs during inference.

6.2 Related Research

We summarize related works from three relevant areas: class imbalance, hardness aware learning, and multi-branch neural networks.

6.2.1 Overcoming class imbalance

The techniques for overcoming class imbalance techniques fall into the following three categories.

Loss rebalancing: These methods reweight the loss contribution of each example such that the loss for minority classes is upweighted, while that of the majority classes is down-weighted. The weighting scheme itself can be uniform across all the examples within a class [43, 44] or can be more fine-grained *i.e*., specific to each example in consideration [45, 46, 47, 48]. The uniform reweighting techniques include reweighting based on inverse class frequency [43, 44] or based on the effective number of samples in each class [43]. On the other hand, fine-grained approaches include Focal loss [45], which reweights based

on sample hardness or recent studies [46, 47, 48] that employ meta-learning to perform sample reweighting.

Data resampling: These methods either repeatedly sample examples from the minority class (over-sampling) [49, 50, 51, 52] or discard samples from the majority class (under-sampling) [53, 54, 55, 56]. Popular strategies include SMOTE that over-samples the minority class through linear interpolation [49]; or [53] that under-samples the majority class by clustering and replacing the majority class examples by a few anchor points. With neural networks, over-sampling generally creates redundancy and risks over-fitting to the rare classes, while, under-sampling is susceptible to losing information from the majority classes [57].

Training strategies: These methods modify the training procedure to mitigate the problem of class imbalance. For instance, LDAM [58], introduces a delayed reweighting scheme wherein, class reweighting is applied after a few epochs of training. Kang et al. [59] show improvement through a two-step training process which decouples representation and classifier learning. Recently, BBN [60] show that gradually shifting emphasis from class sampling to reverse sampling helps improve accuracy.

HAR is complementary to the above three directions in that it introduces a hardness aware learning framework that readily works with many existing loss reweighting and training strategies to further improve accuracy on imbalanced datasets.

6.2.2 Hardness aware learning methods

The concept of example hardness is central to a variety of successful learning techniques including those from curriculum learning [158, 159] (which deals with ordering the training set in increasing order of hardness), hard negative mining [160, 154, 45] (which deals with identifying false positive samples in order to improve classification performance) and reinforcement learning [161, 162] (which deals with ordering tasks from easy to hard in order to aid an agent's learning). The underlying assumption in these methods is that "harder examples" contribute a stronger learning signal and are more informative than their "easier" counterparts [154, 155]. To this end, example hardness can be defined in multiple ways. For example, OHEM [160] uses absolute loss as an indicator of hardness; focal loss [45] uses the prediction-confidence of true class probability as a measure of hardness; Hacohen et. al.[159] use the prediction-confidence of a pre-trained teacher model as the measure of hardness. Differing from these, HAR uses a more general hardness measure which embodies the intuition that easy examples are those which can be confidently and correctly predicted using coarse-features from earlier layers of a neural network. This

enables HAR to impart hardness awareness to many existing loss functions.

6.2.3 Multi-branch neural networks

Research in this area aims to endow a neural network with auxiliary classifier branches (or early-exits) that allow for obtaining predictions from intermediate locations along the backbone DNN. This leads to advantages such as, saving inference time compute [30, 31, 32]; mitigating the vanishing gradient problem as in Inception networks [33]; while also affording a natural application to computing paradigms such as fog computing and 5G [34, 35]. The important research challenges for multi-branch DNNs (see review paper [36]) stem from questions such as: where to place the early-exits, what criterion to use for early-exiting and how to define the training objective. Many works place the early-exits after each block of layers which enables large savings of inference FLOPS [30, 31, 32]. The exit criteria range from early-exiting based on low entropy predictions as in BranchyNet [37] to early exiting based on prediction confidence as in MSDNet [30]. Training objectives include ones that distill knowledge from later exits to earlier ones [31, 38] or ensemble predictions from multiple branches to improve adversarial robustness [32]. To the best of our knowledge, HAR is the first to use multi-branch DNNs to enable hardness-aware loss reweighting for long-tailed imbalance. Our experiments show this leads to large accuracy improvement in the presence of class imbalance.

6.3 Methodology

6.3.1 Why care about hardness?

We hypothesize that within both the majority and minority classes some examples are easier to classify than others. Consequently, not every example in the minority class needs to be equally upweighted; and not every example in the majority class needs to be equally downweighted. In order to verify this hypothesis, we train a ResNet-32 model on CIFAR-10 LT (using vanilla cross entropy) and determine: What proportion of the rarest class examples obtain a high confidence prediction (≥ 0.9) and what proportion of the majority class examples obtain a low confidence prediction ($\leq$0.1). Fig. 6.3 plots outcome of this experiment.

As expected, many examples from the minority class are classified with low confidence while many examples from the majority class are predicted with near certainty. However, confirming our hypothesis, a considerable proportion of the majority class examples obtain a low confidence prediction and vice versa. It is precisely this subset of examples—*low*

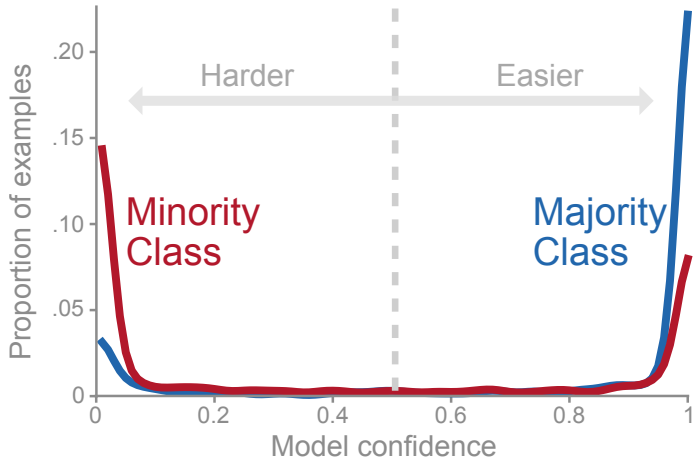


Figure 6.3: Observe that a considerable proportion of the majority class examples predicted by a ResNet-32 trained on CIFAR-10 LT using cross entropy, obtain a low confidence prediction and vice versa. It is precisely this subset of examples—*low confidence majority* and *high confidence minority*—that HAR impacts the most. Particularly, HAR increases the loss contribution of low confidence *majority* examples while retaining the original loss contribution for high confidence *minority*.

confidence majority and *high confidence minority*—that HAR impacts the most. In particular, HAR increases the loss contribution of low confidence *majority* examples while retaining the original loss contribution for high confidence *minority* examples thereby enabling a fine-grained, hardness aware approach to loss reweighting.

Problem Setup. Denote a multi-exit neural network by $f : \boldsymbol{X} \rightarrow \boldsymbol{z}$ that maps an input example $\boldsymbol{X} \in \mathbb{R}^{h \times w \times 3}$ to a list of prediction vectors $\boldsymbol{z} = [\boldsymbol{z}^{(1)}, ..., \boldsymbol{z}^{(k)}]$ where the vector $\boldsymbol{z}^{(k)} \in \mathbb{R}^c$ specifies the prediction confidence over c classes at the k^{th} exit. Assuming the weights $\boldsymbol{\theta}^{(k)}$ parameterize f up to exit k, then we have $\boldsymbol{z}^{(k)} = f(\boldsymbol{X}; \boldsymbol{\theta}^{(k)})$ as the output at the k^{th} exit. The supervised learning task specifies training f on a training dataset $\mathcal{D} = \{(\boldsymbol{X}_1, y_1), ..., (\boldsymbol{X}_n, y_n)\}$ containing c classes. If n_j denotes the number of training examples in class j, then a long tail dataset satisfies $n_i > n_j, \forall i < j$ and $n_1 >> n_c$. Typically, post training, f is evaluated on a balanced test set $\mathcal{D}'$ which satisfies $n'_i = n'_j, \forall i, j$. The goal of imbalanced classification is to maximize an evaluation metric such as top-1 accuracy on $\mathcal{D}'$ while learning a classifier on $\mathcal{D}$.

6.3.2 *Training a multi-branch DNN with* HAR

Preliminaries

Training algorithms for multi-branch neural networks come in many shapes and forms. The most popular ones [163, 164, 37, 30, 32] train all branches simultaneously by defining a

single training objective that incorporates the predictions from all intermediate branches. One way to accomplish this [163, 37, 30] is to define the aggregated loss as a weighted sum of the loss computed at each exit

$$\mathcal{L}_{multi-exit}(\boldsymbol{z}_i, y_i) = \sum_{k=1}^{K} \alpha_k \mathcal{L}(\boldsymbol{z}_i^{(k)}, y_i), \tag{6.1}$$

where $\boldsymbol{z}_i = [\boldsymbol{z}_i^{(1)}; ..., \boldsymbol{z}_i^{(K)}]$ is the list of predictions from K branches, $\alpha_k$are positive constants, and $\mathcal{L}$ is a classification loss such as the cross-entropy. Another strategy [164] is to first combine the prediction vectors at each branch

$$\hat{\boldsymbol{z}}_i = \sum_{k=1}^{K} \alpha_k \boldsymbol{z}_i^{(k)}, \tag{6.2}$$

and then obtain the training loss as $\mathcal{L}_{multi-exit}(\boldsymbol{z}_i, y_i) = \mathcal{L}(\hat{\boldsymbol{z}}, y_i)$. Among these two approaches, HAR's training objective is inspired from the former (*i.e.*, Eq. 6.1).

General HAR *loss*

The goal of the HAR training objective is to up weight the loss contribution of hard examples. To this end, we modify Eq. (6.1) in two ways. First, we simplify the hyperparameter design space by setting $\alpha_k = 1$ (thus weighing all branches equally). Second, we introduce conditional aggregation into the summation term on the right hand side with the general idea: to aggregate the loss up until the first exit where the neural network *correctly* and *confidently* predicts the class of an input example. This objective is defined as

$$\begin{aligned} \mathcal{L}_{\text{HAR}}(\boldsymbol{z}_i, y_i) &= \sum_{k \in [1,...,k_i]} \mathcal{L}\left(\boldsymbol{z}_i^{(k)}, y_i\right), \\ \text{where } k_i &= \underset{j \in \{1,2,...,K\}}{\text{argmin}} \left(g_i^{(j)} > 0\right). \end{aligned} \tag{6.3}$$

Here $g_i^{(j)}$ defines the early exiting criterion for example i at exit j and is defined as below

$$g_i^{(j)} = \begin{cases} 1, & \text{if } \text{argmax}(\boldsymbol{z}_i^{(j)}) = y_i \text{ and } \boldsymbol{z}_i^{(j)}[y_i] > t \\ 0, & \text{otherwise} \end{cases} \tag{6.4}$$

The sum in Eq.(6.3) means that an example $\boldsymbol{X}_i$ accumulates an exit loss $\mathcal{L}\left(\boldsymbol{z}_i^{(k)}, y_i\right)$ up to the first exit k_i where it exits by satisfying the exit criterion of Eq. (6.4). Further, the exit criterion of Eq. (6.4) is satisfied (*i.e.*, outputs a 1) only when the prediction at exit k is both:

correct (*i.e.*, $\text{argmax}(\boldsymbol{z}_i^{(j)}) = y_i$) and *confident* (*i.e.*, $\boldsymbol{z}_i^{(j)}[y_i] > t$). Thus viewed together, Eqs. (6.3) and (6.4) ensure that hard examples exit later by virtue of which, encumber a larger loss.

Instantiating the HAR *loss*

The HAR training objective of Eq.(6.3) is agnostic to the exact instantiation of $\mathcal{L}$ at branch k. In particular, $\mathcal{L}$ can be any loss function useful for class imbalance, including: weighted cross-entropy [43], Focal loss [45], LDAM loss [58] or any combination thereof. We conduct experiments with both weighted cross entropy and the recently proposed LDAM loss as the exit loss type.

When using the class weighted cross-entropy at each exit, the HAR training objective is described as follows

$$\mathcal{L}_{\text{HAR}}^{\text{CE}}(\boldsymbol{z}_i, y_i) = \sum_{k \in [1,\ldots,k_i]} \boldsymbol{w}_{y_i} log\left(\frac{exp(\boldsymbol{z}_i^{(k)}[y_i])}{\sum_{j=1}^{C} exp(\boldsymbol{z}_i^{(k)}[j])}\right), \tag{6.5}$$

where $\boldsymbol{w}_{y_i}$ refers to the class specific weight, which according to prior work, can be set based on the inverse class frequency [151, 152], inverse square root frequency [165, 153] or the effective weighting [43] strategies. This work leverages the weighting strategy from [43] which sets $\boldsymbol{w}_c = \frac{1-\beta}{1-\beta^{n_c}}$, where n_c is the number of samples in class c and β is a hyperparameter with typical values in $\{0.999, 0.9999\}$.

When using LDAM [58] at each exit, the HAR loss is described as

$$\begin{aligned}
\mathcal{L}_{\text{HAR}}^{\text{LDAM}}(\boldsymbol{z}_i, y_i) &= \sum_{k \in [1,\ldots,k_i]} \boldsymbol{w}_{y_i} log\left(\boldsymbol{p}_i^{(k)}\right) \\
\text{where, } \boldsymbol{p}_i^{(k)} &= \frac{exp(\boldsymbol{z}_i^{(k)}[y_i] - \triangle_{y_i})}{exp(\boldsymbol{z}_i^{(k)}[y_i] - \triangle_{y_i}) + \sum_{j \neq y_i} exp(\boldsymbol{z}_i^{(k)}[j])} \\
\text{and } \triangle_{y_i} &= \frac{C}{n_{y_i}}
\end{aligned} \tag{6.6}$$

The quantity $\triangle_{y_i}$ defines a per-class margin that ensures rare classes get a larger margin. It is determined by a hyperparameter C and the number of examples n_{y_i} in class y_i. Following [58], we select C such that the largest margin for any class is 0.5.

Outcome of training with the HAR *loss*

The intended goal of HAR loss is to increase the loss contribution of hard examples. This is formally stated in the following property.

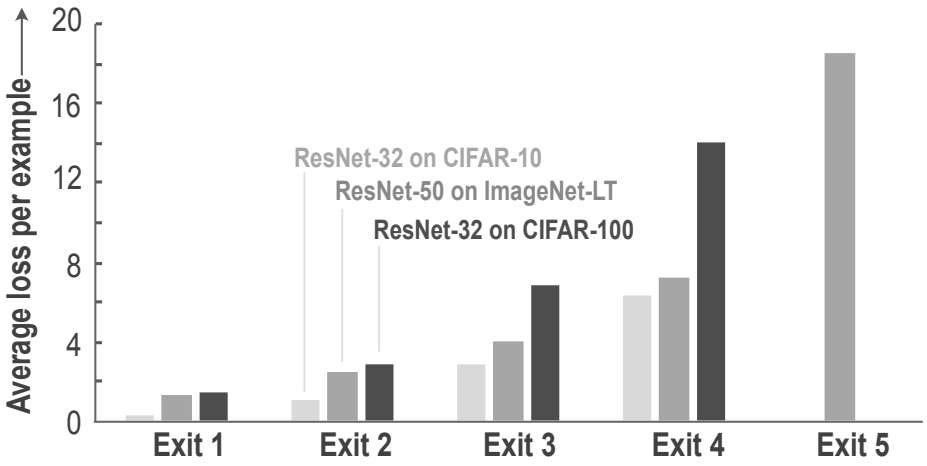


Figure 6.4: On multiple models (ResNet-32/50) and datasets (CIFAR LT/ImageNet LT) we observe that examples exiting later indeed contribute a higher average loss per sample. This figure empirically validates Property 1.

Property 1 *(Increasing Loss Property) For a multi-exit neural network f, if D_k denotes the set of examples exiting at exit k then, $\forall i < j$, $\mathbb{E}_{(\boldsymbol{z}_m, y_m) \in D_i}[\mathcal{L}_{\text{HAR}}(\boldsymbol{z}_m, y_m)] < \mathbb{E}_{(\boldsymbol{z}_m, y_m) \in D_j}[\mathcal{L}_{\text{HAR}}(\boldsymbol{z}_m, y_m)]$.*

The above property simply states that the average training loss for examples exiting at exit *i*, monotonically increases across exits. This enables the neural network to focus on harder examples (which contribute a higher expected loss) from both the minority and majority classes. To empirically validate this property, in Fig. 6.4, we plot the average training loss contributed by examples exiting at each auxiliary branch for ResNet-32 and ResNet-50 models trained with $\mathcal{L}_{\text{HAR}}^{CE}$ in Eq. 6.5, on the CIFAR-10/100 and ImageNet LT datasets. The increasing trend of average loss across exits indeed validates that examples exiting later do contribute a higher loss.

6.3.3 Inference in multi-branch neural networks

Preliminaries

During inference, the outputs of a multi-branch DNN can be aggregated into a single prediction vector in several ways. The popular choices include (1) Inception networks [33] which discard the branch predictions and use only the backbone output as the overall prediction, (2) Hu et al. [32] which uses the mean of all branch outputs as the overall prediction or (3) MSDNet [30] which selects a branch based on prediction confidence, whose output is chosen as the overall prediction. With HAR, a model can support two modes of inference corresponding to choices (1) and (3) above. We term (1) as the *static mode* and (3) as the

dynamic mode. The next subsection dives deeper into the dynamic mode.

Dynamic inference with HAR

Similar to MSDNet [30], HAR selects a branch (based on prediction confidence), who's output is designated as the network prediction. More formally, given the multi-branch prediction $\boldsymbol{z}_i = [\boldsymbol{z}_i^{(1)}, ..., \boldsymbol{z}_i^{(K)}]$ for an input $\boldsymbol{X}_i$, the overall prediction $\hat{\boldsymbol{z}}_i$ of the network is

$$\hat{\boldsymbol{z}}_i = \boldsymbol{z}_i^{(k_i)} \text{ , where } k_i = \underset{j \in \{1,...,K\}}{\operatorname{argmin}} \left(h_i^{(j)} > 0 \right), \tag{6.7}$$

where $h_i^{(j)}$ is the prediction confidence for example i at exit j

$$h_i^{(j)} = \begin{cases} 1, & \text{if } \operatorname{argmax}(\boldsymbol{z}_i^{(j)}) > s \\ 0, & \text{otherwise} \end{cases} \tag{6.8}$$

Intuitively, Eq. 6.7 specifies the overall network prediction as the output of the earliest exit where the exit-criterion of Eq. 6.8 is satisfied. Further, the exit criterion of Eq. 6.8 is satisfied when the prediction confidence (returned by the *argmax*) exceeds a predefined threshold s. Thus, the dynamicity during inference arises out of varying the value of s which affects the exit criterion in the following way: a lower value of s leads to a relaxed version of hardness, and thus more early exiting. Or in other words, s is a control knob for dynamically controlling the inference FLOPS of a network.

6.4 Experiments

In this section, we begin by discussing the experimental setup including: (i) datasets, (ii) evaluation metrics (iii) backbone models and training hyperparameters, (iv) training configurations, and (v) hyperparameter search. Following this, we answer the following three questions:

1. Does HAR benefit in both modes of inference? (Sec. 6.4.3)
2. How does HAR compare to the state of the art? (Sec. 6.4.4)
3. What is the class-composition of early exiting examples in the dynamic mode? (Sec. 6.4.5)

Finally, we end the section with an ablation study on different modelling choices of HAR (Sec. 6.4.6).

6.4.1 Experimental Setup

Datasets. We conduct our evaluation on four long-tailed datasets: CIFAR-10 LT, CIFAR-100 LT [43], ImageNet LT [148] and iNaturalist'18 [157]. For the first three datasets (CIFAR LT & ImageNet LT), the training split is obtained by sub-sampling from their balanced versions. In case of the CIFAR LT datasets, we consider three levels of imbalance, $10\times$, $50\times$ and $100\times$, which is defined as the ratio between the number of samples in the largest and the smallest classes. The ImageNet LT training split consists of 115.8k images from 1,000 classes with largest and smallest classes containing 1,280 and 5 images respectively. The iNaturalist'18 dataset is a naturally imbalanced dataset containing 437,513 training images from 8,142 species of plants and animals. For the ImageNet LT and iNaturalist datasets, similar to [59] we present results on the *many-shot* (classes containing ¿100 examples), *medium-shot* (classes containing 20-100 examples), and *few-shot* (classes containing ¡20 examples) splits. Please refer to Appendix A for additional details on the dataset construction.

Evaluation Metrics. We follow the same evaluation setting as recent methods [43, 58, 59, 60]. For all datasets, the training split is imbalanced (See appendex B.1) while the validation and test splits are balanced. The top1 accuracy on the test split serves as the common metric of comparison across all datasets.

Backbone models & training configurations. We consider several models from the ResNet and DenseNet families. To obtain an augmented HAR model, we attach auxiliary classifier branches before each residual/dense block (see Appendix B for details). On CIFAR datasets, we train the augmented ResNet-32 models for 200 epochs using SGD with an initial learning rate of 0.1 decreased by 0.01 at epochs 160 and 180 [58, 43]. The weight decay is 2×10^{-4}. On ImageNet LT and iNaturalist'18 we train the augmented ResNet-50 and DenseNet-169 models for 100 epochs using SGD with an initial learning rate of 0.1 decreased by 0.1 at epochs 60 and 80. The weight decay is 2×10^{-4}. Similar to [43, 58], all models use a linear warm-up schedule for the first 5 epochs to avoid initial over-fitting.

Implementation. HAR is constructed from three *key* components—(1) an exit loss function, (2) a class reweighting strategy, and (3) a reweighting schedule. For the exit loss, we consider two variations with HAR—at each exit we use either cross entropy loss (CE) or label distribution aware margin loss (LDAM) [58]. These are referred to as $\text{HAR}_{\text{(CE)}}$ and $\text{HAR}_{\text{(LDAM)}}$. For class reweighting, we weight each example of class c according to it's effective number $\frac{1-\beta}{1-\beta^{n_c}}$, where n_c is the number of images in class c [43]. Finally, for the reweighting schedule we use the per-dataset delayed reweighting (DRW) scheme introduced in [58]. All experiments are conducted on a system with four V100 GPUs and

512GB of RAM.

Baselines. To measure the performance of HAR, we compare against three strong baselines: CE [43], Focal [45] and LDAM [58], each reusing the same class reweighting and delayed reweighting schedule discussed above. For Focal loss we set $\gamma = 0.5$ [45], and for LDAM we set C such that the maximum margin is 0.5 [58].

6.4.2 Hyperparameter Search for HAR

HAR introduces two hyperparameters—training and inference exit thresholds t, s in Eq. (6.4),(6.8) respectively. Parameter t controls the number of early exiting examples *during training*. Smaller values result in many examples exiting early leading to a more relaxed definition of example hardness. Parameter s, is a control knob for varying the computational cost during inference. We determine s directly on the test split depending on the desired FLOPS saving, while the optimal value of t is determined through a line search on the validation split. Tab. 6.1 presents the top-1 accuracy on the three (many/med/few shot) splits of ImageNet LT, reached by a ResNet-50 model trained using different values of t.

Selecting t for HAR(CE) For exit loss type cross entropy, we consider a line search in [0.75,0.95] in steps of 0.05. Tab. 6.1a shows the top-1 accuracy peaks when $t = 0.9$.

Selecting t for HAR(LDAM) For exit loss type LDAM [58], we consider a line search in [1.7/—c—, 2.1/—c—] in steps of 0.1/—c— where —c— is the number of classes in the dataset. Tab. 6.1b shows the top-1 accuracy peaks when $t = 2.0/1k$.

We reuse the above hyperparameters: t=0.9 for HAR(CE), and $t = 2.0/|c|$ for HAR(LDAM); on all datasets.

Table 6.1: Line search for inference threshold t in HAR. We observe a clear trend wherein the top-1 validation accuracy (of a ResNet-50 trained on ImageNet LT) increases with larger t until a certain point after which it falls. The column with the optimal value of t is highlighted in gray.

t	0.75	0.8	0.85	0.9	0.95
Many	60.2	60.8	61.1	62.4	62.6
Med	44.6	45.0	45.6	47.3	46.1
Few	25.3	26.0	26.4	28.5	28.1
All	48.0	48.5	48.9	50.5	50.0

(a) Identifying t for HAR(CE)

t	1.7/1k	1.8/1k	1.9/1k	2.0/1k	2.1/1k
Many	59.7	63.2	65.0	65.7	65.5
Med	47.9	48.5	48.4	48.2	47.8
Few	31.2	30.1	30.1	29.9	28.8
All	50.2	51.7	52.3	52.5	52.0

(b) Identifying t for HAR(LDAM)

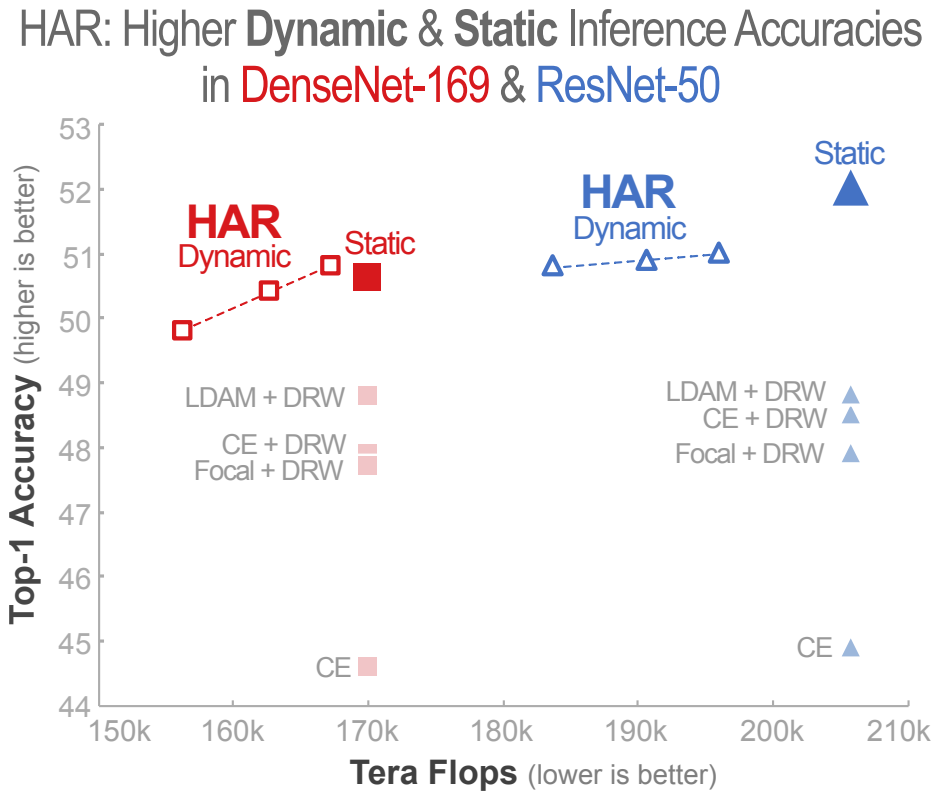


Figure 6.5: The static DenseNet-169 (red squares) and ResNet-50 (blue triangle) trained on ImageNet LT lie along a vertical line and correspond to a *fixed* FLOPS budget. In contrast, the dynamic models trained with HAR lie along an accuracy *vs*. efficiency trade-off curve. Observe that HAR leads to higher accuracy for both static and dynamic modes while additionally leading to FLOPS savings in the dynamic mode.

6.4.3 Two inference modes of HAR

I. Static mode of inference refers to the state when the bare backbone model, without any early exits, is used for inference. Such a model engenders a fixed compute capacity (or FLOPS) during inference. Fig. 6.5 plots the top-1 accuracy *vs*. the inference FLOPS for ResNet-50/DenseNet-169 models trained on ImageNet LT. The solid red squares and solid blue triangles depict the static DenseNet-169 and ResNet-50 models trained with different loss functions. Observe that the static models trained with HAR outperform other loss functions indicating the merits of enabling hardness-awareness during training.

II. Dynamic mode of inference refers to the state when the intermediate branches are preserved (during inference) and used to early-exit easier examples. In this mode, the compute capacity (or FLOPS) can be varied by changing the inference threshold parameter s in Eq. 6.8. Each value of s leads to a new point on the accuracy *vs*. efficiency trade-off curve. The hollow red squares and blue triangles in Fig. 6.5 plot this trade-off for a DesneNet-169 and ResNet-50 model trained with HAR. The three points are obtained by

setting $s \in \{1.7, 1.75, 1.8\} \times 10^{-3}$. We observe that the accuracy-efficiency curve envelopes the other baselines indicating that HAR leads to a favorable trade-off even in the dynamic mode.

	CIFAR-10 LT			CIFAR-100 LT		
	100×	50×	10×	100×	50×	10×
CE†	70.4	74.8	83.6	28.3	43.9	55.7
Focal [45]††	70.4	76.7	86.7	28.4	44.3	55.8
Mixup [166]††	73.1	77.8	87.1	39.5	45.0	58.0
Manifold Mixup [167]††	73.0	78.0	87.0	38.3	43.1	56.5
CE+DRW [43]†	76.3	80.0	87.6	41.4	46.0	58.3
HAR(CE)+DRW (Our)	**76.8**	**80.8**	**87.6**	**42.5**	**47.1**	**58.7**
Flops saving	32%	29%	26%	11%	11%	10%
LDAM+DRW [58]†	77.0	81.4	87.6	42.0	46.6	58.7
HAR(LDAM)+DRW (Our)	**78.1**	**82.4**	**88.0**	**43.1**	**47.5**	**58.9**
Flops Saving	15%	21%	20%	0%	2%	3%

Table 6.2: HAR leads to highest top-1 accuracies while saving inference FLOPS, for ResNet-32 models trained on long tailed CIFAR-10 and CIFAR-100 datasets. We consider three levels of imbalance ($100\times, 50\times, 10\times$). Top rows are recent methods; middle and bottom rows compare HAR fitted with two different exit-loss types (CE/LDAM). Our re-implementation marked by †; results from [60] marked by ††.

Table 6.3: HAR leads to highest top-1 accuracies with compute savings, for ResNet-50 trained on Imagenet LT and iNaturalist'18 datasets. We consider the many-, medium-, and few-shot splits. Top rows are recent methods; middle and bottom rows compare HAR fitted with two different exit-loss types (CE/LDAM). Our re-implementation marked by †; results from [59] marked by ††.

	ImageNet LT				iNaturalist'18			
	Mny	Med	Few	All	Mny	Med	Few	All
CE†	63.8	38.5	13.6	44.6	72.7	63.8	58.7	62.7
CRT [59]††	58.8	44.0	26.1	47.3	69.0	66.0	63.2	65.2
LWS [59]††	57.1	45.2	29.3	47.7	65.0	66.3	65.5	65.9
τ-norm [59]††	56.6	44.2	27.4	46.7	65.6	65.3	65.9	65.6
Focal+DRW [45]†	59.5	44.6	27.0	47.9	66.1	66.0	64.3	65.4
CE+DRW [43]†	60.3	45.2	27.0	48.5	67.1	66.2	**65.4**	65.9
HAR (CE)+DRW (Our)	**60.7**	**45.5**	**27.7**	**48.9**	**67.4**	**66.3**	65.1	**66.0**
Flops Saving				21%				13%
LDAM + DRW [58]†	61.1	44.7	28.0	48.8	70.0	67.4	**66.1**	67.1
HAR (LDAM)+DRW (Our)	**63.8**	**47.2**	**28.1**	**51.0**	**72.2**	**69.0**	65.7	**68.0**
Flops Saving				5%				2%

6.4.4 Comparison to the state-of-the-art

We compare against the s.o.t.a, assuming the dynamic mode of inference for HAR, since it leads to practical compute savings. We observe that HAR is able to improve the accuracy

of s.o.t.a methods (CE+DRW & LDAM+DRW) by incorporating the notion of hardness during training. Further discussion is driven by three broad questions.

I. How does HAR perform under different levels of imbalance? To answer this question, we follow prior work [60, 58] and train a ResNet-32 model on CIFAR-10 LT and CIFAR-100 LT with three imbalance levels $(100\times, 50\times, 10\times)$. The results in Tab. 6.2 present two key observations.

1. **HAR improves accuracy under all imbalance levels** $(100\times,\ 50\times,\ 10\times)$ for both exit-loss types (HAR$_{\text{(CE)}}$ and HAR$_{\text{(LDAM)}}$) while consuming $15-30\%$ fewer FLOPS on CIFAR-10 and $2-10\%$ fewer FLOPS on CIFAR-100.

2. **Accuracy gap increases with imbalance**. The accuracy improvement is higher for greater levels of imbalance (*e.g.*, 1.1%, 1%, 0.4% improvement over LDAM for $100\times, 50\times, 10\times$ on CIFAR-10)

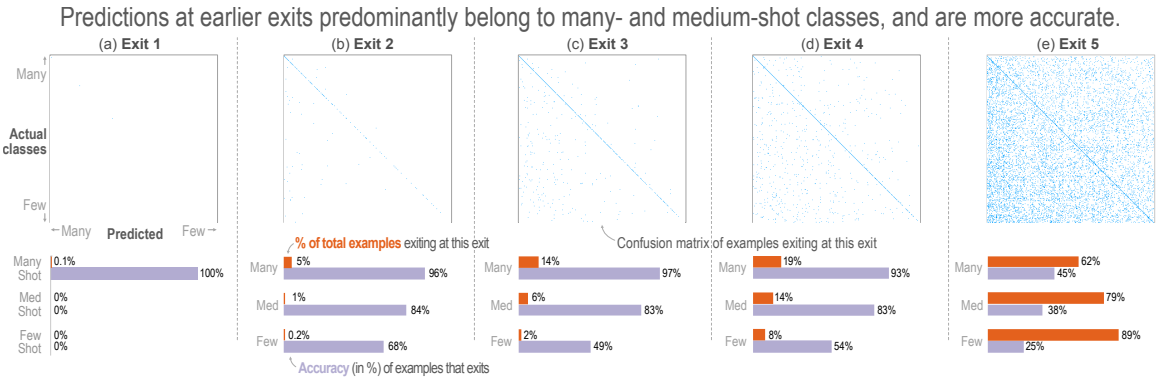


Figure 6.6: For a ResNet-50 trained on ImageNet LT with HAR (LDAM)+DRW, we show at each exit: (*top row*) The confusion matrix of predictions, (*bottom row*) The percentage of total examples for each split (many-, medium-, few-shot) that exit and the accuracy of these predictions. We note that (1) the early exiting examples are predominantly from the many- and medium-shot classes (2) the predictions at early exits, exhibit a high accuracy.

II. Which classes drive the overall improvement in accuracy? To answer this question, we follow prior work [59] to train a ResNet-50 model on ImageNet LT and iNaturalist-18 datasets. In Tab. 6.3, for each method, we dissect its overall top-1 accuracy into accuracies on three class splits: *many-shot* classes (¿100 examples/class), *medium-shot* classes (20-100 examples/class), and *few-shot* classes (¡20 examples/class). Following are the key observations:

1. **Many-shot split drives accuracy improvement**. On both datasets, we observe that the greatest accuracy improvements are observed on the many-shot splits (*e.g.*, $+2.7\%$ relative to LDAM on ImageNet LT, and $+2.2\%$ on iNaturalist'18).

2. **Baseline methods lose significant accuracy on many- and medium-shot classes**. Relative to the CE baseline, we observe SOTA methods lose significant accuracy on the many- and medium-shot classes. This explains how HAR improves overall accuracy: It mitigates the accuracy decline on majority and medium shot classes while leading to comparable accuracy gain on the few-shot classes.

6.4.5 Analyzing the dynamic inference mode

I. Which classes tend to exit earlier? To answer this question, in Fig. 6.6, we present the confusion matrix and the class-composition for each exit of a ResNet-50 trained on ImageNet LT with HAR. Following are some key observations.

1. **Early-exiting examples are predominantly from the majority classes.** The confusion matrices in the top row show a distribution shift from a top-left bright diagonal in part a (exit-1) to a middle heavy diagonal in part d (exit 4). Since the 1000 classes are sorted according to size such that top and left contain the majority classes while bottom and right contain the rare classes, this observation indicates that the initial exits prefer the many-shot classes while the later exits prefer the medium-shot classes. The red bar plots of Fig. 6.6a-e reinforce this observation by showing the percentage of each split that has exited by each exit. Notice that before last exit, 38% of the many-shot, 21% of the medium-shot and 11% of the few-shot split have already been classified.

2. **Early exiting examples are more trustworthy due to higher accuracy.** The purple bar plots of Fig. 6.6a-e present the accuracy for examples exiting at each exit. Notice that for all three splits (many, med, few shots), the accuracy for exits 1-4 is nearly double than that of exit-5. This indicates that an early prediction is much more trustworthy (due to higher accuracy) than that a late prediction.

6.4.6 Ablation studies

How does HAR perform without class reweighting? Tab. 6.4 presents the accuracy of a ResNet-50 model trained on ImageNet LT with three different loss functions—focal, cross entropy and LDAM—and without any class reweighting at the early-exits. We observe that the (static) models trained with HAR outperform the vanilla loss functions (CE, LDAM) on all the three splits.

What is the impact of the location of early exits? To measure this, in Tab. 6.5, we ablate on the location of a single early exit attached to a ResNet-50 model trained on ImageNet LT

Table 6.4: HAR leads to the highest top-1 accuracy for a ResNet-50 model trained on ImageNet LT, and, without any class re-weighting. We consider the static inference mode.

Loss	Many	Med	Few	All
Focal ($\gamma = 0.5$)	63.5	38.5	13.6	44.7
Focal ($\gamma = 1.0$)	62.8	37.1	12.7	43.7
Focal ($\gamma = 2.0$)	62.1	36.6	11.9	43.1
CE	63.8	38.5	13.6	44.9
HAR (CE)	**63.9**	**39.9**	**16.0**	**45.9**
LDAM	64.9	39.1	12.6	45.5
HAR (LDAM)	**66.3**	**42.4**	**14.2**	**47.8**

Table 6.5: Ablating on the location of a single early exit (indicated by a E) using a *static* ResNet-50 trained with HAR (LDAM)+ DRW on ImageNet LT. The naming convention specifies C for a convolution layer, B for a residual block and E for an early exit. Configuration CBBEBBE outperforms others, suggesting the value of an early exit peaks between blocks 2,3.

Model	Many	Med	Few	All
CBBBBE	61.1	44.7	28.0	48.8
CEBBBBE	62.3	44.9	25.9	49.0
CBEBBBE	63.0	45.9	26.7	49.9
CBBEBBE	**63.3**	**46.6**	**27.8**	**50.5**
CBBBEBE	62.5	46.0	27.6	49.8

using the HAR (LDAM)+DRW loss. For comparison, we use the static mode, which means the exits are only used during training and are discarded for inference. Among the several configurations (with naming convention: C=conv layer, B=block, E=exit), the one with an exit after block 2—CBBEBBE—outperforms all others. This suggests a diminishing value of placing an exit too early/late in the backbone.

Can we visualize the learned notion of example hardness? Fig. 6.7 presents a randomly selected subset of test split images exiting through the auxiliary branches of a HAR ResNet-50 model trained on ImageNet LT. Among each class, we observe that the object of interest is easier to distinguish in images exiting from earlier branches which indicates that HAR enables a model to learn an intuitive notion of image hardness.

6.5 Conclusion

We identified the notion of *sample-hardness* as a key concept to improve generalization under a long-tailed class distribution. To incorporate this notion of hardness in the learning process, we proposed the HAR framework. HAR is complementary to existing work in

Figure 6.7: Images exiting from the first, third and the final exit of a HAR model indicate that as the exits increase, so does the visual hardness.

long-tailed classification and can readily integrate with existing approaches to improve classification accuracy. Extensive evaluations demonstrate that HAR outperforms existing state-of-the-art techniques while saving inference FLOPS.

CHAPTER 7
IMB-NAS:NEURAL ARCHITECTURE SEARCH FOR IMBALANCED DATASETS

Class imbalance is a ubiquitous phenomenon occurring in real world data distributions. To overcome its detrimental effect on training accurate classifiers, existing work follows three major directions: class re-balancing, information transfer, and representation learning. In this paper, we propose a new and complementary direction for improving performance on long tailed datasets—optimizing the *backbone architecture* through neural architecture search (NAS). We find that an architecture's accuracy obtained on a balanced dataset is not indicative of good performance on imbalanced ones. This poses the need for a full NAS run on long tailed datasets which can quickly become prohibitively compute intensive. To alleviate this compute burden, we aim to efficiently adapt a NAS super-network from a balanced source dataset to an imbalanced target one. Among several adaptation strategies, we find that the most effective one is to retrain the linear classification head with reweighted loss, while freezing the backbone NAS super-network trained on balanced source dataset. We perform extensive experiments on multiple datasets and provide concrete insights to optimize architectures for long tailed datasets.

7.1 Introduction

The natural world follows a long tail data distribution wherein a small percentage of classes constitute the bulk of data samples, while a small percentage of data is distributed across numerous minority classes. Training accurate classifiers on imbalanced datasets has been an active research direction since the early 90s. Much of prior work [59, 168, 169] centers on improving the performance (measured via accuracy) of a fixed backbone architecture such as ResNet-32. In this work, we take a complementary direction and aim to optimize the backbone architecture via neural architecture search. Indeed this is an important direction since prevalent practices demand that neural architectures be optimized to fit the size/latency constraints of tiny edge devices.

To optimize the backbone architecture, we rely on the recent work from Neural Architecture Search (NAS) [170] that optimizes a neural network's architecture primarily on datasets that are *balanced* across classes. This workflow naturally prompts the question: is the architecture optimized on a class balanced dataset also the optimal one for imbalanced datasets? Table 7.1 provides evidence to the contrary. The first row shows two

Table 7.1: Motivation. We sample four architectures A1-A4 from the DARTS search space and train them on balanced (i.e. $1\times$) and imbalanced versions (i.e. $100\times$) of Cifar10 and Cifar100. (**Top**) Two similarly sized architectures (A1,A2) achieve similar accuracy on balanced Cifar10, but differ by 3% in presence of $100\times$ imbalance. (**Bottom**) The larger architecture (A3) outperforms the smaller on (A4) on balanced Cifar100, but under performs by 3.6% in the presence of $100\times$ imbalance. This suggests that an architecture's performance on balanced datasets is not indicative of it's performance on imbalanced ones.

Dataset	Model	Flops	Accuracy (%)	
			bal($1\times$)	imbal($100\times$)
Cifar10	A1	410	94.6	77.3
	A2	407	94.7	74.1
Cifar100	A3	400	76.1	39.4
	A4	179	75.0	43.0

architectures–A1,A2–sampled from the DARTS search space [171] having similar size and accuracy on balanced Cifar10, but an accuracy gap of 3% in the presence of $100\times$ imbalance. The second row compares a larger architecture A3 outperforms a smaller one A4 on balanced Cifar100. However, in the presence of $100\times$ imbalance, the smaller architecture outperforms the larger one by more than 3%. These results and more in Sec 7.3.2, indicate that the optimal architecture on a balanced dataset may not be the optimal one for imbalanced datasets. This means each target imbalanced dataset requires its own NAS procedure to obtain the optimal architecture.

Running a NAS procedure for each target dataset is computationally expensive and quickly becomes intractable in the presence of multiple target datasets. To overcome the compute burden of running NAS from scratch, we formalize the task of architectural rank adaptation from balanced to imbalanced datasets. Towards this task, Section 7.3.4 describes two intuitive rank adaptation procedures that either fine-tune the classifier only, or together with the backbone. Our comprehensive experiments reveal the key insight that the adaptation procedure is most affected by the linear classification head trained on top of the backbone. Armed with this insight, we propose to re-use a NAS super-net backbone trained on balanced data and re-train only the classification head to efficiently adapt a pre-trained NAS super-net for imbalanced data. This is extremely efficient since it involves training only a linear layer on top of the pre-trained super-network.

Overall, our contributions in this work are:

1. **New insight.** We show that architectural rankings transfers poorly from balanced to imbalanced datasets.

2. **Novel task.** We construct the novel task to efficient adapt a NAS super-network from

balanced to imbalanced datasets.

3. **Novel solution.** We propose a simple and efficient solution–retraining the classifier head while freezing the backbone–to efficiently adapt a NAS super-network from balanced to imbalanced datasets.

7.2 Related Works

We cover relevant work from three related areas.

7.2.1 Overcoming long tail class imbalance

Prior work on tackling long tail imbalance can divided into three broad areas (see survey [172]): class-rebalancing that includes data re-sampling (SMOTE [49], ADASYN [51]), loss re-weighting [59, 43, 173, 169], logit adjustment [174, 175, 176]; Information augmentation that includes transfer learning [152, 177], data augmentation [178]; and module improvement that encompasses methods in representation learning [148], classifier design [179], decoupled training [59] and ensembling [168]. Different from all of the existing works, our work explores a new direction of performance improvement on long tail datasets–that via optimizing the backbone architecture. This complements existing approaches and can work in tandem to further boost accuracy and efficiency on imbalanced datasets.

7.2.2 Neural architecture search

Prior work on architecture search can be categorized in improving its three main pillars (see survey [180])—Search space design with the idea of incorporating a large diversity of architectures. Popular spaces include cell based spaces such as NASNets [23], and recent spaces from the ShuffleNet [181] and MobileNet [182] model families. The second pillar constitutes search strategy design to efficiently locate performant architectures from the search space. Popular strategies involve reinforcement learning [183, 23], evolutionary algorithms [184, 185] or gradient descent on continuous relaxations of the search space [171]. The third pillar constitutes performance estimation strategies [186, 187] with the goal of cheaply estimating the goodness (in terms of accuracy or efficiency) of an architecture. All of the above works search optimal architectures on datasets that are fully balanced across all classes. Our experiments however show that the set of optimal architectures differ significantly from balanced to imbalanced datasets. This calls for developing

new NAS methods or efficient adaptation strategies (e.g. this work) to search for optimal architectures on real world, imbalanced datasets.

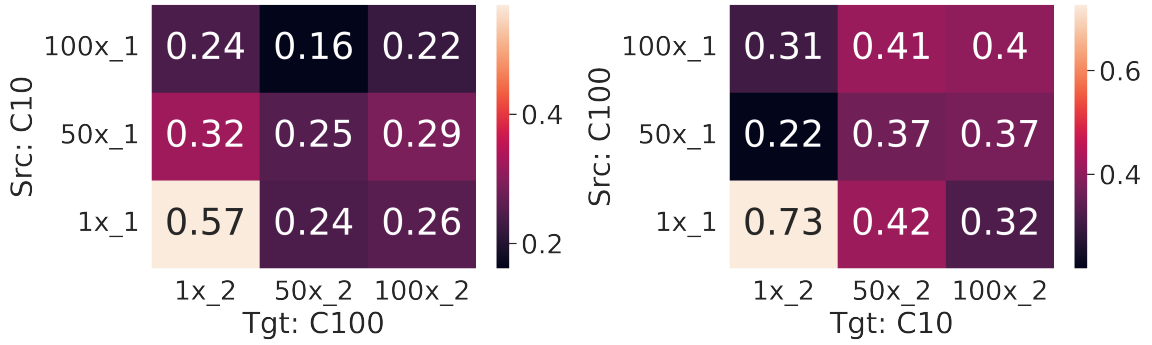
7.2.3 Architecture transfer

We summarize prior work on evaluating robustness of architectures to distributional shifts in the training dataset. Neural Architecture Transfer [188] explore architectural transferability from large-scale to small-scale fine grained datasets. However, there are two limitations–the source and target datasets considered in this work are balanced across all classes and additionally this work assumes all target datasets are known apriori which is infeasible in many industry use-cases. NASTransfer [189] consider transferability between large-scale imbalanced datasets including ImageNet-22k which is a highly imbalanced dataset. Their approach is practically useful for very large datasets (e.g. ImageNet-22k) for whom direct search is prohibitive, however when it is feasible (e.g. on ImageNet) direct search typically leads to better architectures than proxy search. Differing from these, our work advocates to directly adapt a super-network pre-trained on fully balanced datasets (instead of proxies) to imbalanced ones. A key feature of such adaptation is efficiency—the compute required for such an adaptation needs to be much lesser than that for repeating the search on the target dataset.

7.3 Methodology

7.3.1 Notation

Assume $\mathcal{D} = \{x_1, y_i\}$ denotes the training dataset of images where y_i is the label for image x_i. Let n_j specify the number of training images in class j. After sorting the classes by cardinality in decreasing order, the long tail assumption specifies that if $i < j$, then $n_i \geq n_j$ and $n_1 >> n_C$. We use ϕ to denote a deep neural network that is composed of a backbone $\phi(a, w_a)$ with architecture a, weights w_a and a linear classifier $\phi(w_c)$. The model ϕ is trained using a training loss and loss re-weighting strategy. On balanced datasets, we use the cross entropy loss (denoted as CE) to train a neural network. For imbalanced datasetsm we additionally incorporate the effective re-weighting strategy [43] that reweights samples from class j with $\frac{1-\beta}{1-\beta^{n_j}}$ where β is a hyperparameter. Following previous works [58, 173], the re-weighting strategy is applied after a delay of few training epochs which is denoted using the shorthand DRW.



(a) $\mathcal{D}_s$: Cifar10, $\mathcal{D}_t$: Cifar100 **(b)** $\mathcal{D}_s$: Cifar100, $\mathcal{D}_t$: Cifar10

Figure 7.1: Evaluating architectural transferability. We train all 149 Mflop architectures from NATS-Bench on Cifar10, Cifar100 with $1\times, 50\times, 100\times$ imbalance and compute kendall tau correlation between the rank orderings on all datasets. We observe high correlation (bottom left cells) when both $\mathcal{D}_s, \mathcal{D}_t$ are balanced, and low correlation otherwise. This means that architectural rankings transfer poorly across data imbalance.

7.3.2 Architecture ranking transfer: A motivating experiment

We study the impact of backbone architecture on imbalanced datasets using the following experiment. We construct an architecture search space $\mathcal{A}$ by sampling all 149 Mega Flops architectures from the NATS-bench search space [190]. Overall $\mathcal{A}$ contains 135 architectures with exactly the same learning capacity (or Flops), but different architectural patterns (e.g. kernel sizes, layer connectivity). The architectures in $\mathcal{A}$ are trained on the source and target datasets D_s, D_t using loss function CE on balanced datasets, and the re-weighted loss function CE+DRW on imbalanced ones. Following this, the architectures are ranked based on validation accuracy and the kendall Tau metric is computed between the rank orderings obtained on D_s and D_t. A high correlation means similar architectural rankings on both datasets, while a low correlation implies widely different rankings.

Figure 7.1 presents the outcomes on two scenarios: (1) D_s is Cifar10 at three levels of imbalance $(1\times, 50\times, 100\times)$ and D_t is Cifar100 at the same imbalance levels; and (2) the opposite direction. There are two major observations—First, the high correlation in the bottom left square indicates that the architectural rankings transfer quite well across balanced datasets. Second, the low correlation for all other cells indicates low transferability across imbalanced datasets. This means the rank orderings on imbalanced datasets widely differs from that on balanced ones.

To avoid the compute burden of performing a NAS run on every target imbalanced dataset, we develop efficient "adaptation" procedures to adapt a NAS super-net from balanced to imabalanced datsets. Before going into the details, in the next section we provide a brief overview of exisiting NAS methods.

7.3.3 Revisiting neural architecture search

We look at sampling based NAS methods that involve two steps. The first step involves training a super-network with backbone $\phi(a, w_a)$ and classifier $\phi(w_c)$ on a training dataset D via the following minimization

$$w^*_{a,\mathcal{D}}, w^*_{c,\mathcal{D}} = \underset{w_a, w_c}{min} \underset{a \sim \mathcal{A}}{\mathbb{E}} \left(\mathcal{L}(\phi(w_c), \phi(a, w_a); \mathcal{D}) \right). \tag{7.1}$$

Here the inner expectation is performed by sampling architectures a from a search space $\mathcal{A}$ via uniform, or attentive sampling.

The second step involves searching the optimal architecture that maximizes validation accuracy via the following optimization

$$a^*_{\mathcal{D}} = \underset{a \in \mathcal{A}}{max} \, Acc \left(\phi(w_c), \phi(a, w_a); \mathcal{D} \right). \tag{7.2}$$

This maximization is typically implemented via evolutionary search or reinforcement learning. Next, we discuss efficient adaptation procedures to adapt a NAS super-net trained on a balanced dataset onto an imbalanced one.

7.3.4 Rank adaptation procedures

Given source and target datasets $\mathcal{D}_s, \mathcal{D}_t$, we first train a super-network on D_s by solving the following optimization

$$w^*_{a,\mathcal{D}_s}, w^*_{c,\mathcal{D}_s} = \underset{w_a, w_c}{min} \underset{a \sim \mathcal{A}}{\mathbb{E}} \left(\mathcal{L}(\phi(w_c), \phi(a, w_a); \mathcal{D}_s) \right). \tag{7.3}$$

Our goal then is to adapt the optimal super-net weights $w^*_{a,\mathcal{D}_s}, w^*_{c,\mathcal{D}_s}$ found on D_s to the target dataset D_t which suffers from class imbalance. The most efficient adaptation procedure involves freezing the backbone, while adapting only the linear classifier on D_t by minimizing the re-weighted loss $\mathcal{L}_{RW}$

$$w^*_{c,\mathcal{D}_t} = \underset{w_c}{min} \underset{a \sim \mathcal{A}}{\mathbb{E}} \left(\mathcal{L}_{RW}(\phi(w_c), \phi(a, w^*_{a,\mathcal{D}_s}); \mathcal{D}_t) \right). \tag{7.4}$$

The resulting super-network contains backbone weights $w^*_{a,\mathcal{D}_s}$ trained on $\mathcal{D}_s$ and classifier weights $w^*_{c,\mathcal{D}_s}$ trained on $\mathcal{D}_t$. Solving the above optimization is extremely efficient since most of the network is frozen while only the classifier is trained. On the other hand, one could also adapt the backbone by fine-tuning on the target dataset. This is achieved by

minimizing the delayed re-weighted loss $\mathcal{L}_{DRW}$

$$w^{**}_{a,\mathcal{D}_t}, w^{*}_{c,\mathcal{D}_t} = \underset{w_a, w_c}{min} \underset{a \sim \mathcal{A}}{\mathbb{E}} \left(\mathcal{L}_{DRW}(\phi(w_c), \phi(a, w^{*}_{a,\mathcal{D}_s}); \mathcal{D}_t) \right). \tag{7.5}$$

Here, the double star on $w^{**}_{a,\mathcal{D}_t}$ indicates the weights were obtained via fine-tuning $w^{*}_{a,\mathcal{D}_s}$ using one tenth of the original learning rate and one third the number of original training epochs. Also, recall that the delayed re-weighted loss $\mathcal{L}_{DRW}$ is nothing but the unweighted loss $\mathcal{L}$ in the first few epochs and the re-weighted loss $\mathcal{L}_{RW}$ subsequently. Note that our second adaptation procedure is more compute intensive since the backbone is also adapted, but still much less intensive than running the full search on the target dataset.

Our final and most compute intensive procedure involves directly searching on the target dataset via $\mathcal{L}_{DRW}$. This is achieved via the following minimization

$$w^{*}_{a,\mathcal{D}_t}, w^{*}_{c,\mathcal{D}_t} = \underset{w_a, w_c}{min} \underset{a \sim \mathcal{A}}{\mathbb{E}} \left(\mathcal{L}_{DRW}(\phi(w_c), \phi(a, w_a); \mathcal{D}_t) \right). \tag{7.6}$$

Table 7.2: Summarizing rank adaptation procedures.

Adj	Eqn	Description
P0	(7.3)	No adaptation.
P1	(7.4)	Freeze backbone, retrain classifier on $\mathcal{D}_t$.
P2	(7.5)	Finetune backbone and retrain classifier on $\mathcal{D}_t$
P3	(7.6)	Re-train backbone and classifier on $\mathcal{D}_t$.

The three adaptation procedures and their associated compute costs are summarized in Table 7.2.

7.4 Experiments

We begin this section by answering which rank adaptation procedure works best, both in terms of efficiency of the procedure and the accuracy of the resulting networks. We then perform an extensive ablation study to uncover the effect of different design choices.

7.4.1 Implementation details

We implement our methods using Pytorch on a system containing 8 V100 GPUs. Other details are as follows:

Datasets. We construct imbalanced versions of Cifar-10 and Cifar-100 by sub-sampling from their original training splits [43]. The c^{th} class in the resulting datasets contains

Table 7.3: Comparing rank adaptation strategies. Given a NAS super-net trained on $\mathcal{D}_s$, we adapt it to $\mathcal{D}_t$ and search the optimal sub-nets. These are retrained from scratch on $\mathcal{D}_t$ and the average validation accuracy is presented. Note that sub-nets obtained via P1/P2 outperform P0 for high imbalance ratios (shaded yellow) and typically P1 outperforms P2 (the winner is bolded). Results averaged over three seeds.

	Adp	Imbalance Ratio			
		50×	100×	200×	400×
baseline	P0	45.80	40.83	36.30	32.80
	P1	**45.06**	**41.93**	**36.76**	**33.70**
	P2	44.86	41.86	36.70	33.46
paragon	P3	45.93	41.53	37.03	33.40

(a) Cifar10-1× → Cifar100-{50, 100, 200, 400}×

	Adp	Imbalance Ratio			
		100×	200×	400×	800×
baseline	P0	75.96	68.96	63.26	56.90
	P1	**75.93**	**69.70**	**63.80**	**58.23**
	P2	75.86	69.26	63.70	58.03
paragon	P3	76.03	70.23	63.96	57.70

(b) Cifar100-1× → Cifar10-{100, 200, 400, 800}×

$n_c = n\mu^c$ examples where n is the original cardinality of class c, and $\mu \in [0, 1]$. We select μ such that the imbalance ratio—which is defined as the ratio between the number of examples in the largest and smallest class—is $50\times$ to $1000\times$.

Sub-network training strategies. We train a network on balanced Cifar-10/100 for 200 epochs with an initial learning rate of 0.1 decayed by 0.01 at epochs 160 and 180 using the cross entropy loss. On imbalanced versions, we introduce effective re-weighting [43] at epoch 160 and refer to this strategy as delayed re-weighting or DRW-160 [58].

Neural Architecture Search We train a super-network for 600 epochs with an initial learning rate of 0.1, decayed by 0.01 at epochs 400 and 500. On imbalanced datasets, re-weighting is applied at epoch 400. For searching the best subnet, we follow [170] and use an evolutionary search with 20 generations, population of 50, crossover number 25, mutation number 25, mutate probability 0.1 and top-k of 10.

Adaptation Strategies To adapt a super-network, we fine-tune it for 200 epochs with an initial LR of 0.01, decayed by 0.01 at epoch 100. In case of procedure P1, we introduce re-weighting at epoch 1. For P2, we delay the re-weighting to epoch 100. For P3, we follow the NAS strategy detailed above.

7.4.2 Baseline and Paragon for IMB-NAS

Given a NAS super-network trained on a source dataset $\mathcal{D}_s$, our goal is to efficiently adapt it to the target dataset $\mathcal{D}_t$ following which, the best sub-net is searched in the adapted super-net. Table 7.3a illustrates the results for the case when $\mathcal{D}_s$ is Cifar10, and $\mathcal{D}_t$ is Cifar100 with varying levels of imbalance. The first row (i.e. P0) refers to the case when the best sub-nets obtained on $\mathcal{D}_s$ are re-trained on $\mathcal{D}_t$. This serves as our lower bound or *baseline*. The last row (i.e. P3) refers to the case when the NAS super-net is trained on $\mathcal{D}_t$. This

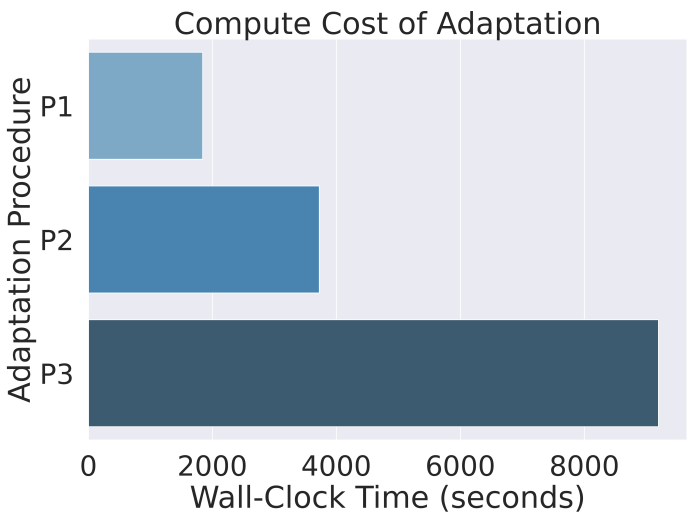


Figure 7.2: Comparing the compute cost of adapting a NAS super-net trained on Cifar10-1$\times$ onto Cifar100-100$\times$. The y-axis plots the wall clock time spent on a single V100 GPU. Observe that P2 and P3 consume $2\times$ and $5\times$ the cost of P1. Results averaged over three seeds.

serves as the upper bound or the *paragon* of accuracy. Our two adaptation procedures (P1, P2) in the middle rows are highlighted yellow when they outperform the baseline, and the better among the two is bolded.

Observe from Tables 7.3a,7.3b that both adaptation procedures comprehensively outperform the baseline at higher levels of imbalance. This means that the architectures searched on $\mathcal{D}_s$ can no longer be assumed as the optimal ones on imbalanced target datasets. Interestingly, between P1 and P2, we find that P1 consistently outperforms P2. This is surprising since P2 also adapts the NAS backbone on the target data whereas P1 re-uses the backbone from the source dataset. We hypothesize this occurs because, class imbalance is much larger an issue for searching the NAS backbone than the domain difference between Cifar10 and Cifar100.

Overall, we find that P1 and P2 achieve very close accuracy to the paragon (P3) while avoiding much of the compute burden of P3 as illustrated in the next section.

7.4.3 Dissecting the performance adaptation

In this section, we analyze different aspects of procedures P1-P3 by applying them to adapt a NAS super-net pre-trained on Cifar10-1x onto Cifar100-100x.

Comparison on training cost. We measure the wall-clock training time on a single V100 GPU as a proxy for training cost. The amortized training cost over three runs is presented in Fig 7.2. It takes P1 2000 seconds to adapt a NAS super-network from cifar10-1$\times$ to cifar100-100$\times$. In comparison P2 consumes $2\times$, and P3 consumes $5\times$ more time. These

Table 7.4: Ablating on the number of backbone fine-tuning epochs with P2 while adapting from Cifar10-1× to Cifar100-$\{100, 200, 400\}\times$. Coinciding the freezing of the backbone with the loss re-weighting at epoch 100 typically outperforms the baseline. Generally more fine-tuning epochs are better. Results averaged over three seeds.

Proc	Epochs	Imbalance Ratio		
	-	100×	200×	400×
P0	-	40.83	36.3	32.80
P1	-	41.93	36.76	33.70
P2	50	40.06	35.46	32.56
	100	41.43	37.26	33.00
	150	41.40	36.63	32.86
	200	41.86	36.70	33.46
P3	-	41.53	37.03	33.4

Table 7.5: Ablating on the loss used to train the NAS super-net on Cifar100-100×. Results presented are the validation accuracy of optimal sub-networks searched from corresponding super-networks. It is inconclusive if training the NAS super-net with re-weighted loss (CE-DRW) induces better sub-networks. Results averaged over three seeds.

Train Loss	Imbalance Ratio		
	100×	200×	400×
CE	41.36	37.5	33.9
CE-DRW	41.53	37.0	33.4

results demonstrate that not only P1 can successfully adapt a super-net to improve accuracy, it is also very efficient.

Impact of fine-tuning the backbone with P2. In procedure P2, we adapt the NAS backbone via fine-tuning on the target dataset $\mathcal{D}_t$. One may wonder, can the backbone be frozen after the loss re-weighting is applied? The intuition being that re-weighting mainly helps adapt the classification boundary while negatively affecting the representation learned by the backbone [59]. To answer this, Table 7.4 presents an ablation on the number of epochs spent on fine-tuning the NAS backbone with P2. Observe that too few fine-tuning epochs (e.g. 50) leads to low sub-net accuracy. At the other end, fine-tuning for 100 epochs is sufficient to improve the sub-net accuracy beyond the paragon (P3). This means that one could further lower the compute burden of P2 by freezing the backbone once loss re-weighting is applied at epoch 100.

Training the NAS super-net with loss re-weighting. We observe that loss re-weighting generally results in improved super-net accuracy on imbalanced datasets. Does this mean

Table 7.6: Dissecting the overall accuracy on Cifar100-$\{100, 400\}\times$ into the accuracy on classes containing many (i.e. > 100), medium (i.e. between 20-100) and few (i.e. < 20) examples per class. Sub-networks obtained via P1 and P2 outperform the baseline (P0) for all class categories (shaded yellow). Results averaged over three seeds.

Adj	Imbalance Ratio							
	$100\times$				$400\times$			
	High	Med	Low	All	High	Med	Low	All
P0	64.1	40.4	14.1	40.8	65.0	39.1	10.1	32.8
P1	65.2	41.6	15.0	41.9	66.3	41.0	10.2	33.7
P2	65.2	40.9	15.7	41.8	66.2	40.3	10.2	33.4
P3	65.0	41.5	14.1	41.5	66.2	39.5	10.5	33.4

the resulting sub-nets are better than the ones obtained from a super-net trained without loss re-weighting? We answer this question we train super-nets on Cifar100-100x with and without re-weighting. Then we search and train the best sub-nets which are presented in Table. 7.5. We find that there is no clear winner among the two NAS training approaches.

Dissecting the overall accuracy improvement. To analyze which classes contribute to an increase accuracy, Table. 7.6 dissects the overall accuracy (denoted by column "All") into the accuracy obtained on classes containing Many (i.e. > 100), Medium (i.e. between 20-100) and Few (i.e. < 20) examples per class. For both $100\times$ and $400\times$ levels of imbalance, the architectures obtained via P1 and P2 outperform those obtained by $P0$ for all class categories. This means that indeed the architectures obtained via P1,P2 are able to learn better representations.

7.5 Conclusion

This work aims to improve performance on class imbalanced datasets by optimizing the backbone architecture. Towards this goal, we discover that an architecture's performance on balanced datasets is not indicative if its performance on imbalanced ones. This observation suggests re-running NAS on each target dataset. To overcome the prohibitive compute burden or re-running NAS, we propose to adapt a NAS super-net trained on balanced datasets onto imbalanced ones. We develop multiple adaptation procedures and find that re-training the linear classification head while freezing the NAS super-net backbone outperforms other adaptation strategies both in terms of efficiency of the adaptation and the accuracy of the resulting sub-networks.

CHAPTER 8
CONCLUSIONS AND FUTURE DIRECTIONS

This dissertation addresses the fundamental and practical challenges for empowering artificial intelligence on edge devices. This entails developing (1) New frameworks for efficient edge AI; and (2) New methods for robust edge AI. Towards this goal, this thesis makes the following contributions.

New algorithms to obtain efficient and robust models.

- Our CUP algorithm (Chapter 3) can prune large deep neural networks trained on ImageNet to more that $2\times$ while reducing training time by over 10 hours to obtain a compact DNN with less than 1% drop in Top-5 accuracy.
- Our CMP-NAS algorithm (Chapter 4) can search for efficient query models, which fitted into a heterogeneous visual search system leads to $80\times$ and $23\times$ compute cost reduction while maintaining accuracy within 0.3% and 1.6% of the paragon for fashion and face image retrieval respectively.
- Our REST algorithm (Chapter 5) produces highly-robust and efficient models that substantially outperform the original full-sized models in the presence of noise. For the sleep staging task over single-channel EEG, the REST model achieves a macro-F1 score of 0.67 vs. 0.39 achieved by a state-of-the-art model in the presence of Gaussian noise while obtaining $19\times$ parameter reduction and $15\times$ MFLOPS reduction on two large, real-world EEG datasets.
- Our HAR algorithm (Chapter 6) endows hardness awareness during the learning process thereby improving state-of-the-art accuracy of networks trained on large imabalanced datasets while saving up to 20% of inference FLOPS.
- Our IMB-NAS algorithm (Chapter 7) searches for efficient network architectures on long tail datasets while saving $5\times$ compute compared to searching from scratch. The searched networks improve accuracy on imbalanced datasets compared to architectures optimized on full balanced datasets.

New insights and knowledge

- We are among the **first to demonstrate the viability of a home based sleep apnea monitoring and diagnosis system**. Through robust experiments, we show that a small neural network can be trained to be robust to real-world gaussian noise such that it can maintain a reasonable accuracy of 67% whereby the state-of-the-art accuracy falls to 39%.
- We are the **first to develop a heterogeneous visual search system that extends real world applications** such as video based threat monitoring and image based product search onto edge devices.
- We are the **first to demonstrate** that architectures optimized on fully class balanced datasets are not the optimal ones for imbalanced datasets.

8.1 Impact

Beyond the contributions to the research community, my work has also benefited society and industry.

- The CMP-NAS system, developed during my internship with Amazon was **showcased to thousands of developers** in the company-wide all-hands meeting. It was also the subject of a patent application.
- CMP-NAS and its follow up work REG-NAS resulted in an **Amazon post-internship fellowship** to fund my research until graduation.
- Our work on REST was **highlighted by several news media outlets** for its ability to accurately monitor sleep in the wild.

8.2 Future Directions

While this dissertation expands the understanding around deploying machine learning models in the wild, it also unlocks new research directions along model compression, neural architecture search and mitigating the effects of long tail class imbalance.

Model compression and specialization
In chapter 3, we propose the CUP algorithm as a means of compressing neural networks for deployment to resource constrained devices. The training time version of CUP can also be extended to the problem of model specialization. This problem of model specialization can support scenarios requiring models trained only on a few classes. One could start off with a huge model trained on ImageNet that contains 1000 classes. Then this model can be

pruned / specialized during further fine-tuning on a subset of classes (*e.g.*, Indoor objects such as tables and chairs). The idea of model specialization assumes more importance in the regime of self supervised learning which typically trains a huge central model in an unsupervised way. Then this model is fine-tuned for a downstream task. One can imagine also pruning / specializing the central model on the downstream task.

Searching for compatible gallery and query models
In CMP-NAS, we build a heterogeneous visual search system by searching for a compatible query model against the *fixed* gallery one. One can further improve compatibility (and retrieval accuracy) by also optimizing the gallery model against the query one. Going a step further, jointly designing the query and gallery models together could lead to highly efficient heterogeneous systems. Orthogonally, the effect of weight and activation quantization remains to be seen with respect to compatibility. This is an important direction to explore since prevalent model deployment practices typically reduce quantize to 8 bits without loss of accuracy.

Beyond sleep monitoring
In REST, we design highly efficient and robust deep neural networks for the task of sleep staging on EEG signals. However, the techniques developed are not limited to the particular application or sensing modality. Indeed, modern fitness trackers operate on multi-modal data (*e.g.*, using accelerometers, gyroscopes, ECG) for activity monitoring, detecting heart rate variability, and estimating calories consumed. These tasks can benefit through robust and efficient deep neural networks.

Dynamic inference for hardness awareness and beyond
In HAR, we employ dynamic inference to impart hardness awareness to a loss function. This improves performance on imbalanced datasets by focusing a model's attention on harder examples. A core element in this work was a multi-branch neural network that can save compute by early exiting easier examples. Future directions can employ multi-branch networks to learn diverse tasks e.g. each branch focuses on a specific task while the trunk benefits from multi-task learning. This idea is already gaining traction in the natural language community with the PaLM model [191]. Another promising direction involves on-the-fly compute adjustment of neural networks, e.g. as the battery drains, sacrificing accuracy for compute savings by increasing the number of early exiting examples.

Efficient supernetwork adaptation for neural architecture search
In IMB-NAS, we adapt a NAS super-net trained on balanced dataset onto an imbalanced one. The main reason being that the performance on a balanced dataset is not indicative of that on an imbalanced dataset. Future directions may involve developing efficient super-

net adaptation procedures for target dataset with domain shift beyond class imbalance, e.g., weather conditions in case of image datasets, writing styles in case of NLP, accented speech in case of speech processing.

Regression constrained architecture design
Current state-of-the-art architecture design strategies develop models for diverse platforms independant of each other. This leads to considerable negative flips: samples that are predicted correctly by a small model, but incorrectly by a larger and more accurate one. Our work (REG-NAS) mitigates this issue by sampling sub-networks suited to different platforms from the same super-network. An important future extension involves developing super-net training strategies that explicitly incorporate the notion of negative flips between sub-networks.

8.3 Conclusion

My work contributes novel frameworks and methods that jointly tackle the challenges of efficiency and robustness of deep neural networks. This advances the deployment frontier of deep learning models to resource constrained edge platforms that are truly deployed in the wild, where they are susceptible to many kinds of real-world noise.

Appendices

APPENDIX A
COMPATIBILITY AWARE HETEROGENEOUS VISUAL SEARCH

A.1 Implementation Details

A.1.1 Training, Validation and Testing Dataset

For searching the best query architecture, we carve out a small validation split from the original training set. For the face tasks, we set aside 5% from the training set of IMDB [115] while for the fashion tasks, we set aside 10% from the training set of DeepFashion2 [117]. The remaining portions of the training sets are actually used to train all our embedding models (query supernet, gallery model, final query models). After a super-network is trained, we evaluate the performance of each candidate architecture (we refer to it as a sub-network) on the held out validation split. The final results presented in this paper are reported on the original validation portions of IMDB and DeepFashion2.

A.1.2 Designing and Training the Super-network

For each computational tier (330, 230, 100 Mflops), we train a different super-network. For the 300 Mflops tier, our super-network is the same as that in [91]. For the 230 and 100 Mflops tiers, we reduce the channel widths by $0.75\times$ and $0.5\times$ in each layer. The super-network is trained through a sampling process: In each batch, a new architecture (we call this a sub-network) is sampled and only the weights corresponding to it are updated. For sampling a sub-network, we use the parameter free *uniform sampling* method. This means that, for each layer, the chosen block (includes four choices from 0-3) and channels width (includes ten choices from 0-9) are sampled uniformly. We notice that the super-network fails to converge if the sampling process is started from the first epoch. To solve this, we use a warm-up phase of 10 epochs wherein the the super-network is trained without sampling. During the warm-up phase, the output of all four blocks in each layer are combined through averaging and the largest channel width is used.

A.1.3 Details of the evolutionary search

We reuse the same hyper-parameters from [91] for the evolutionary search step. Specifically, we search for 20 generations, each with a population size of 50, crossover size of 40, mutate chance of 0.1 and random select chance of 0.1. To guide the evolutionary search for finding the most compatible architectures, we use reward $\mathcal{R}_3$ from Tab. 4.1.

For the face tasks, we compute this reward on the IMDB "validation" split using the 1:1 verification metric of TAR@FAR=10^{-3}. For the fashion tasks, we compute this reward on the DeepFashion2 validation split using the top-50 metric. Note that our rewards metrics (TAR@FAR=10^{-3} for face, top-50 for fashion) are different from the target metrics (TAR@FAR=10^{-4}/TNIR@FPIR=10^{-1} for face, top-10 for fashion). This is mainly because the validation split is smaller (than the test split), and thus target metric (*e.g.*, top-10 accuracy) is noisy compared to the validation metric (*e.g.*, top-50 accuracy).

A.2 Additional Results under Different Evaluation Metrics

Due to space limits, previously we present one evaluation metric per task. In this section, we present the full metric results according to IJB-series and DeepFashion2 benchmark standard for reference. More specifically, in Sec. A.2.1 we show top-k search accuracy on face retrieval task. In Sec A.2.2, we evaluate our CMP-NAS on face verification task at additional operating points; In Sec A.2.3, we show the results of the proposed method using top-1, top-10 and top-20 retrieval accuracy on fashion retrieval task. All these additional results further demonstrate that (1) With CMP-NAS, the compatibility rule holds; (2) The architectures searched with CMP-NAS outperform other baselines for both homogeneous and heterogeneous search accuracy.

A.2.1 Additional Results on Face Retrieval

Tab. A.1 extends Tab. 4.5 by including other popular metrics (top-1, top-5 and top-10) for the face retrieval task. Additionally, we include the homogeneous accuracy achieved by the models.

A.2.2 Additional Results on Face Verification

Besides face retrieval, face verification is another popular task in the "open-universal" problem of face recognition. in Tab. A.2, we extend Tab. 4.6 by showing the results on additional operating points (FAR=$10^{-2}, 10^{-3}, 10^{-4}$).

A.2.3 Additional Results on Fashion Retrieval

Tab. A.3 extends Tab. 4.5 by showing the homogeneous and heterogeneous accuracy through the top-1, top-10 and top-20 metrics.

Table A.1: Extending Tab. 4.5. Evaluating CMP-NAS on the IJB-C 1:N face retrieval benchmark using two additional metrics: top-1, top-5 top-10 accuracy. Observe that the models discovered with CMP-NAS comprehensively outperform the baselines on both, homogeneous and heterogeneous accuracy.

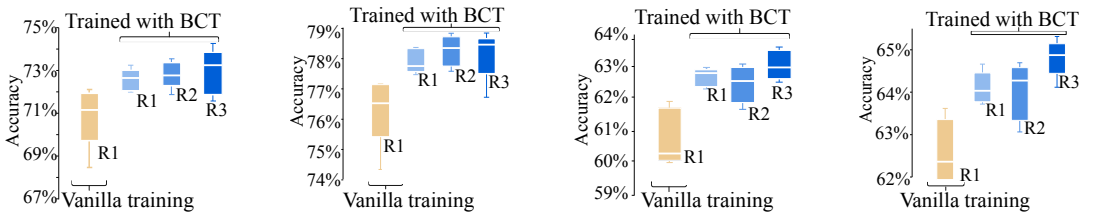
Query Model	MFlops	Homogeneous Acc. Top-k with k=			Heterogeneous Acc Top-k with k=		
		1	5	10	1	5	10
ResNet-101	7597	91.1	95.0	96.1	-	-	-
MobileNetV1	579	80.0	88.9	91.5	83.5	91.4	93.7
MobileNetV2	329	85.8	92.2	94.2	88.1	93.8	95.2
ProxylessNAS	332	86.3	92.5	94.4	88.5	93.9	95.4
CMP-NAS-a(Face)	**327**	**89.7**	**94.2**	**95.5**	**90.7**	**94.7**	**96.1**
MobileNetV3	226	85.6	92.1	94.0	88.0	93.5	95.2
CMP-NAS-b(Face)	**216**	**88.2**	**93.5**	**95.2**	**89.8**	**94.5**	**95.9**
MobileNetV1(0.5x)	155	74.1	77.5	85.3	77.5	88.3	91.3
ShuffleNetV2	149	81.6	89.8	92.2	85.0	92.0	94.1
ShuffleNetV1(g=1)	148	81.3	89.7	92.1	85.1	92.1	94.0
MobileNetV2(0.5x)	100	80.0	88.5	91.3	83.6	90.9	93.3
CMP-NAS-c(Face)	**94**	**84.3**	**91.4**	**93.4**	**86.9**	**93.1**	**94.9**

Table A.2: Extending Tab. 4.6. Evaluating the models CMP-NAS-a,b,c(Face) on the 1:1 face verification task using IJB-C using additional operating points. The searched models outperform the baselines indicating they can generalize across tasks.

Query Model	MFlops	Homogeneous Acc. TAR@FAR=			Heterogeneous Acc. TAR@FAR=		
		10^{-2}	10^{-3}	10^{-4}	10^{-2}	10^{-3}	10^{-4}
Resnet-101(gallery)	7597	96.9	92.8	85.4	-	-	-
MobileNetV1(1x)	579	93.2	82.6	66.7	95.0	86.6	73.0
MobileNetV2(1x)	329	95.6	88.1	75.4	96.5	91.0	80.8
ProxyLess(mobile)	332	95.7	88.2	75.5	96.5	90.7	80.3
CMP-NAS-a(Face)	**327**	**96.7**	**91.5**	**81.6**	**97.1**	**92.7**	**84.5**
MobileNetV3	226	95.5	88.0	74.3	96.5	90.9	79.9
CMP-NAS-b(Face)	**216**	**96.3**	**90.2**	**79.0**	**96.9**	**92.2**	**82.8**
MobileNetV1(0.5x)	155	90.8	76.9	58.0	93.4	82.1	64.3
ShuffleNetV2(1x)	149	93.7	83.8	66.8	95.4	88.7	74.8
MobileNetV2(0.5x)	100	93.3	82.0	64.8	94.9	86.8	72.8
CMP-NAS-c(Face)	**94**	**95.1**	**86.6**	**71.5**	**96.1**	**90.2**	**78.3**

Table A.3: Extending Tab. 4.5. Evaluating CMP-NAS on the Deepfashion2 fashion retrieval benchmark using additional metrics: top-1 and top-20 accuracy. We observe that the models discovered with CMP-NAS comprehensively outperform the baselines on both, homogeneous and heterogeneous accuracies.

Query Model	MFlops	Homogeneous Acc. Top-k with k as			Heterogeneous Acc Top-k with k as		
		1	10	20	1	10	20
ResNet-101		39.4	65.1	72.0	-	-	-
MobileNetV1	579	34.7	60.5	67.8	36.3	62.3	69.1
MobileNetV2	329	32.4	58.0	65.9	33.9	60.4	67.9
ProxylessNAS	332	35.1	60.8	68.5	36.6	62.1	69.4
CMP-NAS-a(Fashion)	**314**	**39.0**	**65.4**	**72.4**	**39.3**	**65.6**	**72.5**
MobileNetV3	226	37.1	62.7	69.9	37.5	63.0	70.2
CMP-NAS-b(Fashion)	**211**	**38.2**	**64.0**	**71.2**	**38.4**	**64.9**	**72.2**
MobileNetV1(0.5x)	155	32.8	57.7	65.6	34.0	60.2	67.5
ShuffleNetV2	149	35.4	60.7	68.1	35.7	62.1	69.7
ShuffleNetV1(g=1)	148	34.4	60.5	68.1	35.3	62.6	69.8
CMP-NAS-c(Fashion)	**93**	**37.6**	**63.5**	**71.0**	**38.4**	**64.8**	**72.1**



(a) Homogeneous (Face) **(b)** Heterogeneous (Face) **(c)** Homogeneous (Fashion) **(d)** Heterogeneous (Fashion)

Figure A.1: Extending Fig. 4.6. The figures are generated by averaging the best five architectures discovered by CMP-NAS (under 100 Mflops) when using different training strategies (Vanilla, BCT) and rewards ($\mathcal{R}_1 - \mathcal{R}_3$). In (a),(b) we plot the homogeneous and heterogeneous accuracy for the 1:N face retrieval task using the metric TNIR@FPIR=10^{-1}. In (c),(d) we plot the homogeneous and heterogeneous accuracy for the fashion retrieval task using the metric top-10. Observe that in all cases, BCT training works best among the training strategies while $\mathcal{R}_3$ outperforms all other rewards.

A.3 Additional results for weight-level compatibility

Due to space limit, Tab. 4.4 compares different training methods for weight-level compatibility using query model achieved by pruning 90% of filters. Tab. A.4 extends Tab. 4.4 by showing heterogeneous accuracy of query models achieved by pruning the gallery model to different levels.

Table A.4: Extending Tab. 4.4. Comparing training methods for heterogeneous accuracy achieved on the 1:N face retrieval task. The query model ϕ_q is obtained via pruning filters from the first two layers of each residual block of the gallery model. We compare two different pruning methods [20, 17] at several pruning amounts. Observe that for all pruning methods and amounts, training the query model with BCT loss leads to (1) non-zero heterogeneous accuracy and (2) the highest heterogeneous accuracy.

Gallery model	Query Prune method	Prune Amt.	Train Scratch	Fine-tune	BCT	KD
ResNet-101	-	0%	87.9	-	-	-
ResNet-101	Magnitude [17]	30%	0.0	87.9	**88.5**	0.0
ResNet-101	Magnitude [17]	50%	0.0	87.3	**88.2**	0.0
ResNet-101	Magnitude [17]	70%	0.0	87.2	**87.9**	0.0
ResNet-101	Magnitude [17]	90%	0.0	86.5	**87.2**	0.0
ResNet-101	Channel [20]	30%	0.0	87.6	**88.4**	0.0
ResNet-101	Channel [20]	50%	0.0	87.5	**87.8**	0.0
ResNet-101	Channel [20]	70%	0.0	87.3	**87.9**	0.0
ResNet-101	Channel [20]	90%	0.0	86.3	**87.4**	0.0

A.4 Comparing different rewards

Fig. A.1 is an extension of Fig. 4.6. We present the homogeneous and heterogeneous accuracy achieved by the best five query models searched using different rewards and training schemes on the face and fashion retrieval tasks. These complementary results further reinforce our conclusions: $\mathcal{R}_3$ generally works better than $\mathcal{R}_1$ and $\mathcal{R}_2$; Training the supernetwork with BCT outperforms vanilla training by a large margin.

APPENDIX B
HARDNESS AWARE REWEIGHTING FOR IMBALANCED DATASETS

B.1 Dataset Construction

We follow prior work [43, 58, 59, 60] for constructing the datasets. Specifically, the train split is subsampled as follows, while the val/test split is left class-balanced.

CIFAR LT datasets. Following [43], the train sets for CIFAR-10 LT, CIFAR-100 LT are sampled from the original training sets of CIFAR-10 and CIFAR-100 according to the exponential distribution $n_c = n\mu^c$. Here n_c refers to the remaining number of examples in class c, n is the original number of examples per class (5000 for CIFAR-10 and 500 for CIFAR-100) and $\mu \in [0, 1]$. We select μ such that the imbalance ratio—which is defined as the ratio between the number of examples in the largest and smallest class—is $10\times, 50\times, 100\times$.

ImageNet LT dataset. Following [148], the training set is subsampled from the original ImageNet training set by following the pareto distribution with $\alpha = 6$. The val split is the same as the original ImageNet dataset.

iNaturalist'18 dataset [157]. This is a naturally imbalanced dataset consisting of images from 8,142 species. The validation and test splits are balanced across classes.

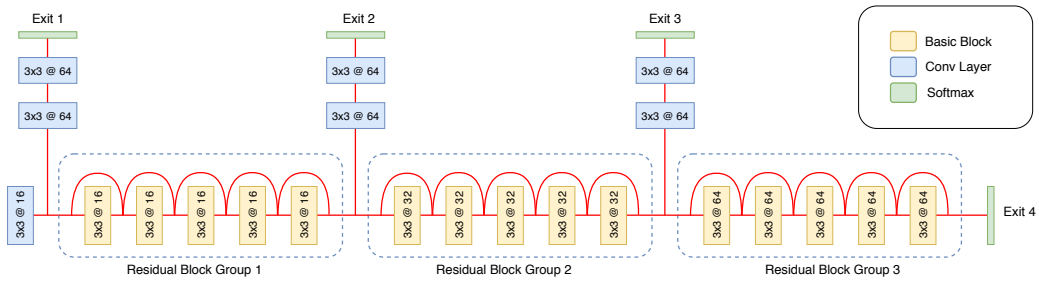
B.2 Architecture of HAR models

HAR attaches an auxiliary exit before each residual/dense block. The augmented models are shown in Fig. B.1, with the auxiliary exit design considerations discussed below.

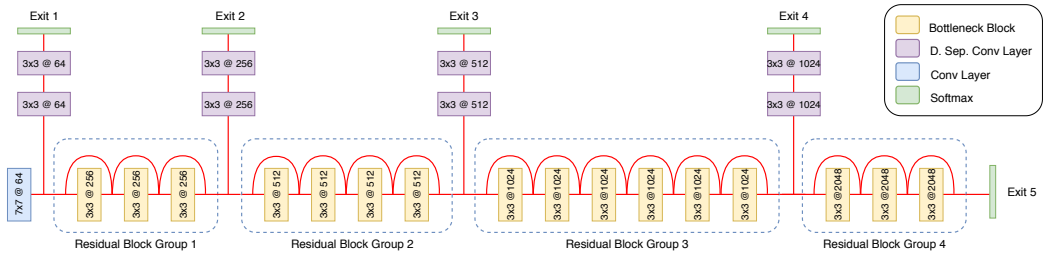
ResNet-32: This backbone model contains three residual block groups (see Fig. B.1a), with each group containing five standard "basic blocks". Each auxiliary exit consists of two convolution layers with sixty-four kernels of size 3×3, followed by an average pooling and dense layer.

ResNet-50: This backbone model contains four residual block groups (see Fig. B.1b), with the groups containing $3, 4, 6$ and 3 "bottleneck" blocks respectively. Since the filter channels increase rapidly in this architecture (e.g. group three has 1024 channels), we use depth-wise separable convolution layers at each exit which helps reduce the additional FLOPS introduced by the auxiliary exits.

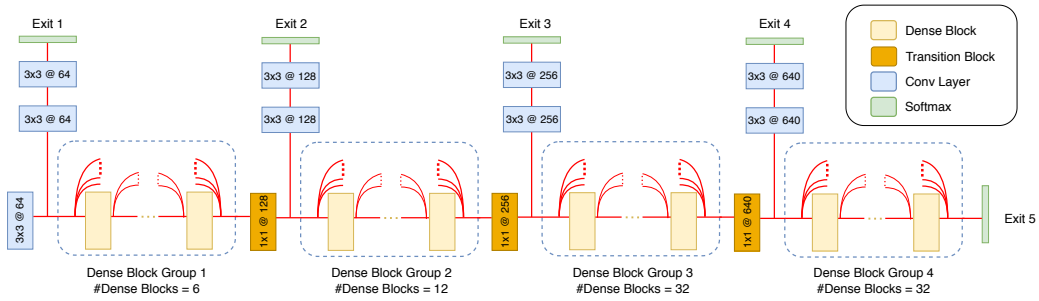
DenseNet-169 This backbone model contains four dense block groups (see Fig. B.1c), with the groups containing $6, 12, 32$ and 32 "dense" blocks respectively. Each auxiliary



(a) ResNet-32 model with early exits.



(b) ResNet-50 model with early exits.



(c) DenseNet-169 model with early exits.

Figure B.1: HAR augments a backbone model with auxilliary exits. This figure describes the configuration of the early exits for the three models considered in this work. The notation $3 \times 3@16$ indicates that the block / layer contains 16 kernels of size 3×3.

exit consists of two convolution layers with 3×3 kernels followed by an average pooling and dense layer.

B.3 Additional Results on DenseNet-169

In Tab. B.1, we present the top-1 accuracy acheived by a DenseNet-169 model trained on ImageNet LT with various loss functions. Similar to our findings with ResNet-50 (see Tab. 6.3 in the main paper), we observe that models trained with HAR(LDAM) improve more than 2.5% on top-1 accuracy while consuming fewer FLOPS. Moreover, the accuracy improvement is acheived on the three class splits corresponding to the *Many*, *Med* and *Few* shot settings.

Table B.1: HAR leads to the highest top-1 accuracy (with compute savings) for DenseNet-169 trained on Imagenet-LT and iNaturalist'18 datasets. We consider the many-, medium-, and few-shot splits. The top panel documents recent SOTA approaches. The middle and bottom panels compare the HAR method fitted with two different exit-loss types (CE/LDAM).

	Imagenet LT				iNaturalist '18			
	Many	Med	Few	All	Many	Med	Few	All
CE	63.5	38.1	14.4	44.6	73.9	64.6	58.1	63.0
CE+DRW [43]	60.3	44.2	25.8	47.9	68.9	67.3	65.6	66.8
Focal+DRW [45]	59.8	44.1	26.0	47.7	68.3	66.4	63.6	65.5
LDAM+DRW [58]	62.2	44.1	27.6	48.8	68.7	67.9	**66.5**	67.5
HAR(LDAM) + DRW (Our)	**63.8**	**46.5**	**29.0**	**50.8**	**71.5**	**68.5**	66.2	**67.9**
Flops saving				1.7%				1.4%

Table B.2: Ablating on the number of early-exits for a ResNet-50 model trained on ImageNet-LT with HAR(LDAM)+DRW. The configuration containing four exits—CEBEBEBEBE—outperforms all others.

# early-exits	Config	Many	Med	Few	All
0	CBBBBE	61.1	44.7	28.0	48.8
1	CEBBBBE	62.3	44.9	25.9	49.0
	CBEBBBE	63.0	45.9	26.7	49.9
	CBBEBBE	63.3	46.6	27.8	50.5
	CBBBEBE	62.5	46.0	27.6	49.8
	Average	62.8	46.9	27.0	49.8
2	CEBEBBBE	63.0	46.0	27.6	50.0
	CEBBEBBE	63.4	46.6	27.4	50.5
	CEBBBEBE	62.2	45.6	26.8	49.4
	CBEBEBBE	63.4	46.3	27.5	50.3
	CBEBBEBE	63.1	46.9	27.9	50.6
	Average	62.9	46.2	27.4	50.0
3	CEBEBEBBE	63.5	46.7	27.8	50.6
	CEBEBBEBE	62.6	45.7	27.7	49.8
	CEBBEBEBE	63.5	47.0	28.1	50.8
	CBEBEBEBE	63.2	46.9	28.0	50.7
	Average	63.2	46.6	28.0	50.5
4	**CEBEBEBEBE**	**63.6**	**47.4**	**28.4**	**51.1**

B.4 Additional ablation study

Q. How does the number of exits impact accuracy? To measure this, in Tab. B.2, we ablate on the number of early-exits attached to a ResNet-50 model trained on ImageNet LT using the HAR (LDAM)+DRW loss. For comparison, we use the static mode which means that the exits are only used during training and are discarded for inference. Among the

different configurations (with the naming convention: C=conv layer, B=block, E=exit), the one containing four exits—CEBEBEBEBE—outperforms all others. This validates our choice of using four branches for all HAR models. Additionally, we observe that the accuracy improves with the number of exits.

REFERENCES

[1] R. Duggal, S. Freitas, C. Xiao, D. H. Chau, and J. Sun, "Rest: Robust and efficient neural networks for sleep monitoring in the wild," in *Proceedings of The Web Conference 2020*, 2020, pp. 1704–1714.

[2] M. Ye, J. Shen, G. Lin, T. Xiang, L. Shao, and S. C. Hoi, "Deep learning for person re-identification: A survey and outlook," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2021.

[3] N. S. Samghabadi, P. Patwa, P. Srinivas, P. Mukherjee, A. Das, and T. Solorio, "Aggression and misogyny detection using bert: A multi-task approach," in *Proceedings of the Second Workshop on Trolling, Aggression and Cyberbullying*, 2020, pp. 126–131.

[4] E. Strubell, A. Ganesh, and A. McCallum, "Energy and policy considerations for deep learning in nlp," *arXiv preprint arXiv:1906.02243*, 2019.

[5] ——, "Energy and policy considerations for modern deep learning research," in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 34, 2020, pp. 13 693–13 696.

[6] C. Zhang, S. Bengio, M. Hardt, B. Recht, and O. Vinyals, "Understanding deep learning (still) requires rethinking generalization," *Communications of the ACM*, vol. 64, no. 3, pp. 107–115, 2021.

[7] A. Sors, S. Bonnet, S. Mirek, L. Vercueil, and J.-F. Payen, "A convolutional neural network for sleep stage scoring from raw single-channel eeg," *Biomedical Signal Processing and Control*, vol. 42, pp. 107 –114, 2018.

[8] S. Han, J. Pool, J. Tran, and W. Dally, "Learning both weights and connections for efficient neural network," in *Neurips*, 2015, pp. 1135–1143.

[9] B. Zoph and Q. V. Le, "Neural architecture search with reinforcement learning," in *ICLR*, 2017.

[10] M. Rastegari, V. Ordonez, J. Redmon, and A. Farhadi, "Xnor-net: Imagenet classification using binary convolutional neural networks," in *European Conference on Computer Vision*, Springer, 2016, pp. 525–542.

[11] J. Xue, J. Li, and Y. Gong, "Restructuring of deep neural network acoustic models with singular value decomposition.," in *Interspeech*, 2013, pp. 2365–2369.

[12] G. Hinton, O. Vinyals, and J. Dean, "Distilling the knowledge in a neural network," *arXiv preprint arXiv:1503.02531*, 2015.

[13] Y. LeCun, J. Denker, and S. Solla, "Optimal brain damage," *Neurips*, vol. 2, pp. 598–605, 1989.

[14] S. Han, H. Mao, and W. J. Dally, "Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding," *arXiv preprint arXiv:1510.00149*, 2015.

[15] D. Molchanov, A. Ashukha, and D. Vetrov, "Variational dropout sparsifies deep neural networks," *arXiv preprint arXiv:1701.05369*, 2017.

[16] C. Louizos, M. Welling, and D. P. Kingma, "Learning sparse neural networks through l0 regularization," *arXiv preprint arXiv:1712.01312*, 2017.

[17] H. Li, A. Kadav, I. Durdanovic, H. Samet, and H. P. Graf, "Pruning filters for efficient convnets," in *ICLR*, 2017.

[18] H. Hu, R. Peng, Y.-W. Tai, and C.-K. Tang, "Network trimming: A data-driven neuron pruning approach towards efficient deep architectures," *arXiv preprint arXiv:1607.03250*, 2016.

[19] Z. Liu, J. Li, Z. Shen, G. Huang, S. Yan, and C. Zhang, "Learning efficient convolutional networks through network slimming," in *ICCV*, 2017.

[20] Y. He, X. Zhang, and J. Sun, "Channel pruning for accelerating very deep neural networks," in *ICCV*, 2017.

[21] Z. Zhuang, M. Tan, B. Zhuang, J. Liu, Y. Guo, Q. Wu, J. Huang, and J. Zhu, "Discrimination-aware channel pruning for deep neural networks," in *Neurips*, 2018, pp. 883–894.

[22] W. Wen, C. Wu, Y. Wang, Y. Chen, and H. Li, "Learning structured sparsity in deep neural networks," in *Neurips*, 2016, pp. 2074–2082.

[23] B. Zoph, V. Vasudevan, J. Shlens, and Q. V. Le, "Learning transferable architectures for scalable image recognition," in *CVPR*, 2018.

[24] H. Liu, K. Simonyan, and Y. Yang, "Darts: Differentiable architecture search," in *ICLR*, 2019.

[25] H. Cai, C. Gan, T. Wang, Z. Zhang, and S. Han, "Once for all: Train one network and specialize it for efficient deployment," in *International Conference on Learning Representations*, 2020.

[26] D. Wang, M. Li, C. Gong, and V. Chandra, "Attentivenas: Improving neural architecture search via attentive sampling," *arXiv preprint arXiv:2011.09011*, 2020.

[27] I. Radosavovic, R. P. Kosaraju, R. Girshick, K. He, and P. Dollár, "Designing network design spaces," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2020, pp. 10 428–10 436.

[28] E. Real, A. Aggarwal, Y. Huang, and Q. V. Le, "Regularized evolution for image classifier architecture search," in *AAAI*, 2019.

[29] A. Klein, S. Falkner, J. T. Springenberg, and F. Hutter, "Learning curve prediction with bayesian neural networks," 2016.

[30] G. Huang, D. Chen, T. Li, F. Wu, L. van der Maaten, and K. Q. Weinberger, "Multi-scale dense networks for resource efficient image classification," in *ICLR*, 2018.

[31] M. Phuong and C. H. Lampert, "Distillation-based training for multi-exit architectures," in *ICCV*, 2019.

[32] T.-K. Hu, T. Chen, H. Wang, and Z. Wang, "Triple wins: Boosting accuracy, robustness and efficiency together by enabling input-adaptive inference," *arXiv preprint arXiv:2002.10025*, 2020.

[33] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich, "Going deeper with convolutions," in *CVPR*, 2015.

[34] E. Baccarelli, S. Scardapane, M. Scarpiniti, A. Momenzadeh, and A. Uncini, "Optimized training and scalable implementation of conditional deep neural networks with early exits for fog-supported iot applications," *Information Sciences*, 2020.

[35] F. Nan and V. Saligrama, "Adaptive classification for prediction under a budget," *arXiv preprint arXiv:1705.10194*, 2017.

[36] S. Scardapane, M. Scarpiniti, E. Baccarelli, and A. Uncini, "Why should we add early exits to neural networks?" *Cognitive Computation*, 2020.

[37] S. Teerapittayanon, B. McDanel, and H.-T. Kung, "Branchynet: Fast inference via early exiting from deep neural networks," in *ICPR*, IEEE, 2016, pp. 2464–2469.

[38] H. Li, H. Zhang, X. Qi, R. Yang, and G. Huang, "Improved techniques for training adaptive deep networks," in *ICCV*, 2019, pp. 1891–1900.

[39] I. J. Goodfellow, J. Shlens, and C. Szegedy, "Explaining and harnessing adversarial examples," *arXiv preprint arXiv:1412.6572*, 2014.

[40] N. Ford, J. Gilmer, N. Carlini, and D. Cubuk, "Adversarial examples are a natural consequence of test error in noise," *CoRR*, vol. abs/1901.10513, 2019. arXiv: 1901.10513.

[41] D. Hendrycks and T. Dietterich, "Benchmarking neural network robustness to common corruptions and perturbations," *arXiv preprint arXiv:1903.12261*, 2019.

[42] D. Tsipras, S. Santurkar, L. Engstrom, A. Turner, and A. Madry, "Robustness may be at odds with accuracy," *stat*, vol. 1050, p. 11, 2018.

[43] Y. Cui, M. Jia, T.-Y. Lin, Y. Song, and S. Belongie, "Class-balanced loss based on effective number of samples," in *CVPR*, 2019, pp. 9268–9277.

[44] C. Huang, Y. Li, C. L. Chen, and X. Tang, "Deep imbalanced learning for face recognition and attribute prediction," *TPAMI*, 2019.

[45] T.-Y. Lin, P. Goyal, R. Girshick, K. He, and P. Dollár, "Focal loss for dense object detection," in *ICCV*, 2017, pp. 2980–2988.

[46] M. Ren, W. Zeng, B. Yang, and R. Urtasun, "Learning to reweight examples for robust deep learning," in *ICML*, 2018.

[47] J. Shu, Q. Xie, L. Yi, Q. Zhao, S. Zhou, Z. Xu, and D. Meng, "Meta-weight-net: Learning an explicit mapping for sample weighting," *arXiv preprint arXiv:1902.07379*, 2019.

[48] M. A. Jamal, M. Brown, M.-H. Yang, L. Wang, and B. Gong, "Rethinking class-balanced methods for long-tailed visual recognition from a domain adaptation perspective," in *CVPR*, 2020.

[49] N. V. Chawla, K. W. Bowyer, L. O. Hall, and W. P. Kegelmeyer, "Smote: Synthetic minority over-sampling technique," *Journal of artificial intelligence research*, vol. 16, pp. 321–357, 2002.

[50] H. Han, W.-Y. Wang, and B.-H. Mao, "Borderline-smote: A new over-sampling method in imbalanced data sets learning," in *International conference on intelligent computing*, Springer, 2005, pp. 878–887.

[51] H. He, Y. Bai, E. A. Garcia, and S. Li, "Adasyn: Adaptive synthetic sampling approach for imbalanced learning," in *IJCNN*.

[52] S. Barua, M. M. Islam, X. Yao, and K. Murase, "Mwmote–majority weighted minority oversampling technique for imbalanced data set learning," *TKDE*, 2012.

[53] S.-J. Yen and Y.-S. Lee, "Cluster-based under-sampling approaches for imbalanced data distributions," *Expert Systems with Applications*, vol. 36, no. 3, pp. 5718–5727, 2009.

[54] M. Galar, A. Fernández, E. Barrenechea, and F. Herrera, "Eusboost: Enhancing ensembles for highly imbalanced data-sets by evolutionary undersampling," *Pattern recognition*, 2013.

[55] W.-C. Lin, C.-F. Tsai, Y.-H. Hu, and J.-S. Jhang, "Clustering-based undersampling in class-imbalanced data," *Information Sciences*, vol. 409, pp. 17–26, 2017.

[56] C.-F. Tsai, W.-C. Lin, Y.-H. Hu, and G.-T. Yao, "Under-sampling class imbalanced datasets by combining clustering analysis and instance selection," *Information Sciences*, vol. 477, pp. 47–54, 2019.

[57] M. Buda, A. Maki, and M. A. Mazurowski, "A systematic study of the class imbalance problem in convolutional neural networks," *Neural Networks*, vol. 106, pp. 249–259, 2018.

[58] K. Cao, C. Wei, A. Gaidon, N. Arechiga, and T. Ma, "Learning imbalanced datasets with label-distribution-aware margin loss," in *NIPS*, 2019, pp. 1565–1576.

[59] B. Kang, S. Xie, M. Rohrbach, Z. Yan, A. Gordo, J. Feng, and Y. Kalantidis, "Decoupling representation and classifier for long-tailed recognition," *ICLR*, 2020.

[60] B. Zhou, Q. Cui, X.-S. Wei, and Z.-M. Chen, "Bbn: Bilateral-branch network with cumulative learning for long-tailed visual recognition," *arXiv preprint arXiv:1912.02413*, 2019.

[61] M. Jaderberg, A. Vedaldi, and A. Zisserman, "Speeding up convolutional neural networks with low rank expansions," *arXiv preprint arXiv:1405.3866*, 2014.

[62] W. Wang, Y. Sun, B. Eriksson, W. Wang, and V. Aggarwal, "Wide compression: Tensor ring nets," *learning*, vol. 14, no. 15, pp. 13–31, 2018.

[63] E. J. Crowley, G. Gray, and A. J. Storkey, "Moonshine: Distilling with cheap convolutions," in *Neurips*, 2018.

[64] M. Courbariaux, Y. Bengio, and J.-P. David, "Binaryconnect: Training deep neural networks with binary weights during propagations," in *Neurips*, 2015, pp. 3123–3131.

[65] B. Jacob, S. Kligys, B. Chen, M. Zhu, M. Tang, A. Howard, H. Adam, and D. Kalenichenko, "Quantization and training of neural networks for efficient integer-arithmetic-only inference," in *CVPR*, 2018, pp. 2704–2713.

[66] Z. Liu, M. Sun, T. Zhou, G. Huang, and T. Darrell, "Rethinking the value of network pruning," *arXiv:1810.05270*, 2018.

[67] Y. He, J. Lin, Z. Liu, H. Wang, L.-J. Li, and S. Han, "Amc: Automl for model compression and acceleration on mobile devices," in *ECCV*, 2018, pp. 784–800.

[68] Y. He, G. Kang, X. Dong, Y. Fu, and Y. Yang, "Soft filter pruning for accelerating deep convolutional neural networks," *arXiv preprint arXiv:1808.06866*, 2018.

[69] Y. He, P. Liu, Z. Wang, Z. Hu, and Y. Yang, "Filter pruning via geometric median for deep convolutional neural networks acceleration," in *CVPR*, 2019, pp. 4340–4349.

[70] S. Lin, R. Ji, C. Yan, B. Zhang, L. Cao, Q. Ye, F. Huang, and D. Doermann, "Towards optimal structured cnn pruning via generative adversarial learning," in *CVPR*, 2019, pp. 2790–2799.

[71] C. Zhao, B. Ni, J. Zhang, Q. Zhao, W. Zhang, and Q. Tian, "Variational convolutional neural network pruning," in *CVPR*, 2019, pp. 2780–2789.

[72] S. Srinivas and R. V. Babu, "Data-free parameter pruning for deep neural networks," *arXiv preprint arXiv:1507.06149*, 2015.

[73] D. S. Wilks, *Statistical methods in the atmospheric sciences*. Academic press, 2011, vol. 100.

[74] L. Li, Y. Xu, and J. Zhu, "Filter level pruning based on similar feature extraction for convolutional neural networks," *IEICE TRANSACTIONS on Information and Systems*, vol. 101, no. 4, pp. 1203–1206, 2018.

[75] L. Li, J. Zhu, and M.-T. Sun, "A spectral clustering based filter-level pruning method for convolutional neural networks," *IEICE TRANSACTIONS on Information and Systems*, vol. 102, no. 12, pp. 2624–2627, 2019.

[76] Y. LeCun, L. Bottou, Y. Bengio, P. Haffner, *et al.*, "Gradient-based learning applied to document recognition," *Proceedings of the IEEE*, vol. 86, no. 11, pp. 2278–2324, 1998.

[77] A. Krizhevsky and G. Hinton, "Learning multiple layers of features from tiny images," Citeseer, Tech. Rep., 2009.

[78] O. Russakovsky, J. Deng, H. Su, J. Krause, S. Satheesh, S. Ma, Z. Huang, A. Karpathy, A. Khosla, M. Bernstein, *et al.*, "Imagenet large scale visual recognition challenge," *IJCV*, 2015.

[79] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," *arXiv preprint arXiv:1409.1556*, 2014.

[80] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *CVPR*, 2016, pp. 770–778.

[81] W. Wang, C. Fu, J. Guo, D. Cai, and X. He, "Cop: Customized deep model compression via regularized correlation-based filter-level pruning," *arXiv preprint arXiv:1906.10337*, 2019.

[82] Z. Liu, H. Mu, X. Zhang, Z. Guo, X. Yang, T. K.-T. Cheng, and J. Sun, "Metapruning: Meta learning for automatic neural network channel pruning," *arXiv preprint arXiv:1903.10258*, 2019.

[83] A. Qayyum, S. M. Anwar, M. Awais, and M. Majid, "Medical image retrieval using deep convolutional neural network," *Neurocomputing*, vol. 266, pp. 8–20, 2017.

[84] A. S. Razavian, J. Sullivan, S. Carlsson, and A. Maki, "Visual instance retrieval with deep convolutional networks," *TMTA*, vol. 4, no. 3, pp. 251–258, 2016.

[85] L. Xie, R. Hong, B. Zhang, and Q. Tian, "Image classification and retrieval are one," in *ICMR*, 2015.

[86] A. Babenko and V. Lempitsky, "Aggregating deep convolutional features for image retrieval," *arXiv:1510.07493*, 2015.

[87] G. Tolias, R. Sicre, and H. Jégou, "Particular object retrieval with integral max-pooling of cnn activations," *arXiv:1511.05879*, 2015.

[88] Y. Shen, Y. Xiong, W. Xia, and S. Soatto, "Towards backward-compatible representation learning," in *CVPR*, 2020.

[89] Z. Liu, M. Sun, T. Zhou, G. Huang, and T. Darrell, "Rethinking the value of network pruning," *ICLR*, 2019.

[90] H. Cai, L. Zhu, and S. Han, "ProxylessNAS: Direct neural architecture search on target task and hardware," in *ICLR*, 2019.

[91] Z. Guo, X. Zhang, H. Mu, W. Heng, Z. Liu, Y. Wei, and J. Sun, "Single path one-shot neural architecture search with uniform sampling," in *ECCV*, 2020.

[92] M. Tan, B. Chen, R. Pang, V. Vasudevan, M. Sandler, A. Howard, and Q. V. Le, "Mnasnet: Platform-aware neural architecture search for mobile," in *ICLR*, 2019.

[93] C. Wengert, M. Douze, and H. Jégou, "Bag-of-colors for improved image search," in *ACM Multimedia*, 2011.

[94] M. Park, J. S. Jin, and L. S. Wilson, "Fast content-based image retrieval using quasi-gabor filter and reduction of image feature dimension," in *Proceedings fifth IEEE southwest symposium on image analysis and interpretation*, 2002.

[95] C. Siagian and L. Itti, "Rapid biologically-inspired scene classification using features shared with visual attention," *TPAMI*, vol. 29, no. 2, pp. 300–312, 2007.

[96] R. Arandjelović and A. Zisserman, "Three things everyone should know to improve object retrieval," in *CVPR*, 2012.

[97] L. Zheng, S. Wang, Z. Liu, and Q. Tian, "Packing and padding: Coupled multi-index for accurate image retrieval," in *CVPR*, 2014.

[98] W. Zhou, Y. Lu, H. Li, and Q. Tian, "Scalar quantization for large scale image search," in *ACM Multimedia*, 2012.

[99] L. Zheng, S. Wang, L. Tian, F. He, Z. Liu, and Q. Tian, "Query-adaptive late fusion for image search and person re-identification," in *CVPR*, 2015.

[100] J. R. Uijlings, K. E. Van De Sande, T. Gevers, and A. W. Smeulders, "Selective search for object recognition," *IJCV*, vol. 104, no. 2, pp. 154–171, 2013.

[101] Y. Kalantidis, C. Mellina, and S. Osindero, "Cross-dimensional weighting for aggregated deep convolutional features," in *ECCV*, 2016.

[102] F. Radenović, G. Tolias, and O. Chum, "Fine-tuning cnn image retrieval with no human annotation," *TPAMI*, vol. 41, no. 7, pp. 1655–1668, 2018.

[103] M. Budnik and Y. Avrithis, "Asymmetric metric learning for knowledge transfer," *arXiv:2006.16331*, 2020.

[104] C.-Y. Wang, Y.-L. Chang, S.-T. Yang, D. Chen, and S.-H. Lai, "Unified representation learning for cross model compatibility," in *BMVC*, 2020.

[105] K. Chen, Y. Wu, H. Qin, D. Liang, X. Liu, and J. Yan, "R3 adversarial network for cross model face recognition," in *CVPR*, 2019.

[106] J. Hu, R. Ji, H. Liu, S. Zhang, C. Deng, and Q. Tian, "Towards visual feature translation," in *CVPR*, 2019.

[107] Y. Liu, X. Jia, M. Tan, R. Vemulapalli, Y. Zhu, B. Green, and X. Wang, "Search to distill: Pearls are everywhere but not the eyes," in *CVPR*, 2020.

[108] C. Li, J. Peng, L. Yuan, G. Wang, X. Liang, L. Lin, and X. Chang, "Block-wisely supervised neural architecture search with knowledge distillation," in *CVPR*, 2020.

[109] M. Sandler, A. Howard, M. Zhu, A. Zhmoginov, and L. Chen, "Mobilenetv2: Inverted residuals and linear bottlenecks," in *CVPR*, 2018.

[110] A. Zhai and H.-Y. Wu, "Classification is a strong baseline for deep metric learning," in *BMVC*, 2019.

[111] H. Wang, Y. Wang, Z. Zhou, X. Ji, D. Gong, J. Zhou, Z. Li, and W. Liu, "Cosface: Large margin cosine loss for deep face recognition," in *CVPR*, 2018.

[112] F. Wang, X. Xiang, J. Cheng, and A. L. Yuille, "Normface: L2 hypersphere embedding for face verification," in *ACM Multimedia*, 2017.

[113] N. Ma, X. Zhang, H.-T. Zheng, and J. Sun, "Shufflenet v2: Practical guidelines for efficient cnn architecture design," in *ECCV*, 2018.

[114] F. Chollet, "Xception: Deep learning with depthwise separable convolutions," in *CVPR*, 2017.

[115] F. Wang, L. Chen, C. Li, S. Huang, Y. Chen, C. Qian, and C. C. Loy, "The devil of face recognition is in the noise," in *ECCV*, 2018.

[116] B. Maze, J. Adams, J. A. Duncan, N. Kalka, T. Miller, C. Otto, A. K. Jain, W. T. Niggel, J. Anderson, J. Cheney, and P. Grother, "Iarpa janus benchmark - c: Face dataset and protocol," in *ICB*, 2018.

[117] Y. Ge, R. Zhang, X. Wang, X. Tang, and P. Luo, "Deepfashion2: A versatile benchmark for detection, pose estimation, segmentation and re-identification of clothing images," in *CVPR*, 2019.

[118] I. Loshchilov and F. Hutter, "Sgdr: Stochastic gradient descent with warm restarts," in *ICLR*, 2017.

[119] A. Howard, M. Sandler, G. Chu, L.-C. Chen, B. Chen, M. Tan, W. Wang, Y. Zhu, R. Pang, V. Vasudevan, Q. V. Le, and H. Adam, "Searching for mobilenetv3," in *ICCV*, 2019.

[120] B. M. Altevogt, H. R. Colten, *et al.*, *Sleep disorders and sleep deprivation: an unmet public health problem*. National Academies Press, 2006.

[121] A. A. of Sleep Medicine *et al.*, "Economic burden of undiagnosed sleep apnea in us is nearly $150 billion per year," *Published on the American Academy of Sleep Medicine's official website, on August*, vol. 8, 2016.

[122] S. Biswal, J. Kulas, H. Sun, B. Goparaju, M. B. Westover, M. T. Bianchi, and J. Sun, "SLEEPNET: automated sleep staging system via deep learning," *CoRR*, vol. abs/1707.08262, 2017. arXiv: 1707.08262.

[123] A. Supratak, H. Dong, C. Wu, and Y. Guo, "Deepsleepnet: A model for automatic sleep stage scoring based on raw single-channel eeg," *IEEE Transactions on Neural Systems and Rehabilitation Engineering*, vol. 25, no. 11, pp. 1998–2008, 2017.

[124] A. Sterr, J. K. Ebajemito, K. B. Mikkelsen, M. A. Bonmati-Carrion, N. Santhi, C. Della Monica, L. Grainger, G. Atzori, V. Revell, S. Debener, *et al.*, "Sleep eeg derived from behind-the-ear electrodes (ceegrid) compared to standard polysomnography: A proof of concept study," *Frontiers in human neuroscience*, vol. 12, p. 452, 2018.

[125] A. Henriksen, M. H. Mikalsen, A. Z. Woldaregay, M. Muzny, G. Hartvigsen, L. A. Hopstock, and S. Grimsgaard, "Using fitness trackers and smartwatches to measure physical activity in research: Analysis of consumer wrist-worn wearables," *Journal of medical Internet research*, vol. 20, no. 3, e110, 2018.

[126] Z Beattie, A Pantelopoulos, A Ghoreyshi, Y Oyang, A Statan, and C Heneghan, "0068 ESTIMATION OF SLEEP STAGES USING CARDIAC AND ACCELEROMETER DATA FROM a WRIST-WORN DEVICE," *Sleep*, vol. 40, no. suppl_1, A26–A26, Apr. 2017.

[127] K.-M. Chang and S.-H. Liu, "Gaussian noise filtering from ecg by wiener filter and ensemble empirical mode decomposition," *Journal of Signal Processing Systems*, vol. 64, no. 2, pp. 249–264, 2011.

[128] M. Blanco-Velasco, B. Weng, and K. E. Barner, "Ecg signal denoising and baseline wander correction based on the empirical mode decomposition," *Computers in biology and medicine*, vol. 38, no. 1, pp. 1–13, 2008.

[129] Y. Chen, M. Akutagawa, T. Emoto, and Y. Kinouchi, "The removal of emg in eeg by neural networks," *Physiological measurement*, vol. 31, no. 12, p. 1567, 2010.

[130] V. Bhateja, S. Urooj, R. Verma, and R. Mehrotra, "A novel approach for suppression of powerline interference and impulse noise in ecg signals," in *IMPACT-2013*, IEEE, 2013, pp. 103–107.

[131] S. Chambon, M. N. Galtier, P. J. Arnal, G. Wainrib, and A. Gramfort, "A deep learning architecture for temporal sleep stage classification using multivariate and multimodal time series," *IEEE Transactions on Neural Systems and Rehabilitation Engineering*, vol. 26, no. 4, pp. 758–769, 2018.

[132] H. Phan, F. Andreotti, N. Cooray, O. Y. Chén, and M. De Vos, "Joint classification and prediction cnn framework for automatic sleep stage classification," *IEEE Transactions on Biomedical Engineering*, vol. 66, no. 5, pp. 1285–1296, 2019.

[133] F. Andreotti, H. Phan, N. Cooray, C. Lo, M. T. M. Hu, and M. De Vos, "Multichannel sleep stage classification and transfer learning using convolutional neural networks," in *2018 40th Annual International Conference of the IEEE Engineering in Medicine and Biology Society (EMBC)*, 2018, pp. 171–174.

[134] M. Zhao, S. Yue, D. Katabi, T. S. Jaakkola, and M. T. Bianchi, "Learning sleep stages from radio signals: A conditional adversarial architecture," Proceedings of Machine Learning Research, vol. 70, D. Precup and Y. W. Teh, Eds., pp. 4100–4109, 2017.

[135] R. Duggal, C. Xiao, R. Vuduc, and J. Sun, *Cup: Cluster pruning for compressing deep neural networks*, 2019. eprint: `arXiv:1911.08630`.

[136] Y. Guo, C. Zhang, C. Zhang, and Y. Chen, "Sparse dnns with improved adversarial robustness," in *Advances in neural information processing systems*, 2018, pp. 242–251.

[137] J. Lin, C. Gan, and S. Han, "Defensive quantization: When efficiency meets robustness," *arXiv preprint arXiv:1904.08444*, 2019.

[138] A. Madry, A. Makelov, L. Schmidt, D. Tsipras, and A. Vladu, "Towards deep learning models resistant to adversarial attacks," *arXiv preprint arXiv:1706.06083*, 2017.

[139] M. Cisse, P. Bojanowski, E. Grave, Y. Dauphin, and N. Usunier, "Parseval networks: Improving robustness to adversarial examples," in *Proceedings of the 34th International Conference on Machine Learning-Volume 70*, JMLR. org, 2017, pp. 854–863.

[140] Y. Yoshida and T. Miyato, "Spectral norm regularization for improving the generalizability of deep learning," *arXiv preprint arXiv:1705.10941*, 2017.

[141] F. Farnia, J. M. Zhang, and D. Tse, "Generalizable adversarial training via spectral normalization," *arXiv preprint arXiv:1811.07457*, 2018.

[142] V. Lebedev and V. Lempitsky, "Fast convnets using group-wise brain damage," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2016, pp. 2554–2564.

[143] A. L. Goldberger, L. A. Amaral, L. Glass, J. M. Hausdorff, P. C. Ivanov, R. G. Mark, J. E. Mietus, G. B. Moody, C.-K. Peng, and H. E. Stanley, "Physiobank,

physiotoolkit, and physionet: Components of a new research resource for complex physiologic signals," *Circulation*, vol. 101, no. 23, e215–e220, 2000.

[144] S. F. Quan, B. V. Howard, C. Iber, J. P. Kiley, F. J. Nieto, G. T. O'Connor, D. M. Rapoport, S. Redline, J. Robbins, J. M. Samet, *et al.*, "The sleep heart health study: Design, rationale, and methods," *Sleep*, vol. 20, no. 12, pp. 1077–1085, 1997.

[145] R. B. Berry, R. Brooks, C. E. Gamaldo, S. M. Harding, C. L. Marcus, B. V. Vaughn, *et al.*, "The aasm manual for the scoring of sleep and associated events," *Rules, Terminology and Technical Specifications, Darien, Illinois, American Academy of Sleep Medicine*, vol. 176, 2012.

[146] S Blanco, S Kochen, O. Rosso, and P Salgado, "Applying time-frequency analysis to seizure eeg activity," *IEEE Engineering in medicine and biology magazine*, vol. 16, no. 1, pp. 64–71, 1997.

[147] M. A. Mazurowski, P. A. Habas, J. M. Zurada, J. Y. Lo, J. A. Baker, and G. D. Tourassi, "Training neural network classifiers for medical decision making: The effects of imbalanced datasets on classification performance," *Neural networks*, 2008.

[148] Z. Liu, Z. Miao, X. Zhan, J. Wang, B. Gong, and S. X. Yu, "Large-scale long-tailed recognition in an open world," in *CVPR*, 2019.

[149] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei, "Imagenet: A large-scale hierarchical image database," in *CVPR*, 2009.

[150] P.-H. C. Chen, Y. Liu, and L. Peng, "How to develop machine learning models for healthcare," *Nature materials*, 2019.

[151] C. Huang, Y. Li, C. Change Loy, and X. Tang, "Learning deep representation for imbalanced classification," in *CVPR*, 2016.

[152] Y.-X. Wang, D. Ramanan, and M. Hebert, "Learning to model the tail," in *NIPS*, 2017, pp. 7029–7039.

[153] D. Mahajan, R. Girshick, V. Ramanathan, K. He, M. Paluri, Y. Li, A. Bharambe, and L. van der Maaten, "Exploring the limits of weakly supervised pretraining," in *ECCV*, 2018, pp. 181–196.

[154] Q. Dong, S. Gong, and X. Zhu, "Class rectification hard mining for imbalanced deep learning," in *ICCV*, 2017.

[155] H.-S. Chang, E. Learned-Miller, and A. McCallum, "Active bias: Training more accurate neural networks by emphasizing high variance samples," *arXiv preprint arXiv:1704.07433*, 2017.

[156] N. V. Chawla, A. Lazarevic, L. O. Hall, and K. W. Bowyer, "Smoteboost: Improving prediction of the minority class in boosting," in *European conference on principles of data mining and knowledge discovery*, Springer, 2003, pp. 107–119.

[157] G. Van Horn, O. Mac Aodha, Y. Song, Y. Cui, C. Sun, A. Shepard, H. Adam, P. Perona, and S. Belongie, "The inaturalist species classification and detection dataset," in *CVPR*, 2018.

[158] Y. Bengio, J. Louradour, R. Collobert, and J. Weston, "Curriculum learning," in *ICML*, 2009.

[159] G. Hacohen and D. Weinshall, "On the power of curriculum learning in training deep networks," in *ICML*, 2019.

[160] A. Shrivastava, A. Gupta, and R. Girshick, "Training region-based object detectors with online hard example mining," in *CVPR*, 2016.

[161] C. Florensa, D. Held, X. Geng, and P. Abbeel, "Automatic goal generation for reinforcement learning agents," in *ICML*, 2018.

[162] T. Matiisen, A. Oliver, T. Cohen, and J. Schulman, "Teacher–student curriculum learning," *IEEE transactions on neural networks and learning systems*, vol. 31, no. 9, pp. 3732–3740, 2019.

[163] C.-Y. Lee, S. Xie, P. Gallagher, Z. Zhang, and Z. Tu, "Deeply-supervised nets," in *AISTATS*, 2015.

[164] J. Kim, J. K. Lee, and K. M. Lee, "Deeply-recursive convolutional network for image super-resolution," in *CVPR*, 2016.

[165] T. Mikolov, I. Sutskever, K. Chen, G. S. Corrado, and J. Dean, "Distributed representations of words and phrases and their compositionality," in *NIPS*, 2013, pp. 3111–3119.

[166] H. Zhang, M. Cisse, Y. N. Dauphin, and D. Lopez-Paz, "Mixup: Beyond empirical risk minimization," *arXiv preprint*, 2017.

[167] V. Verma, A. Lamb, C. Beckham, A. Najafi, I. Mitliagkas, A. Courville, D. Lopez-Paz, and Y. Bengio, "Manifold mixup: Better representations by interpolating hidden states," *arXiv preprint arXiv:1806.05236*, 2018.

[168] B. Zhou, Q. Cui, X.-S. Wei, and Z.-M. Chen, "Bbn: Bilateral-branch network with cumulative learning for long-tailed visual recognition," in *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, 2020, pp. 9719–9728.

[169] R. Duggal, S. Freitas, S. Dhamnani, D. H. Chau, and J. Sun, "Har: Hardness aware reweighting for imbalanced datasets," in *2021 IEEE International Conference on Big Data (Big Data)*, IEEE, 2021, pp. 735–745.

[170] Z. Guo, X. Zhang, H. Mu, W. Heng, Z. Liu, Y. Wei, and J. Sun, "Single path one-shot neural architecture search with uniform sampling," in *European conference on computer vision*, Springer, 2020, pp. 544–560.

[171] H. Liu, K. Simonyan, and Y. Yang, "Darts: Differentiable architecture search," *arXiv preprint arXiv:1806.09055*, 2018.

[172] Y. Zhang, B. Kang, B. Hooi, S. Yan, and J. Feng, "Deep long-tailed learning: A survey," *arXiv preprint arXiv:2110.04596*, 2021.

[173] R. Duggal, S. Freitas, S. Dhamnani, D. H. Chau, and J. Sun, "Elf: An early-exiting framework for long-tailed classification," *arXiv preprint arXiv:2006.11979*, 2020.

[174] A. K. Menon, S. Jayasumana, A. S. Rawat, H. Jain, A. Veit, and S. Kumar, "Long-tail learning via logit adjustment," *arXiv preprint arXiv:2007.07314*, 2020.

[175] J. Tian, Y.-C. Liu, N. Glaser, Y.-C. Hsu, and Z. Kira, "Posterior re-calibration for imbalanced datasets," *Advances in Neural Information Processing Systems*, vol. 33, pp. 8101–8113, 2020.

[176] S. Zhang, Z. Li, S. Yan, X. He, and J. Sun, "Distribution alignment: A unified framework for long-tail visual recognition," in *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, 2021, pp. 2361–2370.

[177] X. Yin, X. Yu, K. Sohn, X. Liu, and M. Chandraker, "Feature transfer learning for face recognition with under-represented data," in *CVPR*, 2019.

[178] P. Chu, X. Bian, S. Liu, and H. Ling, "Feature space augmentation for long-tailed data," in *European Conference on Computer Vision*, Springer, 2020, pp. 694–710.

[179] T.-Y. Wu, P. Morgado, P. Wang, C.-H. Ho, and N. Vasconcelos, "Solving long-tailed recognition with deep realistic taxonomic classifier," in *European Conference on Computer Vision*, Springer, 2020, pp. 171–189.

[180] T. Elsken, J. H. Metzen, and F. Hutter, "Neural architecture search: A survey," *The Journal of Machine Learning Research*, vol. 20, no. 1, pp. 1997–2017, 2019.

[181] X. Zhang, X. Zhou, M. Lin, and J. Sun, "Shufflenet: An extremely efficient convolutional neural network for mobile devices," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2018, pp. 6848–6856.

[182] A. G. Howard, M. Zhu, B. Chen, D. Kalenichenko, W. Wang, T. Weyand, M. Andreetto, and H. Adam, "Mobilenets: Efficient convolutional neural networks for mobile vision applications," *arXiv preprint arXiv:1704.04861*, 2017.

[183] B. Baker, O. Gupta, N. Naik, and R. Raskar, "Designing neural network architectures using reinforcement learning," *arXiv:1611.02167*, 2016.

[184] E. Real, S. Moore, A. Selle, S. Saxena, Y. L. Suematsu, J. Tan, Q. V. Le, and A. Kurakin, "Large-scale evolution of image classifiers," in *International Conference on Machine Learning*, PMLR, 2017, pp. 2902–2911.

[185] R. Duggal, H. Zhou, S. Yang, Y. Xiong, W. Xia, Z. Tu, and S. Soatto, "Compatibility-aware heterogeneous visual search," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2021, pp. 10 723–10 732.

[186] B. Baker, O. Gupta, R. Raskar, and N. Naik, "Accelerating neural architecture search using performance prediction," *arXiv preprint arXiv:1705.10823*, 2017.

[187] S. Falkner, A. Klein, and F. Hutter, "Bohb: Robust and efficient hyperparameter optimization at scale," in *International Conference on Machine Learning*, PMLR, 2018, pp. 1437–1446.

[188] Z. Lu, G. Sreekumar, E. Goodman, W. Banzhaf, K. Deb, and V. N. Boddeti, "Neural architecture transfer," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 43, no. 9, pp. 2971–2989, 2021.

[189] R. Panda, M. Merler, M. S. Jaiswal, H. Wu, K. Ramakrishnan, U. Finkler, C.-F. R. Chen, M. Cho, R. Feris, D. Kung, *et al.*, "Nastransfer: Analyzing architecture transferability in large scale neural architecture search," in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 35, 2021, pp. 9294–9302.

[190] X. Dong, L. Liu, K. Musial, and B. Gabrys, "Nats-bench: Benchmarking nas algorithms for architecture topology and size," *IEEE transactions on pattern analysis and machine intelligence*, 2021.

[191] A. Chowdhery, S. Narang, J. Devlin, M. Bosma, G. Mishra, A. Roberts, P. Barham, H. W. Chung, C. Sutton, S. Gehrmann, *et al.*, "Palm: Scaling language modeling with pathways," *arXiv preprint arXiv:2204.02311*, 2022.