Towards Adaptive Intelligence

Aditya Kusupati

A dissertation

submitted in partial fulfillment of the

requirements for the degree of

Doctor of Philosophy

University of Washington
2024

Reading Committee:
Ali Farhadi, Co-Chair
Sham Kakade, Co-Chair
Luke Zettlemoyer

Program Authorized to Offer Degree:
Computer Science and Engineering

University of Washington

Abstract

Towards Adaptive Intelligence

Aditya Kusupati

Co-chairs of the Supervisory Committee:

Professor Ali Farhadi
Computer Science and Engineering

Professor Sham Kakade
Computer Science and Engineering

Living beings, including humans, are highly adaptive, especially in terms of context and compute (resources). While intelligent machine learning systems are ubiquitous today, their current rigid design hinders adaptation as they struggle with ever-changing data, use cases, and deployment settings, requiring dedicated efforts to function properly. In this thesis, I present my work towards enabling adaptive machine learning solutions for flexible and seamless deployment across widely changing scenarios. First, I present Matryoshka information packing for adaptive data representations to handle growing data size and task-specific usage seamlessly. Then, I build a web-scale search system, AdANNS, leveraging matryoshka representations to enable adaptive search across data. Next, I extend these principles to the neural networks, crafting MatFormer models. This next-generation Transformer architecture adapts its computational footprint based on input and device with minimal overhead during deployment. Along the way, I worked on the first end-to-end learnable sparsity solution to solve the problem of optimal compute allocation across layers of neural networks. Further, to address the inherent rigidity in the design of web-scale intelligent systems, I worked on differentiable search solutions, fundamentally rethinking how large-scale AI pipelines harness data for continuous improvement. Finally, I conclude with the impact these works had in real-world deployments and present future works directed towards adaptive contextual and continual intelligence across disciplines.

Acknowledgements

It has been a delightful PhD experience, the past 5 years, and I would attribute it to all my advisors, mentors, collaborators, friends, and family. It might be rare to hear, but I would do it all over again in a heartbeat!

First, I am eternally grateful to my advisors, Ali Farhadi, Sham Kakade, and Prateek Jain. Each of them helped me evolve in many aspects beyond technical and research acumen. Something I can never fathom is the amount of trust and confidence they had in me, even when (most of the time, honestly) I did not believe in myself. I thank them for their unwavering support all along. Ali is the reason I joined the Allen School. The 1-1 meeting during visit days in 2019 convinced me beyond doubt that I would be happy working with him at UW – it turned out to be better than I could have ever dreamt. He has been an advisor I aspire to be like in every way. Ali has been the pillar of my strength in countless ways. He just gets me, which has been a rarity all my life! All I can say is thank you, Ali. I am always surprised by how Sham operates, his technical depth in seemingly unrelated research directions amazes me. Sham has one ability that I have rarely seen in anyone and want to imbibe, apart from the flair (I do not think I can ever match that) – he can see 6 months into the future of a problem anyone is working on and tell what shall work and what will fail, he rarely was wrong in that. He along with Ali cheered me on and supported my convictions even when they were never the hottest things and strongly supported my faculty search. Sham took a chance on me to do empirical research and I am so glad I got to work with him on everything Machine Learning. Prateek, my unofficial third advisor, mentor, confidant, and friend for the past seven years, has borne the brunt of my shenanigans. I still remember the call I picked up at KFC in Bombay about MSR when I was confused about what to do next. He supported me through every challenge, sometimes in person, most times across time zones. He brought me to Google, which has incredibly changed my research trajectory. He is the reason I love having co-advisors, and I thank him for treating me as a peer despite his accomplishments – which still surprises people.

I admire my PhD committee – Luke Zettlemoyer, Zaid Harchaoui, and Rahul Sukthankar – and thank

them for all the suggestions and support along the way. Luke told me 2 years ago that my research will be relevant 2 years down the line and he was spot on. He is an extremely knowledgeable and fun person to be around. I knew Zaid through my academic siblings and then I worked on something he worked on over a decade ago, which caught his eye. He was incredibly supportive of that work before anyone else in the community saw the potential. That was partly the reason I kept working on new ways of representation learning for a while now. I thank Luke and Zaid for helping me in unexpected ways during my PhD. Rahul made an exception to be on my committee and it has been an incredible last couple of years knowing him at Google. He has championed most of the ideas I brought to the table. I am constantly amazed by his ability to stay on top of things despite leading a huge organization, yet still finds time to talk with me whenever needed. I also thank him for his support during my faculty applications and all the wisdom he shared.

I have also been very fortunate to have many great mentors along the way. My research journey started with inimitable advising from Soumen sir working on representation learning – who would have thought! He set me on a course and helped me reach my first checkpoint – Microsoft Research India – for which I am grateful. At MSR India, I learned the ropes of doing research primarily from my advisor Manik Varma with Prateek joining in a bit later. I owe the majority of my research philosophy and methodology to Manik. I am extremely lucky to have worked with him so closely for 2 years and I credit him for all of my real-world deployment obsession. Manik has taught me how to do empirical research right, given me life and career advice, and made me feel at home in Delhi and I thank him for that. Funnily, the research statement I wrote to work with him was about multi-modal search, which is what I eventually ended up doing in the latter part of my PhD. Tom Duerig came out of nowhere in my life. He matches my energy and has taken my work to places where I could never imagine. He believed in me and let me do so much stuff as a part-time student researcher in Google which is not typical. I enjoyed the chats with him and all the moonshots I pitched him – as he says I still have to deliver on one more thing I promised and I shall. Another person I never thought I would have such close ties to is Jeff Dean. Over the years, he patiently listened to my ideas, debated them, validated them, and encouraged me to take the leap when I was hesitant. I cherish the chats I have had with him and will remember how he fought for our ideas, just because it was the right thing to do, despite not having any skin in the game – I will try and pay it forward. Another person I would like to mention here is Ranjay Krishna who reminds me of what I could be if I do everything right. He has been a support system I

constantly relied on for the past 2 years since he came to UW. He also gave me the opportunity to co-teach a class for the first time which I am thankful for. I am very happy to have spent time with and learned from all my teachers in school, Supratik sir and other professors at IIT Bombay, Vishwa and Pierre during my undergrad internships, Naga, Harsha, Praneeth, Sriram and many others at MSR, Sewoong, Ludwig, Kevin, Hanna, Pang Wei, Abhishek, Byron and countless others at UW CSE, Sanja and Alyosha for my PhD internships, Kaifeng, Mojtaba, Howard, Manish, Inderjit, Sanjiv, Raphael, Jon, Iftekhar, Aakanksha, Rohan, and everyone who believed in me at Google. I also am grateful for the advice and help I received from people in the research community like Lucas, Abhinav, Matei, and Carlos to name a few. I especially want to shout out how amazing Kaifeng is as a manager at Google. I am also grateful to Inderjit, Abhinav, and Manish for their support during my faculty search. Innumerable mentors helped me along the way and I do not think I would be able to do justice in a single paragraph.

This is a special note dedicated to Elise and team (Elle, Les, Joe, and Chris) + UW CSE staff. UW CSE runs because Elise is here – there's nothing more that needs to be said. She has been my one-stop solution for countless things, and I am incredibly grateful that she chose our well-being above all else. She also supported some of my out-of-the-box ideas, such as PAMS/PARS and removing the GRE requirement, for which I am very thankful. The impact we have had with any of these programs is primarily due to Elise's efforts. UW CSE staff helped with so many last-minute things during the last 5 years and I also want to appreciate the CSE2 2nd floor housekeeping staff for her warm smile every day over the years. While not CSE staff, Tran and Hanna have done an incredible job handling my odd requests with Ali's calendar over the years.

I have been extremely fortunate to be part of RAIVN lab and Sham's lab. RAIVN lab was a lifeline. 5 of us, Matt, Mitchell, Gabe, James and I joined PhD together and I am lucky to have worked alongside them and learn from them. Especially, Matt has been an incredible soundboard and my default second author, and so was I to him – I learned so much from him and cherish his friendship. James helped me most unexpectedly by just being there in the lab during covid – that was the most important thing I needed then. I was always in competitive environments growing up and Mitchell provided me with the implicit competitiveness that helped me go beyond. Gabe showed me the much-needed relaxed attitude in life. I also benefited a lot from Kiana, Aaron, Rowan, and Wei-Chiu's advice at various stages of my life. I liked spending my time in the lab most days just hanging out with Sarah, Reza, Vivek, Ainaz, Kuo-Hao, Keivan, and Matt Deitke. I

have special memories with every one of them and will miss them once I am locked out of the lab – be it gossiping with Sarah, making teasers with Vivek, joking around with Ainaz, and hearing Reza's passionate speeches about problems with video benchmarks. It has been more fun since Ranjay's students joined the lab, especially Santy, Zixian, Amita, Scott, Cheng-Yu, and others. I also had a great time with my academic siblings from Sham's lab – Motoya, John, Krishna, and Aravind. Motoya joined at the same time as me and I wish covid hadn't happened so we could have spent more time together. John was a great mentor and teacher and Aravind had enough crazy ideas to go around. Finally, Krishna is someone who looked up to in undergrad and I was stoked to have him as a friend and mentor over the years.

The success of my work is due in large part to my amazing collaborators and mentees. I want to thank Kush and Ashish for helping me with my first research paper and I tried to pay it forward. I am grateful to also have worked with Manish, Don, Sachin, Oindrila, Yash, Nilesh, Sahil Bhatia & Verma. Dhruv, Gantavya, Aniket, Alex, Ani, Roozbeh, Sharan, Alan, Ethan, Sneha, Jinhyuk, Nithi, Spandana, Anirudh, Aishwarya, Lovish, Ramnath, Devvrit, Tim, Anshul, Gagan, Nidhi, Sujoy, Arsha, Umangi, Sailesh, Sheshansh, Shishir, Sridhar, Chirag, Kunal, Saurabh, Akarsh and many more – I am missing so many people here. I have learned something from each of you and thanks for helping me become a better researcher. Aniket and Ethan have trusted me enough to have been solely mentored for over a year during my PhD and I wish them both the best in their career. Sailesh, Sneha, Anirudh, Spandana, Nithi, Kaifeng, and many others at Google have believed in my ideas and helped them be a reality in serving billions of users daily.

Friends, I made a lot of them and am grateful for what every one of them brought to my life. I am grateful to have two of my closest friends, Raghav and Tapan, as my housemates for the past 5 years – I cherish quite a few core memories with them all while discussing crazy research (contact Raghav for more stories), Tapan's hospital adventures, our long drives in the focus and stupid adventures on the snow. We have been through a lot of ups and downs all while growing old enough to get PhDs. All my friends in UW, there are so many of them, not a day goes by where I don't surprise someone around me about knowing someone else. All the desi grads who have been together for a long time Pratyush, Kamath, Sudheesh, Priyal, Venky, Vishwas, Prashanth, Gantavya, Reshabh, Ashish, Bandhav, Sneha, Anshul, Divyansh, Rohan, Sriyas, Sahil, Santy – I am sure I missing a lot more here. Friends I made across UW labs Ana, Vivek, Mike, Alisa, Jon, Alex, Andrew, Josh, Ian, Thao, Dhruba, Tim, Sewon, Akari, Yizhong, Alice, Jingwei, Johanna, Yuxuan, Xiang,

Joel, Sally, Ewin, Kentrell, Kaiming, Miranda, Anant, Dorna, Maddy, Audrey, Joseph, Suchin, Bharagavi, Mandar, Jieyu, Aashaka, Mohit, Anas, Arti, Galen, Kalyani, Sam, Ofir, Aleks, Ashrujit, Jake, Nathan, Lalit, Ramya – yeah for sure I am blanking here. I want to highlight the support system Tim, Sewon, Krishna Murthy and Wei-Chiu were to me (a junior) during the faculty search.

I am also lucky to have a strong friend group in Seattle. This, I kid you not, includes my closest friends for the past 24 years. Pavan, Mahima, Rahul, and Sravanth have been an at-home feeling in a distant land. Adding Sashank to the mix, I am glad to have all of you in my life for over 20 years now and looking forward to the next 50 years. More friends trickled in through them and others – Rohit, Sukriti, Surya, Sriram, Anmol ..., the listing would do no justice, but I appreciate you all so much. Surprisingly, a core of my IIT B friends ended up living in Bellevue. Owais and Ritesh are two people I am glad to have in my life at various stages be it technical or personal. Vishnu, DB, Meher, Nitesh, and Park all used to be my respite – as they call, renew my membership – when I wanted to clear up my mind through their never-ending board games obsession. I also am lucky to have had other close friends from undergrad be around in the last few years like PV, Krishna Harsha, Vishal, Sailesh, Suma, Charitha, Siva, Samarth, Sandeep, Vaibhav, Suhas, Rahul, Dileep – there are so many of them, now that I think of it – but I appreciate everything you all have done for me and all your friendship. Abhinav's friendship was a surprise and he has been a constant in my life for the last 2 years now. I also want to shout out all my school friends – trust me, if I start listing everyone everywhere, this section will be bigger than my thesis – they were extremely crucial during my formative years. As evident from everything I wrote till now, I was not content with the number of friends I made here, so I expanded on even more through my other channels (MSR, conferences, research community and beyond), I especially want to thank Ashu, Shivam, Jathushan, Utkarsh, Yash, Jonathan among many others for being welcoming faces at conferences and giving me perspective. I also want to thank Sundar and Venky for being amazing roommates in Bangalore, and Kranthi and Akhil in insti. To all my friends, just remember that I shall be there if you ever need me, you can count on me!

I constantly joke that I have more extended family in the US than in India – which is mostly true. I am extremely grateful to Deepthakka and Vijay mama for having a place for me in Detroit for the past 5 years and helping me in every means possible. I also want to thank my Babai, Pinni, Peddamma, Tathagaru, Mama, and Atha among many people who hosted me across the US. I am lucky to have most of my cousins in the

US, Vinnu, Tinku, Amala, Vamsi, Bunty, Chinnari, Vinita, Chandu, and Babloo among many others – I have not listed here – and have had fun with them. All my love to the next-generation kids in these households. It feels good to detach from research and just spend some quality time with all these people.

Finally, I am ridiculously lucky to have a great family – despite all my mischief and what I bring to the mix. To be honest, all I did was follow the trail blazed by my parents (Amma and Dad). The extraordinary amounts of freedom they let me have, shaped me to be the way I am right now. My mom has a reproductive physiology PhD and my dad is a medical doctor, they rarely cared about what I was up to in my life as long as I was happy – if they wanted I would have been an actual Doctor – well I settled down by being a Doctor in Computer Science (sounds funny). My mom sacrificed a lot for me and my brother growing up and has ever since vicariously lived her dreams of academic success through us – all while never asking for it nor expecting it. She gets the most credit for anything we have achieved – rightfully so! My dad is the reason I am so extroverted, he is great with people and the amount of love he gets everywhere is unparalleled. He gave me one life advice that has been a north star – strive to be the best in whatever you choose to work on and it does not matter what you choose to work on as long as you love it. He and my mom have a gap of at least a generation in terms of socioeconomic status and his rise through everything last 28 years has been inspiring. I love both of you and someday (probably my defense) show what I did as part of my PhD. I also happen to have an incredible brother, Uday. He is much more hardworking and inquisitive than I ever will be – no wonder he is doing a harder and more useful PhD than me. He has done an incredible job being my partner in crime all while validating every one of my moves by following in my footsteps as is or closely – thanks a lot for this! We went to the same school, undergrad, and pursued computer science research with passion leading to our doctoral journeys. Now that I am graduating, maybe we will do more idiotic adventures than usual – which already gives our mom enough panic attacks. My partner, Sanjana, has been a turning point in my life both personally and professionally – I found all the successes in my PhD only after I met her. Her love knows no bounds and care only ever grows. She has been my emotional support even with the distance of the entire I-90 between us the last year. Now that I have no more PhD excuses left to give, I want to spend a lot more time with her, travel the world, explore new things, and most importantly love her to the moon and back. Thank you for everything Sanju, I am a happier and better person because of you!

DEDICATION

To *my family*,

and the *ever-so-fascinating world* we live in,

that *led me to where I am* and shall *lead me to where I will*.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Modern intelligent systems are becoming increasingly monolithic, powered by gigantic foundation models trained on trillions of tokens of web data. To democratize Artificial Intelligence (AI) systems, it is imperative to ensure that they are not limited to running on multi-accelerator clusters but also on commodity devices like mobile phones seamlessly. Additionally, foundation models [24] exhibit a performance disparity between frequently encountered *head* tasks in the training data and less common *tail* tasks, necessitating their adaptation through efficient retrieval of relevant contextual data. Furthermore, echoing human learning principles, *not all tasks are equally challenging or require the entirety of the vast web data* – showcasing the need for adaptivity. My research methodology centers on translating these concepts into practical solutions for real-world implementation, ensuring that these intelligent systems are adaptive and can be scaled reliably and responsibly to serve all users equitably.

With the goal of *efficient, elastic and contextual intelligence* – in short **adaptive intelligence**, I focus on building fundamental machine learning (ML) building blocks that encompass: (1) elastic representations and models for accurate, adaptive and efficient deployment (Chapters 2, 4 and 5) and (2) mechanisms to make contextual data efficiently accessible to models for equitable adaptation (Chapters 3 and 6).

Chapter 2 introduces the "Matryoshka" way of packing information in a dense vector – the atomic building block of every ML model. This chapter covers work primarily from Kusupati et al. [147] where we proposed "Matryoshka" structure in dense vector representations to order the information from left to right based on importance in a nested fashion. Matryoshka representation learning (MRL) [147] helps neural

networks output dense vectors that are inherently multi-granular by jointly optimizing the same learning task at a select few embedding granularities. MRL helps obtain accurate low-dimensional representations of desired quality and cost/size by taking the appropriate number of leftmost coordinates. This helps *elastically* cater to downstream tasks of varying requirements like retrieval, classification, etc., in the transfer learning paradigm. MRL is simple, scalable, and agnostic to representation learning setups, modalities, and models.

Chapter 3 covers work primarily from Rege et al. [209] where we developed approximate nearest neighbor search (ANNS) methods that leverage the elastic embeddings for flexible search [209]. We focused on flexibility within ANNS building blocks by leveraging the multi-granular and elastic MRL representations for web-scale data. This helped design $2-10\times$ efficient ANNS indices for web data without compromising accuracy. Now, matryoshka representations could be used at web-scale for on-the-fly adaptive and equitable search without the need to rebuild indices across granularities.

Chapter 4 covers work primarily from Devvrit et al. [62] where we introduced elastic universal matryoshka neural network models – primarily Transformers. We developed MatFormer [62] which brought the matryoshka structure to all of the Transformer [245] architecture. MatFormer enables extractions of 100s of smaller accurate models for a *wide range of static deployment constraints and also supports dynamic conditional inference* on-the-fly based on task hardness [218, 121, 32] and resource constraints. Additionally, MatFormer provides smaller submodels that are inherently consistent with the universal model due to the preservation of metric-space structure. This allows for significant speed-ups in inference time optimization of generative language models [157] as well as enabling adaptive query encoders for large-scale retrieval for the first time. Similar to MRL, MatFormer is domain and setup agnostic while scaling, to internet-scale, as reliably as the default Transformer. Overall, MatFormer is a next-generation architecture that elicits *elasticity* and *virtualization* within foundation models that form the basis of modern-day web-scale intelligent systems.

Chapter 5 takes a slight detour towards solving the problem of compute allocation across neural network layers – which is important for the adaptive deployment of MatFormer models. This chapter primarily covers work from Kusupati et al. [145] where we introduced the concept of learnable sparsity which was the first end-to-end differentiable method that achieved state-of-the-art "Accuracy vs FLOPs vs Model size".

Chapter 6 covers work from Kusupati et al. [146] where we fundamentally rethought traditional search by learning compact binary codes for data points that double as accurate representations and efficient web-scale

indices. We formulated entire pipeline as a representation learning problem, through the lens of compression and scalable instance classification, where each data point is assigned a learned low-dimensional binary code [146]. These binary codes have the required semantic information for downstream tasks, while also acting as a native hash-based index for all the data points. This works at scale resulting in an accurate encoding of 1 Million images with just 20 bits per image which also serves as an extremely efficient web-scale index for search on-demand. Rethinking search to be end-to-end differentiable and free of scaffolds [143] can result in large amounts of data being available for offline search based on the context during deployment.

During my PhD, I also had the wonderful opportunity to work on deploying efficient ML solutions for HCI applications[120], large-scale multi-modal models [277], better object-centric [252] and continual learning [251], web-scale datasets for underserved tasks like 3D modeling [54] and multi-lingual NLP [139], web-scale state-of-the-art text embeddings [154], state-of-the-art image to scene synthesis [253] and novel autoregressive decoding algorithms [226].

I conclude with a short discussion on the real-world impact of the work presented in this thesis and talk about future directions spanning indexing the world, contextual, and continually learning foundation models in Chapter 7. The overarching theme of my research is to improve the building blocks of ML systems to do more for the same resource usage with simple and scalable techniques – **adaptive ML building blocks**. Due to the fundamental nature, the techniques I built, for modeling and retrieving data, work together seamlessly and can help build truly elastic and adaptive web-scale intelligent systems to serve the users dynamically and equitably based on task, context, and resource constraints. Finally, each of these research directions stands on its merit in solving high-impact fundamental problems like large-scale search and efficient deployment that have potential applications across various fields that extend beyond computer science.

Chapter 2

Matryoshka Representation Learning

2.1 Overview

Learned representations are a central component in modern ML systems, serving a multitude of downstream tasks. When training such representations, it is often the case that computational and statistical constraints for each downstream task are unknown. In this context, rigid fixed-capacity representations can be either over or under-accommodating to the task at hand. This leads us to ask: *can we design a flexible representation that can adapt to multiple downstream tasks with varying computational resources?* In this chapter, our main contribution is 🪆 Matryoshka Representation Learning (MRL) which encodes information at different granularities and allows a single embedding to adapt to the computational constraints of downstream tasks. MRL minimally modifies existing representation learning pipelines and imposes no additional cost during inference and deployment. MRL learns coarse-to-fine representations that are at least as accurate and rich as independently trained low-dimensional representations. The flexibility within the learned Matryoshka Representations offer: (a) up to $\mathbf{14}\times$ smaller embedding size for ImageNet-1K classification at the same level of accuracy; (b) up to $\mathbf{14}\times$ real-world speed-ups for large-scale retrieval on ImageNet-1K and 4K; and (c) up to $\mathbf{2}\%$ accuracy improvements for long-tail few-shot classification, all while being as robust as the original representations. Finally, we show that MRL extends seamlessly to web-scale datasets (ImageNet, JFT) across various modalities – vision (ViT, ResNet), vision + language (ALIGN) and language (BERT). MRL code and pretrained models are open-sourced at https://github.com/RAIVNLab/MRL.

2.2 Introduction

Learned representations [153] are fundamental building blocks of real-world ML systems [190, 248]. Trained once and frozen, d-dimensional representations encode rich information and can be used to perform multiple downstream tasks [17]. The deployment of deep representations has two steps: (1) an expensive yet constant-cost forward pass to compute the representation [98] and (2) utilization of the representation for downstream applications [134, 244]. Compute costs for the latter part of the pipeline scale with the embedding dimensionality as well as the data size (N) and label space (L). At web-scale [51, 235] this utilization cost overshadows the feature computation cost. The rigidity in these representations forces the use of high-dimensional embedding vectors across multiple tasks despite the varying resource and accuracy constraints that require flexibility.

Human perception of the natural world has a naturally coarse-to-fine granularity [94, 103]. However, perhaps due to the inductive bias of gradient-based training [232], deep learning models tend to diffuse "information" across the entire representation vector. The desired elasticity is usually enabled in the existing flat and fixed representations either through training multiple low-dimensional models [98], jointly optimizing sub-networks of varying capacity [30, 275] or post-hoc compression [109, 165]. Each of these techniques struggle to meet the requirements for adaptive large-scale deployment either due to training/maintenance overhead, numerous expensive forward passes through all of the data, storage and memory cost for multiple copies of encoded data, expensive on-the-fly feature selection or a significant drop in accuracy. By encoding coarse-to-fine-grained representations, which are as accurate as the independently trained counterparts, we learn with minimal overhead a representation that can be deployed *adaptively* at no additional cost during inference.

We introduce Matryoshka Representation Learning (MRL) to induce flexibility in the learned representation. MRL learns representations of varying capacities within the same high-dimensional vector through explicit optimization of $O(\log(d))$ lower-dimensional vectors in a nested fashion, hence the name Matryoshka. MRL can be adapted to any existing representation pipeline and is easily extended to many standard tasks in computer vision and natural language processing. Figure 2.1 illustrates the core idea of Matryoshka Representation Learning (MRL) and the adaptive deployment settings of the learned Matryoshka Representations.

The first m-dimensions, $m \in [d]$, of the Matryoshka Representation is an information-rich low-dimensional vector, at no additional training cost, that is as accurate as an independently trained m-dimensional representation. The information within the Matryoshka Representation increases with the dimensionality creating a coarse-to-fine grained representation, all without significant training or additional deployment overhead. MRL equips the representation vector with the desired flexibility and multifidelity that can ensure a near-optimal accuracy-vs-compute trade-off. With these advantages, MRL enables adaptive deployment based on accuracy and compute constraints.

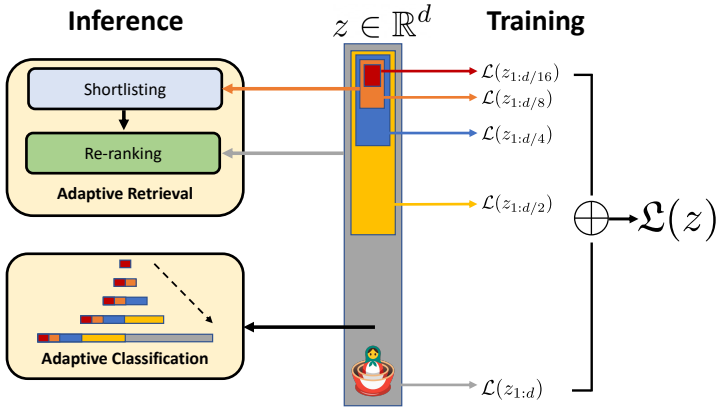


Figure 2.1: Matryoshka Representation Learning is adaptable to any representation learning setup and begets a Matryoshka Representation z by optimizing the original loss $\mathcal{L}(.)$ at $O(\log(d))$ chosen representation sizes. Matryoshka Representation can be utilized effectively for adaptive deployment across environments and downstream tasks.

The Matryoshka Representations improve efficiency for large-scale classification and retrieval without any significant loss of accuracy. While there are potentially several applications of coarse-to-fine Matryoshka Representations, in this work we focus on two key building blocks of real-world ML systems: large-scale classification and retrieval. For classification, we use adaptive cascades with the variable-size representations from a model trained with MRL, significantly reducing the average dimension of embeddings needed to achieve a particular accuracy. For example, on ImageNet-1K, MRL + adaptive classification results in up to a $14\times$ smaller representation size at the same accuracy as baselines (Section 2.5.2). Similarly, we use MRL in an adaptive retrieval system. Given a query, we shortlist retrieval candidates using the first few dimensions of the query embedding, and then successively use more dimensions to re-rank the retrieved set. A simple implementation of this approach leads to $128\times$ theoretical (in terms of FLOPS) and $14\times$ wall-clock time speedups compared to a single-shot retrieval system that uses a standard embedding vector; note that MRL's retrieval accuracy is comparable to that of single-shot retrieval (Section 2.5.3). Finally, as MRL explicitly learns coarse-to-fine representation vectors, intuitively it should share more semantic information among its various dimensions (Figure 2.5). This is reflected in up to 2% accuracy gains in long-tail continual

learning settings while being as robust as the original embeddings. Furthermore, due to its coarse-to-fine grained nature, MRL can also be used as method to analyze hardness of classification among instances and information bottlenecks.

We make the following key contributions:

1. We introduce Matryoshka Representation Learning (MRL) to obtain flexible representations (Matryoshka Representations) for adaptive deployment (Section 2.4).
2. Up to $14\times$ faster yet accurate large-scale classification and retrieval using MRL (Section 2.5).
3. Seamless adaptation of MRL across modalities (vision - ResNet & ViT, vision + language - ALIGN, language - BERT) and to web-scale data (ImageNet-1K/4K, JFT-300M and ALIGN data).
4. Further analysis of MRL's representations in the context of other downstream tasks (Section 2.6).

2.3 Related Work

Representation Learning. Large-scale datasets like ImageNet [55, 215] and JFT [235] enabled the learning of general purpose representations for computer vision [17, 270]. These representations are typically learned through supervised and un/self-supervised learning paradigms. Supervised pretraining [98, 137, 229] casts representation learning as a multi-class/label classification problem, while un/self-supervised learning learns representation via proxy tasks like instance classification [268] and reconstruction [100, 181]. Recent advances [41, 99] in contrastive learning [92] enabled learning from web-scale data [65] that powers large-capacity cross-modal models [58, 127, 202, 277]. Similarly, natural language applications are built [112] on large language models [27] that are pretrained [198, 214] in a un/self-supervised fashion with masked language modelling [61] or autoregressive training [201].

Matryoshka Representation Learning (MRL) is complementary to all these setups and can be adapted with minimal overhead (Section 2.4). MRL equips representations with multifidelity at no additional cost which enables adaptive deployment based on the data and task (Section 2.5).

Efficient Classification and Retrieval. Efficiency in classification and retrieval during inference can be studied with respect to the high yet constant deep featurization costs or the search cost which scales with the size of the label space and data. Efficient neural networks address the first issue through a variety of

algorithms [81, 145] and design choices [111, 144, 237]. However, with a strong featurizer, most of the issues with scale are due to the linear dependence on number of labels (L), size of the data (N) and representation size (d), stressing RAM, disk and processor all at the same time.

The sub-linear complexity dependence on number of labels has been well studied in context of compute [16, 122, 200] and memory [64] using Approximate Nearest Neighbor Search (ANNS) [179] or leveraging the underlying hierarchy [56, 146]. In case of the representation size, often dimensionality reduction [216, 242], hashing techniques [50, 142, 217] and feature selection [182] help in alleviating selective aspects of the $O(d)$ scaling at a cost of significant drops in accuracy. Lastly, most real-world search systems [38, 51] are often powered by large-scale embedding based retrieval [37, 190] that scales in cost with the ever increasing web-data. While categorization [244, 272] clusters similar things together, it is imperative to be equipped with retrieval capabilities that can bring forward every instance [26]. Approximate Nearest Neighbor Search (ANNS) [118] makes it feasible with efficient indexing [50] and traversal [19, 22] to present the users with the most similar documents/images from the database for a requested query. Widely adopted HNSW [179] ($O(d\log(N))$) is as accurate as exact retrieval ($O(dN)$) at the cost of a graph-based index overhead for RAM and disk [125].

MRL tackles the linear dependence on embedding size, d, by learning multifidelity Matryoshka Representations. Lower-dimensional Matryoshka Representations are as accurate as independently trained counterparts without the multiple expensive forward passes. Matryoshka Representations provide an *intermediate abstraction* between high-dimensional vectors and their efficient ANNS indices through the adaptive embeddings nested within the original representation vector (Section 2.5). All other aforementioned efficiency techniques are complementary and can be readily applied to the learned Matryoshka Representations obtained from MRL.

Several works in efficient neural network literature [30, 250, 275] aim at packing neural networks of varying capacity within the same larger network. However, the weights for each progressively smaller network can be different and often require distinct forward passes to isolate the final representations. This is detrimental for adaptive inference due to the need for re-encoding the entire retrieval database with expensive sub-net forward passes of varying capacities. Several works [70, 83, 188, 160] investigate the notions of intrinsic dimensionality and redundancy of representations and objective spaces pointing to minimum

description length [212]. Finally, ordered representations proposed by Rippel et al. [211] use nested dropout in the context of autoencoders to learn nested representations. MRL differentiates itself in formulation by optimizing only for $O(\log(d))$ nesting dimensions instead of $O(d)$. Despite this, MRL diffuses information to intermediate dimensions interpolating between the optimized Matryoshka Representation sizes accurately (Figure 2.5); making web-scale feasible.

2.4 Matryoshka Representation Learning

For $d \in \mathbb{N}$, consider a set $\mathcal{M} \subset [d]$ of representation sizes. For a datapoint x in the input domain $\mathcal{X}$, our goal is to learn a d-dimensional representation vector $z \in \mathbb{R}^d$. For every $m \in \mathcal{M}$, Matryoshka Representation Learning (MRL) enables each of the first m dimensions of the embedding vector, $z_{1:m} \in \mathbb{R}^m$ to be independently capable of being a transferable and general purpose representation of the datapoint x. We obtain z using a deep neural network $F(\cdot\,; \theta_F) \colon \mathcal{X} \to \mathbb{R}^d$ parameterized by learnable weights θ_F, i.e., $z \coloneqq F(x; \theta_F)$. The multi-granularity is captured through the set of the chosen dimensions $\mathcal{M}$, that contains less than $\log(d)$ elements, i.e., $|\mathcal{M}| \leq \lfloor \log(d) \rfloor$. The usual set $\mathcal{M}$ consists of consistent halving until the representation size hits a low information bottleneck. We discuss the design choices in Section 2.5 for each of the representation learning settings.

For the ease of exposition, we present the formulation for fully supervised representation learning via multi-class classification. Matryoshka Representation Learning modifies the typical setting to become a multi-scale representation learning problem on the same task. For example, we train ResNet50 [98] on ImageNet-1K [215] which embeds a 224×224 pixel image into a $d = 2048$ representation vector and then passed through a linear classifier to make a prediction, $\hat{y}$ among the $L = 1000$ labels. For MRL, we choose $\mathcal{M} = \{8, 16, \ldots, 1024, 2048\}$ as the nesting dimensions.

Suppose we are given a labelled dataset $\mathcal{D} = \{(x_1, y_1), \ldots, (x_N, y_N)\}$ where $x_i \in \mathcal{X}$ is an input point and $y_i \in [L]$ is the label of x_i for all $i \in [N]$. MRL optimizes the multi-class classification loss for each of the nested dimension $m \in \mathcal{M}$ using standard empirical risk minimization using a separate linear classifier, parameterized by $\mathbf{W}^{(m)} \in \mathbb{R}^{L \times m}$. All the losses are aggregated after scaling with their relative importance

$(c_m \geq 0)_{m \in \mathcal{M}}$ respectively. That is, we solve

$$\min_{\{\mathbf{W}^{(m)}\}_{m \in \mathcal{M}},\, \theta_F} \frac{1}{N} \sum_{i \in [N]} \sum_{m \in \mathcal{M}} c_m \cdot \mathcal{L}\left(\mathbf{W}^{(m)} \cdot F(x_i; \theta_F)_{1:m}\ ;\ y_i\right) , \tag{2.1}$$

where $\mathcal{L}\colon \mathbb{R}^L \times [L] \rightarrow \mathbb{R}_+$ is the multi-class softmax cross-entropy loss function. This is a standard optimization problem that can be solved using sub-gradient descent methods. We set all the importance scales, $c_m = 1$ for all $m \in \mathcal{M}$; see Section 2.6 for ablations. Lastly, despite only optimizing for $O(\log(d))$ nested dimensions, MRL results in accurate representations, that interpolate, for dimensions that fall between the chosen granularity of the representations (Section 2.5.2).

We call this formulation as Matryoshka Representation Learning (MRL). A natural way to make this efficient is through weight-tying across all the linear classifiers, i.e., by defining $\mathbf{W}^{(m)} = \mathbf{W}_{1:m}$ for a set of common weights $\mathbf{W} \in \mathbb{R}^{L \times d}$. This would reduce the memory cost due to the linear classifiers by almost half, which would be crucial in cases of extremely large output spaces [244, 272]. This variant is called *Efficient* Matryoshka Representation Learning (MRL–E). Refer to Algorithms in Appendix A of Kusupati et al. [147] for the building blocks of Matryoshka Representation Learning (MRL).

Adaptation to Learning Frameworks. MRL can be adapted seamlessly to most representation learning frameworks at web-scale with minimal modifications (Section 2.5.1). For example, MRL's adaptation to masked language modelling reduces to MRL–E due to the weight-tying between the input embedding matrix and the linear classifier. For contrastive learning, both in context of vision & vision + language, MRL is applied to both the embeddings that are being contrasted with each other. The presence of normalization on the representation needs to be handled independently for each of the nesting dimension for best results (see Appendix C of Kusupati et al. [147] for more details).

2.5 Applications

In this section, we discuss Matryoshka Representation Learning (MRL) for a diverse set of applications along with an extensive evaluation of the learned multifidelity representations. Further, we showcase the downstream applications of the learned Matryoshka Representations for flexible large-scale deployment

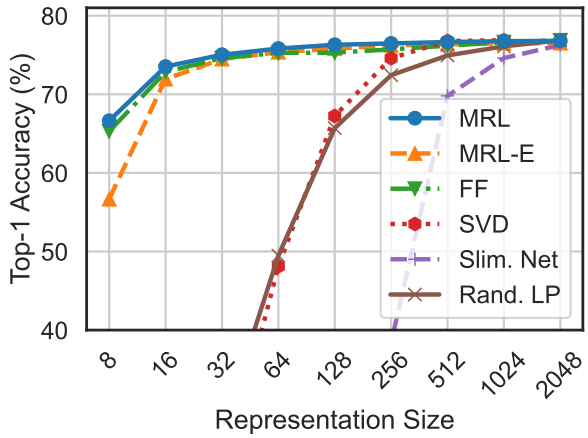


Figure 2.2: ImageNet-1K linear classification accuracy of ResNet50 models. MRL is as accurate as the independently trained FF models for every representation size.

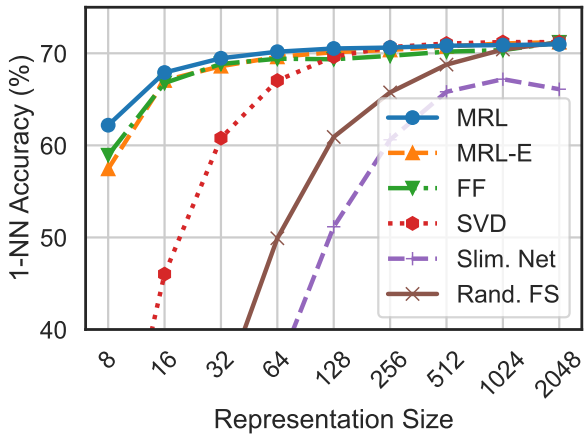


Figure 2.3: ImageNet-1K 1-NN accuracy of ResNet50 models measuring the representation quality for downstream task. MRL outperforms all the baselines across all representation sizes.

through (a) Adaptive Classification (AC) and (b) Adaptive Retrieval (AR).

2.5.1 Representation Learning

We adapt Matryoshka Representation Learning (MRL) to various representation learning setups (a) Supervised learning for vision: ResNet50 [98] on ImageNet-1K [215] and ViT-B/16 [67] on JFT-300M [235], (b) Contrastive learning for vision + language: ALIGN model with ViT-B/16 vision encoder and BERT language encoder on ALIGN data [127] and (c) Masked language modelling: BERT [61] on English Wikipedia and BooksCorpus [283]. Please refer to Appendices B and C of Kusupati et al. [147] for details regarding the model architectures, datasets and training specifics.

We do not search for best hyper-parameters for all MRL experiments but use the same hyper-parameters as the independently trained baselines. ResNet50 outputs a 2048-dimensional representation while ViT-B/16 and BERT-Base output 768-dimensional embeddings for each data point. We use $\mathcal{M} = \{8, 16, 32, 64, 128, 256, 512, 1024, 2048\}$ and $\mathcal{M} = \{12, 24, 48, 96, 192, 384, 768\}$ as the explicitly optimized nested dimensions respectively. Lastly, we extensively compare the MRL and MRL–E models to independently trained low-dimensional (fixed feature) representations (FF), dimensionality reduction (SVD), sub-net method (slimmable networks [275]) and randomly selected features of the highest capacity FF model.

In section 2.5.2, we evaluate the quality and capacity of the learned representations through linear

classification/probe (LP) and 1-nearest neighbour (1-NN) accuracy. Experiments show that MRL models remove the dependence on $|\mathcal{M}|$ resource-intensive independently trained models for the coarse-to-fine representations while being as accurate. Lastly, we show that despite optimizing only for $|\mathcal{M}|$ dimensions, MRL models diffuse the information, in an interpolative fashion, across all the d dimensions providing the finest granularity required for adaptive deployment.

2.5.2 Classification

Figure 2.2 compares the linear classification accuracy of ResNet50 models trained and evaluated on ImageNet-1K. ResNet50–MRL model is at least as accurate as each FF model at every representation size in $\mathcal{M}$ while MRL–E is within 1% starting from 16-dim. Similarly, Figure 2.3 showcases the comparison of learned representation quality through 1-NN accuracy on ImageNet-1K (trainset with 1.3M samples as the database and validation set with 50K samples as the queries). Matryoshka Representations are up to 2% more accurate than their fixed-feature counterparts for the lower-dimensions while being as accurate elsewhere. 1-NN accuracy is an excellent proxy, at no additional training cost, to gauge the utility of learned representations in the downstream tasks.

We also evaluate the quality of the representations from training ViT-B/16 on JFT-300M alongside the ViT-B/16 vision encoder of the ALIGN model – two web-scale setups. Due to the expensive nature of these experiments, we only train the highest capacity fixed feature model and choose random features for evaluation in lower-dimensions. Web-scale is a compelling setting for MRL due to its relatively inexpensive training overhead while providing multifidelity representations for downstream tasks. Figure 2.4, evaluated with 1-NN on ImageNet-1K, shows that all the MRL models for JFT and ALIGN are highly accurate while providing an excellent cost-vs-accuracy trade-off at lower-dimensions. These experiments show that MRL seamlessly scales to large-scale models and web-scale datasets while providing the otherwise prohibitively expensive multi-granularity in the process. We also have similar observations when pretraining BERT; please see Appendix D.2 of Kusupati et al. [147] for more details. Our experiments also show that post-hoc compression (SVD), linear probe on random features, and sub-net style slimmable networks drastically lose accuracy compared to MRL as the representation size decreases. Finally, Figure 2.5 shows that, while MRL explicitly optimizes $O(\log(d))$ nested representations – removing the $O(d)$ dependence [211] –, the

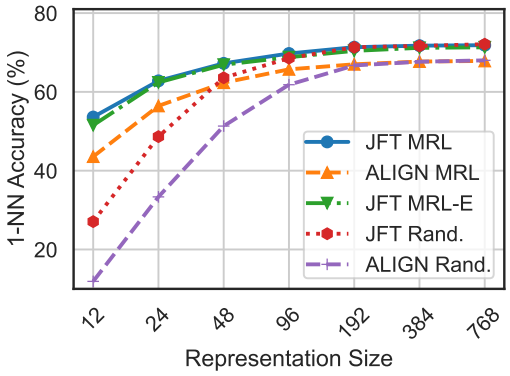


Figure 2.4: ImageNet-1K 1-NN accuracy for ViT-B/16 models trained on JFT-300M & as part of ALIGN. MRL scales seamlessly to web-scale with minimal training overhead.

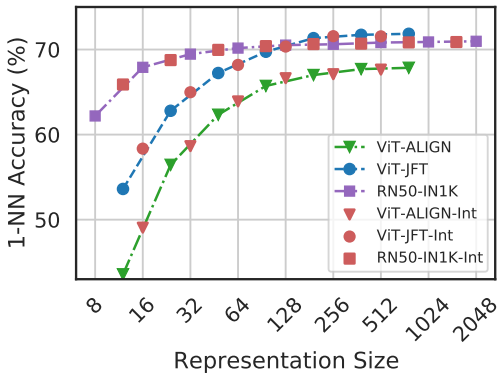


Figure 2.5: Despite optimizing MRL only for $O(\log(d))$ dimensions for ResNet50 and ViT-B/16 models; the accuracy in the intermediate dimensions shows interpolating behaviour.

coarse-to-fine grained information is interpolated across all d dimensions providing highest flexibility for adaptive deployment.

Adaptive Classification

The flexibility and coarse-to-fine granularity within Matryoshka Representations allows model cascades [246] for Adaptive Classification (AC) [94]. Unlike standard model cascades [258], MRL does not require multiple expensive neural network forward passes. To perform AC with an MRL trained model, we learn thresholds on the maximum softmax probability [104] for each nested classifier on a holdout validation set. We then use these thresholds to decide when to transition to the higher dimensional representation (e.g $8 \rightarrow 16 \rightarrow 32$) of the MRL model. Appendix D.1 of Kusupati et al. [147] discusses the implementation and learning of thresholds for cascades used for adaptive classification in detail.

Figure 2.6 shows the comparison between cascaded MRL representations (MRL–AC) and independently trained fixed feature (FF) models on ImageNet-1K with ResNet50. We computed the expected representation size for MRL–AC based on the final dimensionality used in the cascade. We observed that MRL–AC was as accurate, 76.30%, as a 512-dimensional FF model but required an expected dimensionality of ~ 37 while being only 0.8% lower than the 2048-dimensional FF baseline. Note that all MRL–AC models are significantly more accurate than the FF baselines at comparable representation sizes. MRL–AC uses up to $\sim 14\times$ smaller representation size for the same accuracy which affords computational efficiency as the label space grows [244]. Lastly, our results with MRL–AC indicate that instances and classes vary in difficulty

which we analyze in Section 2.6 and Appendix J of Kusupati et al. [147].

2.5.3 Retrieval

Nearest neighbour search with learned representations powers a plethora of retrieval and search applications [51, 248, 38, 190]. In this section, we discuss the image retrieval performance of the pretrained ResNet50 models (Section 2.5.1) on two large-scale datasets ImageNet-1K [215] and ImageNet-4K. ImageNet-1K has a database size of $\sim$1.3M and a query set of 50K samples uniformly spanning 1000 classes. We also introduce ImageNet-4K which has a database size of $\sim$4.2M and query set of $\sim$200K samples uniformly spanning 4202 classes (see Appendix B of Kusupati et al. [147] for details). A single forward pass on ResNet50 costs 4 GFLOPs while exact retrieval costs 2.6 GFLOPs per query for ImageNet-1K. Although retrieval overhead is 40% of the total cost, retrieval cost grows linearly with the size of the database. ImageNet-4K presents a retrieval benchmark where the exact search cost becomes the computational bottleneck (8.6 GFLOPs per query). In both these settings, the memory and disk usage are also often bottlenecked by the large databases. However, in most real-world applications exact search, $O(dN)$, is replaced with an approximate nearest neighbor search (ANNS) method like HNSW [179], $O(d\log(N))$, with minimal accuracy drop at the cost of additional memory overhead.

The goal of image retrieval is to find images that belong to the same class as the query using representations obtained from a pretrained model. In this section, we compare retrieval performance using mean Average Precision @ 10 (mAP@10) which comprehensively captures the setup of relevant image retrieval at scale. We measure the cost per query using exact search in MFLOPs. All embeddings are unit normalized and retrieved using the L2 distance metric. Lastly, we report an extensive set of metrics spanning mAP@k and P@k for $k = \{10, 25, 50, 100\}$ and real-world wall-clock times for exact search and HNSW. See Appendices E and F of Kusupati et al. [147] for more details.

Figure 2.7 compares the mAP@10 performance of ResNet50 representations on ImageNet-1K across dimensionalities for MRL, MRL–E, FF, slimmable networks along with post-hoc compression of vectors using SVD and random feature selection. Matryoshka Representations are often the most accurate while being up to 3% better than the FF baselines. Similar to classification, post-hoc compression and slimmable network baselines suffer from significant drop-off in retrieval mAP@10 with ≤ 256 dimensions. Appendix E

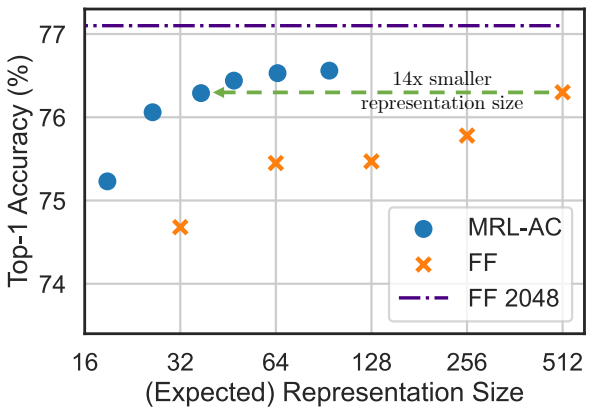


Figure 2.6: Adaptive classification on MRL ResNet50 using cascades results in $14\times$ smaller representation size for the same level of accuracy on ImageNet-1K ($\sim$ 37 vs 512 dims for 76.3%).

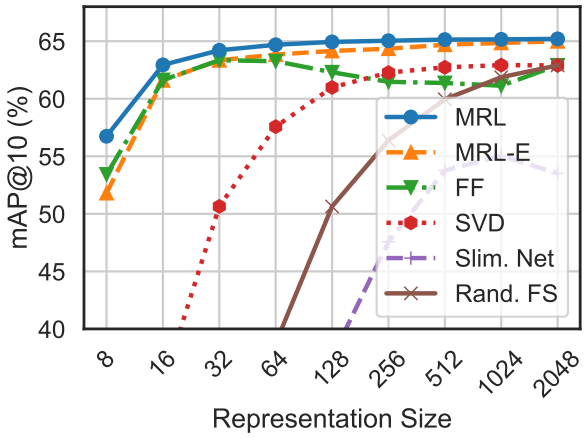


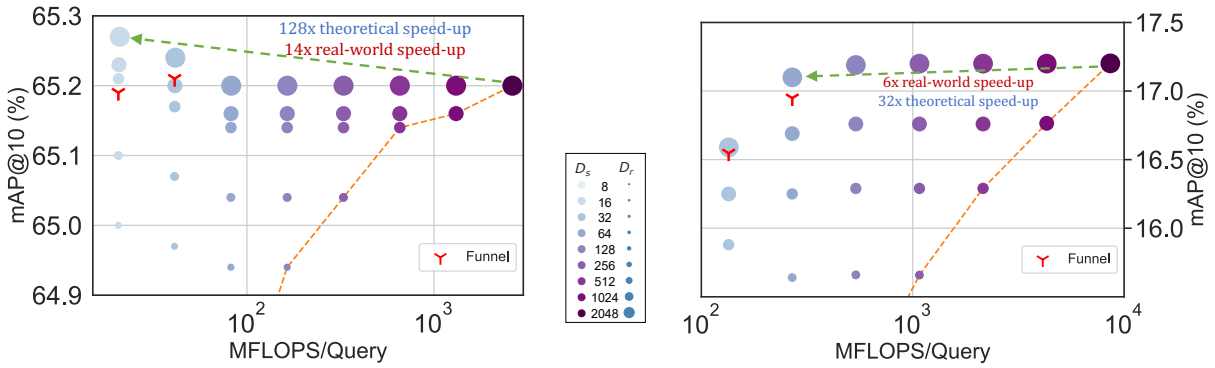
Figure 2.7: mAP@10 for Image Retrieval on ImageNet-1K with ResNet50. MRL consistently produces better retrieval performance over the baselines across all the representation sizes.

of Kusupati et al. [147] discusses the mAP@10 of the same models on ImageNet-4K.

MRL models are capable of performing accurate retrieval at various granularities without the additional expense of multiple model forward passes for the web-scale databases. FF models also generate independent databases which become prohibitively expense to store and switch in between. Matryoshka Representations enable adaptive retrieval (AR) which alleviates the need to use full-capacity representations, $d = 2048$, for all data and downstream tasks. Lastly, all the vector compression techniques [165, 126] used as part of the ANNS pipelines are complimentary to Matryoshka Representations and can further improve the efficiency-vs-accuracy trade-off.

Adaptive Retrieval

We benchmark MRL in the adaptive retrieval setting (AR) [134]. For a given query image, we obtained a shortlist, $K = 200$, of images from the database using a lower-dimensional representation, e.g. $D_s = 16$ followed by reranking with a higher capacity representation, e.g. $D_r = 2048$. In real-world scenarios where top ranking performance is the key objective, measured with mAP@k where k covers a limited yet crucial real-estate, AR provides significant compute and memory gains over single-shot retrieval with representations of fixed dimensionality. Finally, the most expensive part of AR, as with any retrieval pipeline, is the nearest neighbour search for shortlisting. For example, even naive re-ranking of 200 images with 2048 dimensions



(a) ImageNet-1K (b) ImageNet-4K

Figure 2.8: The trade-off between mAP@10 vs MFLOPs/Query for Adaptive Retrieval (AR) on ImageNet-1K (left) and ImageNet-4K (right). Every combination of D_s & D_r falls above the Pareto line (orange dots) of single-shot retrieval with a fixed representation size while having configurations that are as accurate while being up to $14\times$ faster in real-world deployment. Funnel retrieval is almost as accurate as the baseline while alleviating some of the parameter choices of Adaptive Retrieval.

only costs 400 KFLOPs. While we report exact search cost per query for all AR experiments, the shortlisting component of the pipeline can be sped-up using ANNS (HNSW). Appendix I of Kusupati et al. [147] has a detailed discussion on compute cost for exact search, memory overhead of HNSW indices and wall-clock times for both implementations. We note that using HNSW with 32 neighbours for shortlisting does not decrease accuracy during retrieval.

Figure 2.8 showcases the compute-vs-accuracy trade-off for adaptive retrieval using Matryoshka Representations compared to single-shot using fixed features with ResNet50 on ImageNet-1K. We observed that all AR settings lied above the Pareto frontier of single-shot retrieval with varying representation sizes. In particular for ImageNet-1K, we show that the AR model with $D_s = 16$ & $D_r = 2048$ is as accurate as single-shot retrieval with $d = 2048$ while being $\sim \mathbf{128}\times$ more efficient in theory and $\sim \mathbf{14}\times$ faster in practice (compared using HNSW on the same hardware). We show similar trends with ImageNet-4K, but note that we require $D_s = 64$ given the increased difficulty of the dataset. This results in $\sim 32\times$ and $\sim 6\times$ theoretical and in-practice speedups respectively. Lastly, while $K = 200$ works well for our adaptive retrieval experiments, we ablated over the shortlist size k in Appendix K.2 of Kusupati et al. [147] and found that the accuracy gains stopped after a point, further strengthening the use-case for Matryoshka Representation Learning and adaptive retrieval.

Even with adaptive retrieval, it is hard to determine the choice of D_s & D_r. In order to alleviate this issue to an extent, we propose **Funnel Retrieval**, a consistent cascade for adaptive retrieval. Funnel thins out the

initial shortlist by a repeated re-ranking and shortlisting with a series of increasing capacity representations. Funnel halves the shortlist size and doubles the representation size at every step of re-ranking. For example on ImageNet-1K, a funnel with the shortlist progression of $200 \rightarrow 100 \rightarrow 50 \rightarrow 25 \rightarrow 10$ with the cascade of $16 \rightarrow 32 \rightarrow 64 \rightarrow 128 \rightarrow 256 \rightarrow 2048$ representation sizes within Matryoshka Representation is as accurate as the single-shot 2048-dim retrieval while being $\sim 128\times$ more efficient theoretically (see Appendix F of Kusupati et al. [147] for more results). All these results showcase the potential of MRL and AR for large-scale multi-stage search systems [51].

2.6 Further Analysis and Ablations

Robustness. We evaluate the robustness of the MRL models trained on ImageNet-1K on out-of-domain datasets, ImageNetV2/R/A/Sketch [207, 106, 107, 254], and compare them to the FF baselines. Table 17 in Appendix H of Kusupati et al. [147] demonstrates that Matryoshka Representations for classification are at least as robust as the original representation while improving the performance on ImageNet-A by 0.6% – a 20% relative improvement. We also study the robustness in the context of retrieval by using ImageNetV2 as the query set for ImageNet-1K database. Table 9 in Appendix E of Kusupati et al. [147] shows that MRL models have more robust retrieval compared to the FF baselines by having up to 3% higher mAP@10 performance. This observation also suggests the need for further investigation into robustness using nearest neighbour based classification and retrieval instead of the standard linear probing setup. We also find that the zero-shot robustness of ALIGN-MRL (Table 18 in Appendix H of Kusupati et al. [147]) agrees with the observations made by Wortsman et al. [267]. Lastly, Table 6 in Appendix D.2 of Kusupati et al. [147] shows that MRL also improves the cosine similarity span between positive and random image-text pairs.

Few-shot and Long-tail Learning. We exhaustively evaluated few-shot learning on MRL models using nearest class mean [219]. Table 15 in Appendix G of Kusupati et al. [147] shows that that representations learned through MRL perform comparably to FF representations across varying shots and number of classes.

Matryoshka Representations realize a unique pattern while evaluating on FLUID [249], a long-tail sequential learning framework. We observed that MRL provides up to 2% accuracy higher on novel classes in the tail of the distribution, without sacrificing accuracy on other classes (Table 16 in Appendix G

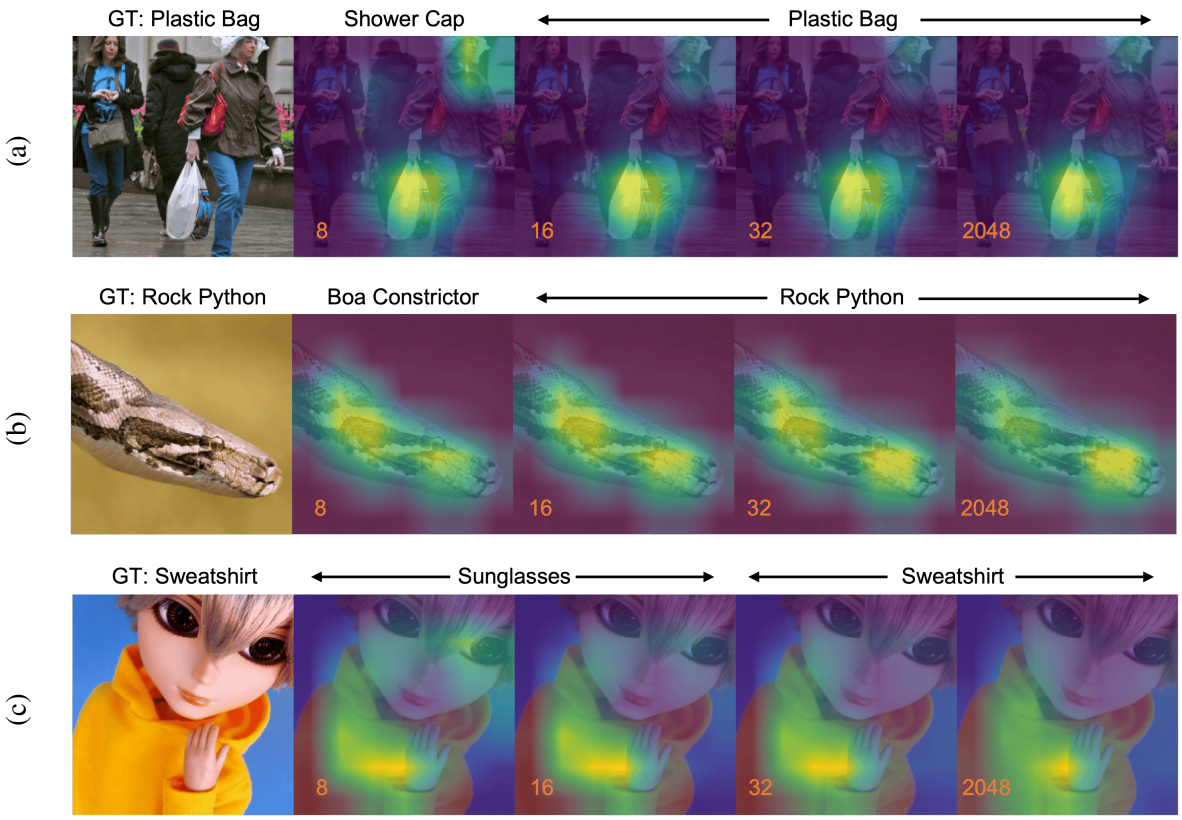


Figure 2.9: Grad-CAM [224] progression of predictions in MRL model across $8, 16, 32$ and 2048 dimensions. (a) 8-dimensional representation confuses due to presence of other relevant objects (with a larger field of view) in the scene and predicts "shower cap" ; (b) 8-dim model confuses within the same super-class of "boa" ; (c) 8 and 16-dim models incorrectly focus on the eyes of the doll ("sunglasses") and not the "sweatshirt" which is correctly in focus at higher dimensions; MRL fails gracefully in these scenarios and shows potential use cases of disagreement across dimensions.

of Kusupati et al. [147]). Additionally we find the accuracy between low-dimensional and high-dimensional representations is marginal for pretrain classes. We hypothesize that the higher-dimensional representations are required to differentiate the classes when few training examples of each are known. This results provides further evidence that different tasks require varying capacity based on their difficulty.

Disagreement across Dimensions. The information packing in Matryoshka Representations often results in gradual increase of accuracy with increase in capacity. However, we observed that this trend was not ubiquitous and certain instances and classes were more accurate when evaluated with lower-dimensions (Figure 12 in Appendix J of Kusupati et al. [147]). With perfect routing of instances to appropriate dimension, MRL can gain up to 4.6% classification accuracy. At the same time, the low-dimensional models are less

accurate either due to confusion within the same superclass [71] of the ImageNet hierarchy or presence of multiple objects of interest. Figure 2.9 showcases 2 such examples for 8-dimensional representation. These results along with Appendix J of Kusupati et al. [147] put forward the potential for MRL to be a systematic framework for analyzing the utility and efficiency of information bottlenecks.

Superclass Accuracy. As the information bottleneck becomes smaller, the overall accuracy on fine-grained classes decreases rapidly (Figure 2.3). However, the drop-off is not as significant when evaluated at a superclass level (Table 24 in Appendix J of Kusupati et al. [147]). Figure 2.10 presents that this phenomenon occurs with both MRL and FF models; MRL is more accurate across dimensions. This shows that tight information bottlenecks while not highly accurate for fine-grained classification, do capture required semantic information for coarser classification that could be leveraged for adaptive routing for retrieval and classification. Mutifidelity of Matryoshka Representation naturally captures the underlying hierarchy of the class labels with one single model. Lastly, Figure 2.11 showcases the accuracy trends per superclass with MRL. The utility of additional dimensions in distinguishing a class from others within the same superclass is evident for "garment" which has up to 11% improvement for $8 \rightarrow 16$ dimensional representation transition. We also observed that superclasses such as "oscine (songbird)" had a clear visual distinction between the object and background and thus predictions using 8 dimensions also led to a good inter-class separability within the superclass.

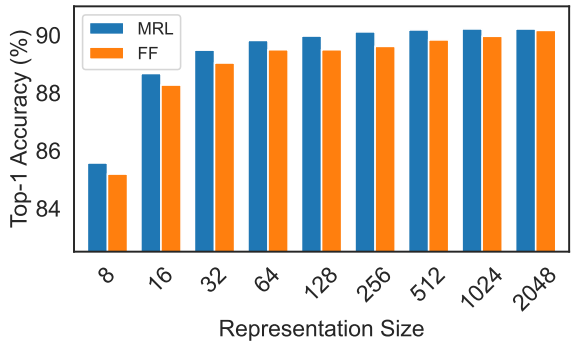


Figure 2.10: 31-way ImageNet-1K superclass classification across representation size for MRL & FF models showing the capture of underlying hierarchy through tight information bottlenecks.

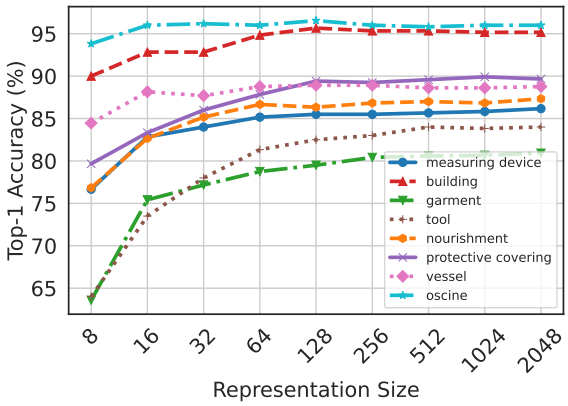


Figure 2.11: Diverse per-superclass accuracy trends across representation sizes for ResNet50-MRL on ImageNet-1K.

2.6.1 Ablations

Table 26 in Appendix K of Kusupati et al. [147] presents that Matryoshka Representations can be enabled within off-the-shelf pretrained models with inexpensive partial finetuning thus paving a way for ubiquitous adoption of MRL. At the same time, Table 27 in Appendix C of Kusupati et al. [147] indicates that with optimal weighting of the nested losses we could improve accuracy of lower-dimensions representations without accuracy loss. Tables 28 and 29 in Appendix C of Kusupati et al. [147] ablate over the choice of initial granularity and spacing of the granularites. Table 28 reaffirms the design choice to shun extremely low dimensions that have poor classification accuracy as initial granularity for MRL while Table 29 confirms the effectiveness of logarthmic granularity spacing inspired from the behaviour of accuracy saturation across dimensions over uniform. Lastly, Tables 30 and 31 in Appendix K.2 show that the retrieval performance saturates after a certain shortlist dimension and length depending on the complexity of the dataset.

2.7 Discussion and Conclusions

The results in Section 2.6.1 reveal interesting weaknesses of MRL that would be logical directions for future work. (1) Optimizing the weightings of the nested losses to obtain a Pareto optimal accuracy-vs-efficiency trade-off – a potential solution could emerge from adaptive loss balancing aspects of anytime neural networks [114]. (2) Using different losses at various fidelities aimed at solving a specific aspect of adaptive deployment – e.g. high recall for 8-dimension and robustness for 2048-dimension. (3) Learning a search data-structure, like differentiable k-d tree, on top of Matryoshka Representation to enable dataset and representation aware retrieval. (4) Finally, the joint optimization of multi-objective MRL combined with end-to-end learnable search data-structure to have data-driven adaptive large-scale retrieval for web-scale search applications.

In conclusion, we presented Matryoshka Representation Learning (MRL), a flexible representation learning approach that encodes information at multiple granularities in a single embedding vector. This enables the MRL to adapt to a downstream task's statistical complexity as well as the available compute resources. We demonstrate that MRL can be used for large-scale adaptive classification as well as adaptive retrieval. On standard benchmarks, MRL matches the accuracy of the fixed-feature baseline despite using

$14\times$ smaller representation size on average. Furthermore, the $\mathrm{Matryoshka\ Representation}$ based adaptive shortlisting and re-ranking system ensures comparable mAP@10 to the baseline while being $128\times$ cheaper in FLOPs and $14\times$ faster in wall-clock time. Finally, most of the efficiency techniques for model inference and vector search are complementary to MRL 🪆 further assisting in deployment at the compute-extreme environments.

Chapter 3

AdANNS: A Framework for Adaptive Semantic Search

3.1 Overview

Web-scale search systems learn an encoder to embed a given query which is then hooked into an approximate nearest neighbor search (ANNS) pipeline to retrieve similar data points. To accurately capture tail queries and data points, learned representations typically are *rigid, high-dimensional* vectors that are generally used as-is in the entire ANNS pipeline and can lead to computationally expensive retrieval. In this chapter, we argue that instead of rigid representations, different stages of ANNS can leverage *adaptive representations* of varying capacities to achieve significantly better accuracy-compute trade-offs, i.e., stages of ANNS that can get away with more approximate computation should use a lower-capacity representation of the same data point. To this end, we introduce AdANNS, a novel ANNS design framework that explicitly leverages the flexibility of Matryoshka Representations [147] introduced in Chapter 2. We demonstrate state-of-the-art accuracy-compute trade-offs using novel AdANNS-based key ANNS building blocks like search data structures (AdANNS-IVF) and quantization (AdANNS-OPQ). For example on ImageNet retrieval, AdANNS-IVF is up to $\mathbf{1.5}\%$ more accurate than the rigid representations-based IVF [230] at the same compute budget; and matches accuracy while being up to $\mathbf{90}\times$ faster in *wall-clock time*. For Natural Questions, 32-byte AdANNS-OPQ matches the accuracy of the 64-byte OPQ baseline [80] constructed using rigid representations – *same*

accuracy at half the cost! We further show that the gains from AdANNS translate to modern-day composite ANNS indices that combine search structures and quantization. Finally, we demonstrate that AdANNS can enable inference-time adaptivity for compute-aware search on ANNS indices built non-adaptively on matryoshka representations. Code is open-sourced at https://github.com/RAIVNLab/AdANNS.

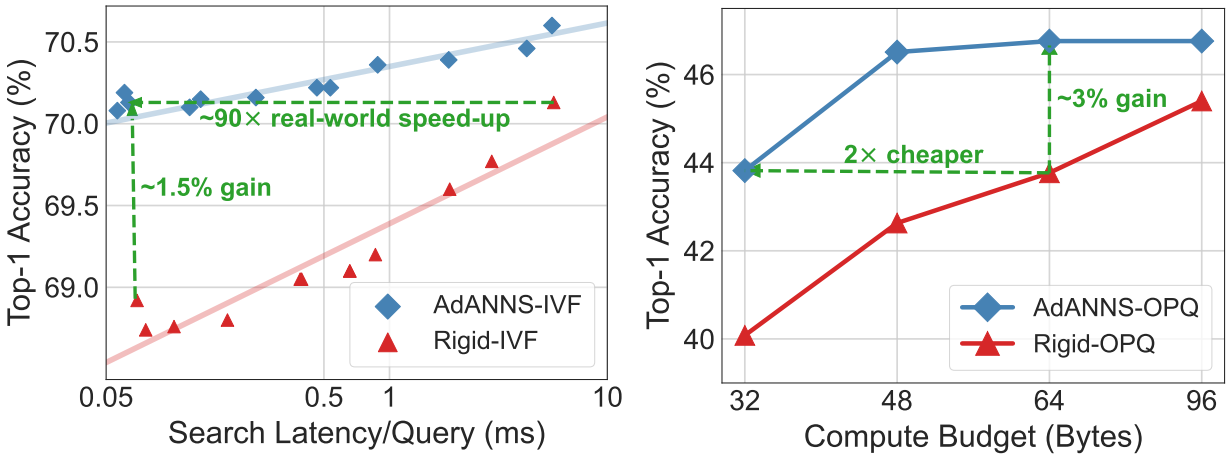
3.2 Introduction

Semantic search [128] on learned representations [190, 191, 248] is a major component in retrieval pipelines [26, 51]. In its simplest form, semantic search methods learn a neural network to embed queries as well as a large number (N) of data points in a d-dimensional vector space. For a given query, the nearest (in embedding space) point is retrieved using either an exact search or using approximate nearest neighbor search (ANNS) [118] which is now indispensable for real-time large-scale retrieval.

Existing semantic search methods learn fixed or *rigid* representations (RRs), also referred to as fixed features (FFs) previously, which are used as is in all the stages of ANNS (data structures for data pruning and quantization for cheaper distance computation; see Section 3.3). That is, while ANNS indices allow a variety of parameters for searching the design space to optimize the accuracy-compute trade-off, the provided data dimensionality is typically assumed to be an *immutable* parameter. To make it concrete, let us consider inverted file index (IVF) [230], a popular web-scale ANNS technique [89]. IVF has two stages (Section 3.4) during inference: (a) *cluster mapping*: mapping the query to a cluster of data points [172], and (b) *linear scan*: distance computation w.r.t all points in the retrieved cluster to find the nearest neighbor (NN). Standard IVF utilizes the same high-dimensional RR for both phases, which can be sub-optimal.

Why the sub-optimality? Imagine one needs to partition a dataset into k clusters for IVF and the dimensionality of the data is d – IVF uses full d representation to partition into k clusters. However, suppose we have an alternate approach that somehow projects the data in $d/2$ dimensions and learns $2k$ clusters. Note that the storage and computation to find the nearest cluster remains the same in both cases, i.e., when we have k clusters of d dimensions or $2k$ clusters of $d/2$ dimensions. $2k$ clusters can provide significantly more refined partitioning, but the distances computed between queries and clusters could be significantly more inaccurate after projection to $d/2$ dimensions.

So, if we can find a mechanism to obtain a $d/2$-dimensional representation of points that can accurately



(a) Image retrieval on ImageNet-1K.

(b) Passage retrieval on Natural Questions.

Figure 3.1: AdANNS helps design search data structures and quantization methods with *better accuracy-compute trade-offs* than the existing solutions. In particular, (a) AdANNS-IVF improves on standard IVF by up to 1.5% in accuracy while being $90\times$ faster in deployment and (b) AdANNS-OPQ is as accurate as the baseline at *half the cost!* Rigid-IVF and Rigid-OPQ are standard techniques that are built on rigid representations (RRs) while AdANNS uses matryoshka representations (MRs) [147].

approximate the topology/distances of d-dimensional representation, then we can potentially build significantly better ANNS structure that utilizes different capacity representations for the cluster mapping and linear scan phases of IVF. But how do we find such *adaptive representations*? These desired adaptive representations should be cheap to obtain and still ensure distance preservation across dimensionality. Post-hoc dimensionality reduction techniques like SVD [82] and random projections [129] on high-dimensional RRs are potential candidates, but our experiments indicate that in practice they are highly inaccurate and do not preserve distances well enough (Figure 3.2).

Instead, we identify that the recently proposed Matryoshka Representations (MRs) [147] satisfy the specifications for adaptive representations. Matryoshka representations pack information in a hierarchical nested manner, i.e., the first m-dimensions of the d-dimensional MR form an accurate low-dimensional representation while being aware of the information in the higher dimensions. This allows us to deploy MRs in two major and novel ways as part of ANNS: (a) low-dimensional representations for accuracy-compute optimal clustering and quantization, and (b) high-dimensional representations for precise re-ranking when feasible.

To this effort, we introduce AdANNS, a novel design framework for semantic search that uses matryoshka representation-based *adaptive representations* across different stages of ANNS to ensure significantly

better accuracy-compute trade-off than the state-of-the-art baselines.

Typical ANNS systems have two key components: (a) search data structure to store datapoints, (b) distance computation to map a given query to points in the data structure. Through AdANNS, we address both these components and significantly improve their performance. In particular, we first propose AdANNS-IVF (Section 3.5.1) which tackles the first component of ANNS systems. AdANNS-IVF uses standard full-precision computations but uses adaptive representations for different IVF stages. On ImageNet 1-NN image retrieval (Figure 3.1a), AdANNS-IVF is up to 1.5% more accurate for the compute budget and $90\times$ cheaper in deployment for the same accuracy as IVF.

We then propose AdANNS-OPQ (Section 3.5.2) which addresses the second component by using AdANNS-based quantization (OPQ [80]) – here we use exhaustive search overall points. AdANNS-OPQ is as accurate as the baseline OPQ on RRs while being at least $\mathbf{2}\times$ faster on Natural Questions [148] 1-NN passage retrieval (Figure 3.1b). Finally, we combine the two techniques to obtain AdANNS-IVFOPQ (Section 3.5.3) which is more accurate while being much cheaper – up to $\mathbf{8}\times$ – than the traditional IVFOPQ [128] index. To demonstrate generality of our technique, we adapt AdANNS to DiskANN [125] which provides interesting accuracy-compute tradeoff; see Table 3.1.

While MR already has multi-granular representations, careful integration with ANNS building blocks is critical to obtain a practical method and is *our main contribution*. In fact, Kusupati et al. [147] proposed a simple adaptive retrieval setup that uses smaller-dimensional MR for shortlisting in retrieval followed by precise re-ranking with a higher-dimensional MR. Such techniques, unfortunately, cannot be scaled to industrial systems as they require forming a new index for every shortlisting provided by low-dimensional MR. Ensuring that the method aligns well with the modern-day ANNS pipelines is important as they already have mechanisms to handle real-world constraints like load-balancing [89] and random access from disk [125]. So, AdANNS is a step towards making the abstraction of adaptive search and retrieval feasible at the web-scale.

Through extensive experimentation, we also show that AdANNS generalizes across search data structures, distance approximations, modalities (text & image), and encoders (CNNs & Transformers) while still translating the theoretical gains to latency reductions in deployment. While we have mainly focused on IVF and OPQ-based ANNS in this work, AdANNS also blends well with other ANNS pipelines. We

also show that AdANNS can enable compute-aware elastic search on prebuilt indices without making any modifications (Section 3.6.1); note that this is in contrast to AdANNS-IVF that builds the index explicitly utilizing "adaptivity" in representations. Finally, we provide an extensive analysis on the alignment of matryoshka representation for better semantic search (Section 3.6.2).

We make the following key contributions:

- We introduce AdANNS, a novel framework for semantic search that leverages matryoshka representations for designing ANNS systems with better accuracy-compute trade-offs.
- AdANNS powered search data structure (AdANNS-IVF) and quantization (AdANNS-OPQ) show a significant improvement in accuracy-compute tradeoff compared to existing solutions.
- AdANNS generalizes to modern-day composite ANNS indices and can also enable compute-aware elastic search during inference with no modifications.

3.3 Related Work

Approximate nearest neighbour search (ANNS) is a paradigm to come as close as possible [46] to retrieving the "true" nearest neighbor (NN) without the exorbitant search costs associated with exhaustive search [118, 260]. The "approximate" nature comes from data pruning as well as the cheaper distance computation that enable real-time web-scale search. In its naive form, NN-search has a complexity of $\mathcal{O}(dN)$; d is the data dimensionality used for distance computation and N is the size of the database. ANNS employs each of these approximations to reduce the linear dependence on the dimensionality (cheaper distance computation) and data points visited during search (data pruning).

Cheaper distance computation. From a bird's eye view, cheaper distance computation is always obtained through dimensionality reduction (quantization included). PCA and SVD [82, 130] can reduce dimensionality and preserve distances only to a limited extent without sacrificing accuracy. On the other hand, quantization-based techniques [42, 86] like (optimized) product quantization ((O)PQ) [80, 126] have proved extremely crucial for relatively accurate yet cheap distance computation and simultaneously reduce the memory overhead significantly. Another naive solution is to independently train the representation function with varying low-dimensional information bottlenecks [147] which is rarely used due to the costs of maintaining multiple models and databases.

Data pruning. Enabled by various data structures, data pruning reduces the number of data points visited as part of the search. This is often achieved through hashing [50, 217], trees [20, 78, 89, 230] and graphs [125, 178]. More recently there have been efforts towards end-to-end learning of the search data structures [91, 136, 146]. However, web-scale ANNS indices are often constructed on rigid d-dimensional real vectors using the aforementioned data structures that assist with the real-time search. For a more comprehensive review of ANNS structures please refer to [29, 163, 257].

Composite indices. ANNS pipelines often benefit from the complementary nature of various building blocks [128, 202]. In practice, often the data structures (coarse-quantizer) like IVF [230] and HNSW [177] are combined with cheaper distance alternatives like PQ [126] (fine-quantizer) for massive speed-ups in web-scale search. While the data structures are built on d-dimensional real vectors, past works consistently show that PQ can be safely used for distance computation during search time. As evident in modern web-scale ANNS systems like DiskANN [125], the data structures are built on d-dimensional real vectors but work with PQ vectors ($32 - 64$-byte) for fast distance computations.

ANNS benchmark datasets. Despite the Herculean advances in representation learning [98, 202], ANNS progress is often only benchmarked on fixed representation vectors provided for about a dozen million to billion scale datasets [11, 228] with limited access to the raw data. This resulted in the improvement of algorithmic design for rigid representations (RRs) that are often not specifically designed for search. All the existing ANNS methods work with the assumption of using the provided d-dimensional representation which might not be Pareto-optimal for the accuracy-compute trade-off in the first place. Note that the lack of raw-image and text-based benchmarks led us to using ImageNet-1K [215] (1.3M images, 50K queries) and Natural Questions [148] (21M passages, 3.6K queries) for experimentation. While not billion-scale, the results observed on ImageNet often translate to real-world progress [135], and Natural Questions is one of the largest question answering datasets benchmarked for dense passage retrieval [133], making our results generalizable and widely applicable.

In this chapter, we investigate the utility of adaptive representations – embeddings of different dimensionalities having similar semantic information – in improving the design of ANNS algorithms. This helps in transitioning out of restricted construction and inference on rigid representations for ANNS. To this end, we extensively use Matryoshka Representations (MRs) [147] which have desired adaptive properties in-built.

To the best of our knowledge, this is the first work that improves accuracy-compute trade-off in ANNS by leveraging adaptive representations on different phases of construction and inference for ANNS data structures.

3.4 Problem Setup, Notation, and Preliminaries

The problem setup of approximate nearest neighbor search (ANNS) [118] consists of a database of N data points, $[x_1, x_2, \ldots, x_N]$, and a query, q, where the goal is to "approximately" retrieve the nearest data point to the query. Both the database and query are embedded to $\mathbb{R}^d$ using a representation function $\phi : \mathcal{X} \rightarrow \mathbb{R}^d$, often a neural network that can be learned through various representation learning paradigms [17, 98, 99, 190, 202].

Matryoshka Representations (MRs)**.** The d-dimensional representations from ϕ can have a nested structure like Matryoshka Representations (MRs) [147] in-built – $\phi^{\text{MR}(d)}$. Matryoshka Representation Learning (MRL) learns these nested representations with a simple strategy of optimizing the same training objective at varying dimensionalities. These granularities are ordered such that the lowest representation size forms a prefix for the higher-dimensional representations. So, high-dimensional MR inherently contains low-dimensional representations of varying granularities that can be accessed for free – first m-dimensions ($m \in [d]$) ie., $\phi^{\text{MR}(d)}[1:m]$ from the d-dimensional MR form an m-dimensional representation which is as accurate as its independently trained rigid representation (RR) counterpart – $\phi^{\text{RR}(m)}$. Training an encoder with MRL does not involve any overhead or hyperparameter tuning and works seamlessly across modalities, training objectives, and architectures.

Inverted File Index (IVF). IVF [230] is an ANNS data structure used in web-scale search systems [89] owing to its simplicity, minimal compute overhead, and high accuracy. IVF construction involves clustering (coarse quantization through k-means) [172] on d-dimensional representation that results in an inverted file list [265] of all the data points in each cluster. During search, d-dimensional query representation is assigned to the most relevant cluster (C_i; $i \in [k]$) by finding the closest centroid (μ_i) using an appropriate distance metric (L_2 or cosine). This is followed by an exhaustive linear search across all data points in the cluster which gives the closest NN (see Figure 5 in Appendix A of Rege et al. [209] for IVF overview). Lastly, IVF

can scale to web-scale by utilizing a hierarchical IVF structure within each cluster [89]. Table2 in Appendix A of Rege et al. [209] describes the retrieval formula for multiple variants of IVF.

Optimized Product Quantization (OPQ). Product Quantization (PQ) [126] works by splitting a d-dimensional real vector into m sub-vectors and quantizing each sub-vector with an independent 2^b length codebook across the database. After PQ, each d-dimensional vector can be represented by a compact $m \times b$ bit vector; we make each vector m bytes long by fixing $b = 8$. During search time, distance computation between the query vector and PQ database is extremely efficient with only m codebook lookups. The generality of PQ encompasses scalar/vector quantization [86, 172] as special cases. However, PQ can be further improved by rotating the d-dimensional space appropriately to maximize distance preservation after PQ. Optimized Product Quantization (OPQ) [80] achieves this by learning an orthonormal projection matrix R that rotates the d-dimensional space to be more amenable to PQ. OPQ shows consistent gains over PQ across a variety of ANNS tasks and has become the default choice in standard composite indices [125, 128].

Datasets. We evaluate the ANNS algorithms while changing the representations used for the search thus making it impossible to evaluate on the usual benchmarks [11]. Hence we experiment with two public datasets: (a) ImageNet-1K [215] dataset on the task of image retrieval – where the goal is to retrieve images from a database (1.3M image train set) belonging to the same class as the query image (50K image validation set) and (b) Natural Questions (NQ) [148] dataset on the task of question answering through dense passage retrieval – where the goal is to retrieve the relevant passage from a database (21M Wikipedia passages) for a query (3.6K questions).

Metrics Performance of ANNS is often measured using recall score [125], k-recall@N – recall of the exact NN across search complexities which denotes the recall of k "true" NN when N data points are retrieved. However, the presence of labels allows us to compute 1-NN (top-1) accuracy. Top-1 accuracy is a harder and more fine-grained metric that correlates well with typical retrieval metrics like recall and mean average precision (mAP@k). Even though we report top-1 accuracy by default during experimentation, we discuss other metrics in Appendix C of Rege et al. [209]. Finally, we measure the compute overhead of ANNS using MFLOPS/query and also provide wall-clock times (see Appendix B.1 of Rege et al. [209]).

Encoders. For ImageNet, we encode both the database and query set using a ResNet50 (ϕ_I) [98] trained on ImageNet-1K. For NQ, we encode both the passages in the database and the questions in the query set using a BERT-Base (ϕ_N) [61] model fine-tuned on NQ for dense passage retrieval [133].

We use the trained ResNet50 models with varying representation sizes ($d = [8, 16, \ldots, 2048]$; default being 2048) as suggested by Kusupati et al. [147] alongside the MRL-ResNet50 models trained with MRL for the same dimensionalities. The RR and MR models are trained to ensure the supervised one-vs-all classification accuracy across all data dimensionalities is nearly the same – 1-NN accuracy of 2048-d RR and MR models are 71.19% and 70.97% respectively on ImageNet-1K. Independently trained models, $\phi_I^{\mathrm{RR}(d)}$, output $d = [8, 16 \ldots, 2048]$ dimensional RRs while a single MRL-ResNet50 model, $\phi_I^{\mathrm{MR}(d)}$, outputs a $d = 2048$-dimensional MR that contains all the 9 granularities.

We also train BERT-Base models in a similar vein as the aforementioned ResNet50 models. The key difference is that we take a pre-trained BERT-Base model and fine-tune on NQ as suggested by Karpukhin et al. [133] with varying (5) representation sizes (bottlenecks) ($d = [48, 96, \ldots, 768]$; default being 768) to obtain $\phi_N^{\mathrm{RR}(d)}$ that creates RRs for the NQ dataset. To get the MRL-BERT-Base model, we fine-tune a pre-trained BERT-Base encoder on the NQ train dataset using the MRL objective with the same granularities as RRs to obtain $\phi_N^{\mathrm{MR}(d)}$ which contains all five granularities. Akin to ResNet50 models, the RR and MR BERT-Base models on NQ are built to have similar 1-NN accuracy for 768-d of 52.2% and 51.5% respectively. More implementation details can be found in Appendix B of Rege et al. [209] and additional experiment-specific information is provided at the appropriate places.

3.5 AdANNS – Adaptive ANNS

In this section, we present our proposed AdANNS framework that exploits the inherent flexibility of matryoshka representations to improve the accuracy-compute trade-off for semantic search components. Standard ANNS pipeline can be split into two key components: (a) search data structure that indexes and stores data points, (b) query-point computation method that outputs (approximate) distance between a given query and data point. For example, standard IVFOPQ [128] method uses an IVF structure to index points on full-precision vectors and then relies on OPQ for more efficient distance computation between the query and the data points during the linear scan.

Below, we show that AdANNS can be applied to both the above-mentioned ANNS components and provides significant gains on the computation-accuracy tradeoff curve. In particular, we present AdANNS-IVF which is AdANNS version of the standard IVF index structure [230], and the closely related ScaNN structure [89]. We also present AdANNS-OPQ which introduces representation adaptivity in the OPQ, an industry-default quantization. Then, in Section 3.5.3 we further demonstrate the combination of the two techniques to get AdANNS-IVFOPQ – an AdANNS version of IVFOPQ [128] – and AdANNS-DiskANN, a similar variant of DiskANN [125]. Overall, our experiments show that AdANNS-IVF is significantly more accuracy-compute optimal compared to the IVF indices built on RRs and AdANNS-OPQ is as accurate as the OPQ on RRs while being significantly cheaper.

3.5.1 AdANNS-IVF

Recall from Section 3.2 that IVF has a clustering and a linear scan phase, where both phase use same dimensional rigid representation. Now, AdANNS-IVF allows the clustering phase to use the first d_c dimensions of the given matryoshka representation (MR). Similarly, the linear scan within each cluster uses d_s dimensions, where again d_s represents top d_s coordinates from MR. Note that setting $d_c = d_s$ results in non-adaptive regular IVF. Intuitively, we would set $d_c \ll d_s$, so that instead of clustering with a high-dimensional representation, we can approximate it accurately with a low-dimensional embedding of size d_c followed by a linear scan with a higher d_s-dimensional representation. Intuitively, this helps in the smooth search of design space for state-of-the-art accuracy-compute trade-off. Furthermore, this can provide a precise operating point on accuracy-compute

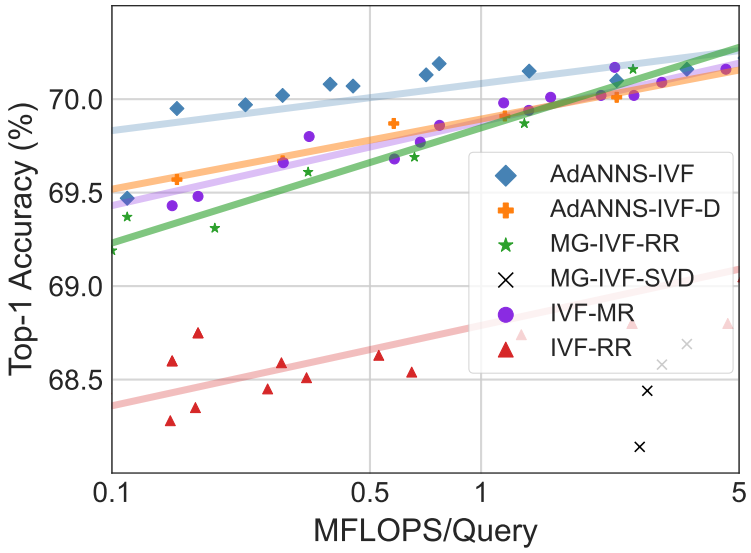


Figure 3.2: 1-NN accuracy on ImageNet retrieval shows that AdANNS-IVF achieves near-optimal accuracy-compute trade-off compared across various rigid and adaptive baselines. Both adaptive variants of MR and RR significantly outperform their rigid counterparts (IVF-XX) while post-hoc compression on RR using SVD for adaptivity falls short.

tradeoff curve which is critical in several practical settings.

Our experiments on regular IVF with MRs and RRs (IVF-MR & IVF-RR) of varying dimensionalities and IVF configurations (# clusters, # probes) show that (Figure 3.2) matryoshka representations result in a significantly better accuracy-compute trade-off. We further studied and found that learned lower-dimensional representations offer better accuracy-compute trade-offs for IVF than higher-dimensional embeddings (see Appendix E of Rege et al. [209] for more results).

AdANNS utilizes d-dimensional matryoshka representation to get accurate d_c and d_s dimensional vectors at no extra compute cost. The resulting AdANNS-IVF provides a much better accuracy-compute trade-off (Figure 3.2) on ImageNet-1K retrieval compared to IVF-MR, IVF-RR, and MG-IVF-RR – multi-granular IVF with rigid representations (akin to AdANNS without MR) – a strong baseline that uses d_c and d_s dimensional RRs. Finally, we exhaustively search the design space of IVF by varying $d_c, d_s \in [8, 16, \ldots, 2048]$ and the number of clusters $k \in [8, 16, \ldots, 2048]$. Please see Appendix E of Rege et al. [209] for more details. For IVF experiments on the NQ dataset, please refer to Appendix G of Rege et al. [209].

Empirical results. Figure 3.2 shows that AdANNS-IVF outperforms the baselines across all accuracy-compute settings for ImageNet-1K retrieval. AdANNS-IVF results in $10\times$ lower compute for the best accuracy of the extremely expensive MG-IVF-RR and non-adaptive IVF-MR. Specifically, as shown in Figure 3.1a, AdANNS-IVF is up to 1.5% more accurate for the same compute and has up to $100\times$ lesser FLOPS/query ($90\times$ real-world speed-up!) than the status quo ANNS on rigid representations (IVF-RR). We filter out points for the sake of presentation and encourage the reader to check out Figure 8 in Appendix E of Rege et al. [209] for an expansive plot of all the configurations searched.

The advantage of AdANNS for construction of search structures is evident from the improvements in IVF (AdANNS-IVF) and can be easily extended to other ANNS structures like ScaNN [89] and HNSW [178]. For example, HNSW consists of multiple layers with graphs of NSW graphs [177] of increasing complexity. AdANNS can be adopted to HNSW, where the construction of each level can be powered by appropriate dimensionalities for an optimal accuracy-compute trade-off. In general, AdANNS provides fine-grained control over compute overhead (storage, working memory, inference, and construction cost) during construction and inference while providing the best possible accuracy.

3.5.2 AdANNS-OPQ

Standard Product Quantization (PQ) essentially performs block-wise vector quantization via clustering. For example, suppose we need 32-byte PQ compressed vectors from the given 2048 dimensional representations. Then, we can chunk the representations in $m = 32$ equal blocks/sub-vectors of 64-d each, and each sub-vector space is clustered into $2^8 = 256$ partitions. That is, the representation of each point is essentially cluster-id for each block. Optimized PQ (OPQ) [80] further refines this idea, by first rotating the representations using a learned orthogonal matrix, and then applying PQ on top of the rotated representations. In ANNS, OPQ is used extensively to compress vectors and improves approximate distance computation primarily due to significantly lower memory overhead than storing full-precision data points IVF.

AdANNS-OPQ utilizes MR representations to apply OPQ on lower-dimensional representations. That is, for a given quantization budget, AdANNS allows using top $d_s \ll d$ dimensions from MR and then computing clusters with d_s/m-dimensional blocks where m is the number of blocks. Depending on d_s and m, we have further flexibility of trading-off dimensionality/capacity for increasing the number of clusters to meet the given quantization budget. AdANNS-OPQ tries multiple d_s, m, and number of clusters for a fixed quantization budget to obtain the best performing configuration.

We experimented with $8 - 128$ byte OPQ budgets for both ImageNet and Natural Questions retrieval with an exhaustive search on the quantized vectors. We compare AdANNS-OPQ which uses MRs of varying granularities to the baseline OPQ built on the highest dimensional RRs. We also evaluate OPQ vectors obtained projection using SVD [82] on top of the highest-dimensional RRs.

Empirical results. Figures 3.3 and 3.1b show that AdANNS-OPQ significantly outperforms – up to 4% accuracy gain – the baselines (OPQ on RRs) across compute budgets on both ImageNet and NQ. In particular, AdANNS-OPQ tends to match the accuracy of a 64-byte (a typical choice in ANNS) OPQ baseline with only a 32-byte budget. This results in a $2\times$ reduction in both storage and compute FLOPS which translates to significant gains in real-world web-scale deployment (see Appendix D of Rege et al. [209]).

We only report the best AdANNS-OPQ for each budget typically obtained through a much lower-dimensional MR (128 & 192; much faster to build as well) than the highest-dimensional MR (2048 & 768) for ImageNet and NQ respectively (see Appendix G of Rege et al. [209] for more details). At the same time,

we note that building compressed OPQ vectors on projected RRs using SVD to the smaller dimensions (or using low-dimensional RRs, see Appendix D of Rege et al. [209]) as the optimal AdANNS-OPQ does not help in improving the accuracy. The significant gains we observe in AdANNS-OPQ are purely due to better information packing in MRs – we hypothesize that packing the most important information in the initial coordinates results in a better PQ quantization than RRs where the information is uniformly distributed across all the dimensions [147, 232]. See Appendix D of Rege et al. [209] for more details and experiments.

3.5.3 AdANNS for Composite Indices

We now extend AdANNS to composite indices [128] which put together two main ANNS building blocks – search structures and quantization – together to obtain efficient web-scale ANNS indices used in practice. A simple instantiation of a composite index would be the combination of IVF and OPQ – IVFOPQ – where the clustering in IVF happens with full-precision real vectors but the linear scan within each cluster is approximated using OPQ-compressed variants of the representation – since often the full-precision vectors of the database cannot fit in RAM. Contemporary ANNS indices like DiskANN [125] make this a default choice where they build the search graph with a full-precision vector and approximate the distance computations during search with an OPQ-compressed vector to obtain a very small shortlist of retrieved datapoints. In

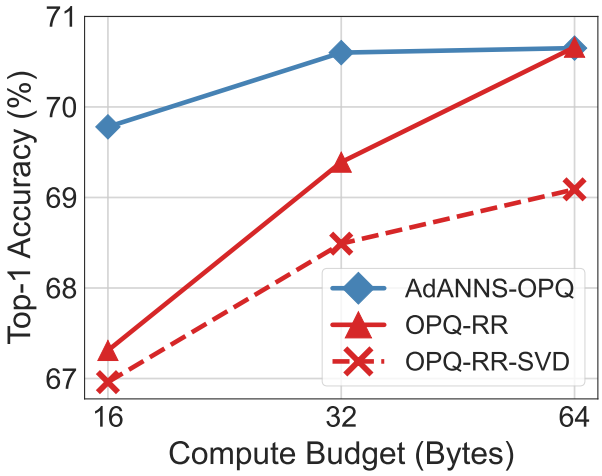


Figure 3.3: AdANNS-OPQ matches the accuracy of 64-byte OPQ on RR using only 32-bytes for ImageNet retrieval. AdANNS provides large gains at lower compute budgets and saturates to baseline performance for larger budgets.

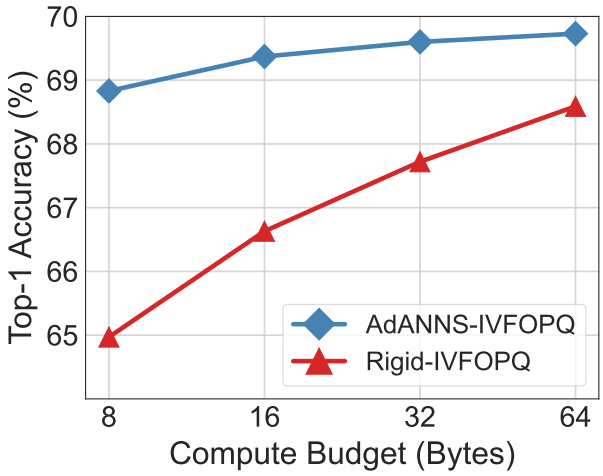


Figure 3.4: Combining the gains of AdANNS for IVF and OPQ leads to better IVFOPQ composite indices. On ImageNet retrieval, AdANNS-IVFOPQ is 8× cheaper for the same accuracy and provides 1 - 4% gains over IVFOPQ on RRs.

DiskANN, the shortlist of data points is then re-ranked to form the final list using their full-precision vectors fetched from the disk. AdANNS is naturally suited to this shortlist-rerank framework: we use a low-d MR for forming index, where we could tune AdANNS parameters according to the accuracy-compute trade-off of the graph and OPQ vectors. We then use a high-d MR for re-ranking.

Empirical results. Figure 3.4 shows that AdANNS-IVFOPQ is $1-4\%$ better than the baseline at all the PQ compute budgets. Furthermore, AdANNS-IVFOPQ has the same accuracy as the baselines at $8\times$ lower overhead. With DiskANN, AdANNS accelerates shortlist generation by using low-dimensional representations and recoups the accuracy by re-ranking with the highest-dimensional MR at negligible cost. Table 3.1 shows that AdANNS-DiskANN is more accurate than the baseline for both 1-NN and ranking performance at only $half$ the cost. Using low-dimensional representations further speeds up inference in AdANNS-DiskANN (see Appendix F of Rege et al. [209]).

Table 3.1: AdANNS-DiskANN using a 16-d MR + re-ranking with the 2048-d MR outperforms DiskANN built on 2048-d RR at *half* the compute cost on ImageNet retrieval.

	RR-2048	AdANNS
PQ Budget (Bytes)	32	**16**
Top-1 Accuracy (%)	70.37	**70.56**
mAP@10 (%)	62.46	**64.70**
Precision@40 (%)	65.65	**68.25**

These results show the generality of AdANNS and its broad applicability across a variety of ANNS indices built on top of the base building blocks. Currently, AdANNS piggybacks on typical ANNS pipelines for their inherent accounting of the real-world system constraints [89, 125, 129]. However, we believe that AdANNS's flexibility and significantly better accuracy-compute trade-off can be further informed by real-world deployment constraints. We leave this high-potential line of work that requires extensive study to future research.

3.6 Further Analysis and Discussion

3.6.1 Compute-aware Elastic Search During Inference

AdANNS search structures cater to many specific large-scale use scenarios that need to satisfy precise resource constraints during construction as well as inference. However, in many cases, construction and storage of the indices are not the bottlenecks or the user is unable to search the design space. In these settings, AdANNS-D enables adaptive inference through accurate yet cheaper distance computation using the low-dimensional prefix of matryoshka representation. Akin to composite indices (Section 3.5.3) that use PQ vectors for cheaper distance computation, we can use the low-dimensional MR for faster distance computation on ANNS structure built *non-adaptively* with a high-dimensional MR without any modifications to the existing index.

Empirical results. Figure 3.2 shows that for a given compute budget using IVF on ImageNet-1K retrieval, AdANNS-IVF is better than AdANNS-IVF-D due to the explicit control during the building of the ANNS structure which is expected. However, the interesting observation is that AdANNS-D *matches or outperforms* the IVF indices built with MRs of varying capacities for ImageNet retrieval.

However, these methods are applicable in specific scenarios of deployment. Obtaining optimal AdANNS search structure (highly accurate) or even the best IVF-MR index relies on a relatively expensive design search but delivers indices that fit the storage, memory, compute, and accuracy constraints all at once. On the other hand AdANNS-D does not require a precisely built ANNS index but can enable compute-aware search during inference. AdANNS-D is a great choice for setups that can afford only one single database/index but need to cater to varying deployment constraints, e.g., one task requires 70% accuracy while another task has a compute budget of 1 MFLOPS/query.

3.6.2 Why MRs over RRs?

Quite a few of the gains from AdANNS are owing to the quality and capabilities of matryoshka representations. So, we conducted extensive analysis to understand why matryoshka representations seem to be more aligned for semantic search than the status-quo rigid representations.

Difficulty of NN search. Relative contrast (C_r) [97] is inversely proportional to the difficulty of nearest

neighbor search on a given database. On ImageNet-1K, Figure 14 of Rege et al. [209] shows that MRs have better C_r than RRs across dimensionalities, further supporting that matryoshka representations are more aligned (easier) for NN search than existing rigid representations for the same accuracy. More details and analysis about this experiment can be found in Appendix H.2 of Rege et al. [209] .

Clustering distributions. We also investigate the potential deviation in clustering distributions for MRs across dimensionalities compared to RRs. Unlike the RRs where the information is uniformly diffused across dimensions [232], MRs have hierarchical information packing. Figure 11 in Appendix E.3 of Rege et al. [209] shows that matryoshka representations result in clusters similar (measured by total variation distance [158]) to that of rigid representations and do not result in any unusual artifacts.

Robustness. Figure 9 in Appendix E of Rege et al. [209] shows that MRs continue to be better than RRs even for out-of-distribution (OOD) image queries (ImageNetV2 [207]) using ANNS. It also shows that the highest data dimensionality need not always be the most robust which is further supported by the higher recall using lower dimensions. Further details about this experiment can be found in Appendix E.1 of Rege et al. [209].

Generality across encoders. IVF-MR consistently has higher accuracy than IVF-RR across dimensionalities despite having similar accuracies with exact NN search (for ResNet50 on ImageNet and BERT-Base on NQ). We find that our observations on better alignment of MRs for NN search hold across neural network architectures, ResNet18/34/101 [98] and ConvNeXt-Tiny [171]. Appendix H.3 of Rege et al. [209] delves deep into the experimentation done using various neural architectures on ImageNet-1K.

Recall score analysis. Analysis of recall score (see Appendix C of Rege et al. [209]) in Appendix H.1 of Rege et al. [209] shows that for a similar top-1 accuracy, lower-dimensional representations have better 1-Recall@1 across search complexities for IVF and HNSW on ImageNet-1K. Across the board, MRs have higher recall scores and top-1 accuracy pointing to easier "searchability" and thus suitability of matryoshka representations for ANNS. Larger-scale experiments and further analysis can be found in Appendix H of Rege et al. [209].

Through these analyses, we argue that matryoshka representations are better suited for semantic search than rigid representations, thus making them an ideal choice for AdANNS.

3.6.3 Search for AdANNS Hyperparameters

Choosing the optimal hyperparameters for AdANNS, such as d_c, d_s, m, # clusters, # probes, is an interesting and open problem that requires more rigorous examination. As the ANNS index is formed *once* and used for potentially billions of queries with massive implications for cost, latency and queries-per-second, a hyperparameter search for the best index is generally an acceptable industry practice [125, 178]. The Faiss library [128] provides guidelines[1] to choose the appropriate index for a specific problem, including memory constraints, database size, and the need for exact results. There have been efforts at automating the search for optimal indexing parameters, such as Autofaiss[2], which maximizes recall given compute constraints.

In case of AdANNS, we suggest starting at the best configurations of MRs followed by a local design space search to lead to near-optimal AdANNS configurations (e.g. use IVF-MR to bootstrap AdANNS-IVF). We also share some observations during the course of our experiments:

1. AdANNS-IVF: Top-1 accuracy generally improves (with diminishing returns after a point) with increasing dimensionality of clustering (d_c) and search (d_s), as we show on ImageNet variants and with multiple encoders in the Appendix (Figures 9 and 15) of Rege et al. [209]. Clustering with low-d MRs matches the performance of high-d MRs as they likely contain similar amounts of useful information, making the increased compute cost not worth the marginal gains. Increasing # probes naturally boosts performance (Appendix, Figure 10a, of Rege et al. [209]). Lastly, it is generally accepted that a good starting point for the # clusters k is $\sqrt{N_D/2}$, where N_D is the number of indexable items [180]. $k = \sqrt{N_D}$ is the optimal choice of k from a FLOPS computation perspective as can be seen in Appendix B.1 of Rege et al. [209].

2. AdANNS-OPQ: we observe that for a fixed compute budget in bytes (m), the top-1 accuracy reaches a peak at $d < d_{max}$ (Appendix, Table 4, of Rege et al. [209]). We hypothesize that the better performance of AdANNS-OPQ at $d < d_{max}$ is due to the curse of dimensionality, i.e. it is easier to learn PQ codebooks on smaller embeddings with similar amounts of information. We find that using an MR with $d = 4 \times m$ is a good starting point on ImageNet and NQ. We also suggest using an 8-bit (256-length) codebook for OPQ as the default for each of the sub-block quantizer.

3. AdANNS-DiskANN: Our observations with DiskANN are consistent with other indexing structures, i.e.

[1] https://github.com/facebookresearch/faiss/wiki/Guidelines-to-choose-an-index
[2] https://github.com/criteo/autofaiss

the optimal graph construction dimensionality $d < d_{max}$ (Appendix, Figure 12, of Rege et al. [209]). A careful study of DiskANN on different datasets is required for more general guidelines to choose graph construction and OPQ dimensionality d.

3.6.4 Limitations

AdANNS's core focus is to improve the design of the existing ANNS pipelines. To use AdANNS on a corpus, we need to back-fill [204] the MRs of the data – a significant yet a one-time overhead. We also notice that high-dimensional MRs start to degrade in performance when optimizing also for an extremely low-dimensional granularity (e.g., < 24-d for NQ) – otherwise is it quite easy to have comparable accuracies with both RRs and MRs. Lastly, the existing dense representations can only in theory be converted to MRs with an auto-encoder-style non-linear transformation. We believe most of these limitations form excellent future work to improve AdANNS further.

3.7 Conclusions

We proposed a novel framework, AdANNS, that leverages adaptive representations for different phases of ANNS pipelines to improve the accuracy-compute tradeoff. AdANNS utilizes the inherent flexibility of matryoshka representations [147] to design better ANNS building blocks than the standard ones which use the rigid representation in each phase. AdANNS achieves SOTA accuracy-compute trade-off for the two main ANNS building blocks: search data structures (AdANNS-IVF) and quantization (AdANNS-OPQ). The combination of AdANNS-based building blocks leads to the construction of better real-world composite ANNS indices – with as much as $8\times$ reduction in cost at the same accuracy as strong baselines – while also enabling compute-aware elastic search. Finally, we note that combining AdANNS with elastic encoders [62] enables truly adaptive large-scale retrieval.

Chapter 4

MatFormer: Nested Transformer for Elastic Inference

4.1 Overview

Transformer models are deployed in a wide range of settings, from multi-accelerator clusters to standalone mobile phones. The diverse inference constraints in these scenarios necessitate practitioners to train foundation models such as PaLM 2, Llama, & ViTs as a series of models of varying sizes. Due to significant training costs, only a select few model sizes are trained and supported, limiting more fine-grained control over relevant tradeoffs, including latency, cost, and accuracy. This chapter introduces MatFormer[1], a nested Transformer architecture designed to offer elasticity in a variety of deployment constraints. Each Feed Forward Network (FFN) block of a MatFormer model is jointly optimized with a few nested smaller FFN blocks. This training procedure allows for the Mix'n'Match of model granularities across layers – i.e., a trained universal MatFormer model enables extraction of *hundreds* of accurate smaller models, which were never explicitly optimized. We empirically demonstrate MatFormer's effectiveness across different model classes (decoders & encoders), modalities (language & vision), and scales (up to 2.6B parameters). We find that a 2.6B decoder-only MatFormer language model (MatLM) allows us to extract smaller models spanning from 1.5B to 2.6B, each exhibiting comparable validation loss and one-shot downstream evaluations

[1] MatFormer stands for **Mat**ryoshka Trans**former** due to the model's inherent nested nature.

to their independently trained counterparts. Furthermore, we observe that smaller encoders extracted from a universal MatFormer-based ViT (MatViT) encoder preserve the metric-space structure for adaptive large-scale retrieval. Finally, we showcase that speculative decoding with the accurate and *consistent* submodels extracted from MatFormer can further reduce inference latency. Code and pretrained models are open-sourced at https://github.com/RAIVNLab/MatFormer-OLMo and https://github.com/google-research/scenic/tree/main/scenic/projects/matvit.

4.2 Introduction

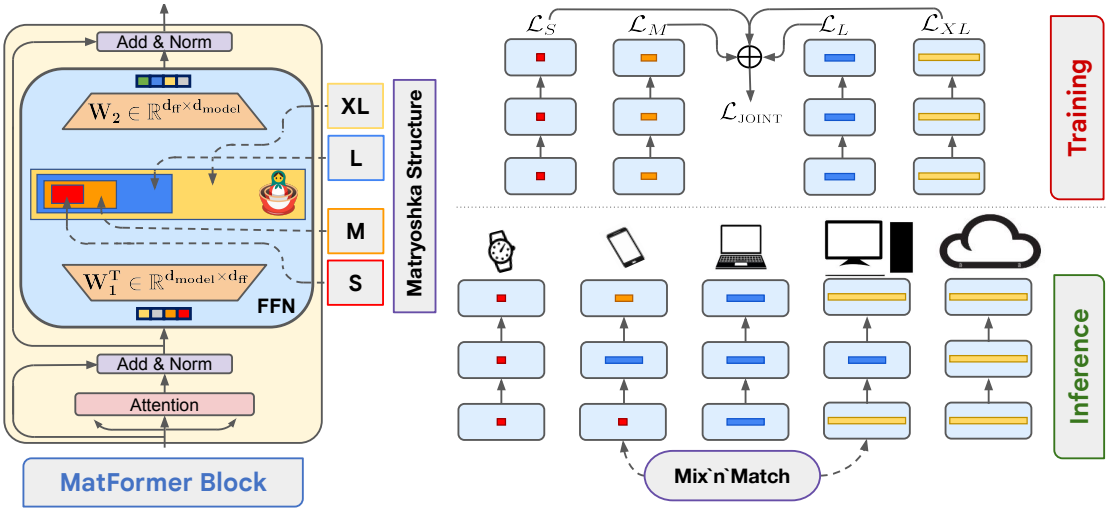


Figure 4.1: MatFormer introduces nested structure into the Transformer's FFN block & jointly trains all the submodels, enabling free extraction of hundreds of accurate submodels for elastic inference.

Large Foundation models [9, 195, 53] are deployed in a variety of settings like real-time response on mobile phones or in batch setting on multi-cluster GPUs for web-scale serving. To handle such varied settings, each model family provides a few *independently trained* models of different sizes. In order to cover a wide range of applications, typically these models' sizes are nearly linear on log-scale. For example, Llama family provides models with 7B, 13B, 33B and 65B parameters [239].

Such an approach has two key drawbacks: (a) as the models are independently trained, they incur significant overhead for colocation during inference and are not behaviorally consistent with each other which are detrimental to inference optimization techniques like speculative decoding [157] and model

cascades [258], and (b) due to training overhead, practitioners typically train only a few models which do not cover the entire set of downstream use-cases. For example, a deployment setup might, say, have the latency budget to support 40B parameter Llama model, but can only host a 33B variant because the next bigger model (65B) has significantly higher latency. So, one would need to settle for a less accurate model despite the larger latency budget. While model compression approaches aim to address this issue, they typically require additional training for each model that needs to be extracted. Furthermore, when applied to LLMs, these techniques are known to significantly drop the accuracy [124].

In this chapter, we propose MatFormer, a natively elastic Transformer [245] architecture that allows for training one *universal* model which can be used to extract hundreds of smaller submodels without *any additional training* (Figure 4.1). MatFormer is a general architecture that can be applied to both encoders and decoders, is domain agnostic, and is compatible with most design choices and training pipelines of large Transformer-based models – LLMs & ViTs.

MatFormer follows the principle of matryoshka representation learning [147], discussed in Chapter 2, to introduce nested substructure inside the standard Transformer block. Formally, MatFormer defines a Transformer blocks T_i, such that, $T_1 \subset T_2 \subset \cdots \subset T_g$, where g is the number of nested transformer blocks, and $T_i \subset T_{i+1}$ relation indicates that the parameters of T_i are contained in those of T_{i+1}. MatFormer can induce such sub-structure in both the attention and the feedforward network (FFN) blocks of the Transformer (see Figure 4.1). Consider a FFN block that has d_{ff} neurons in the hidden layer. Then, MatFormer induces matryoshka structure on these neurons, where T_i contains the first m_i neurons and $1 \leq m_1 \leq m_2 \cdots \leq m_g = d_{\text{ff}}$ represent the number of neurons for each granularity or sub-model. Intuitively, this implies that the first m_1 neurons are "most significant" neurons as they belong to all the blocks followed by the next $m_2 - m_1$, and so on. We can form a similar sub-structure on the attention heads, with the heads being organized from "most" to "least" significant, where the more significant heads are shared by more sub-models. That is, we use only the first m_i attention heads for the ith granularity. In fact, we can also introduce this sub-structure in the token embedding (d_{model}) supplied to each Transformer block.

However, in most LLMs and ViTs, the FFN block in the Transformer accounts for more than 60% non-embedding parameters and is responsible for the largest chunk of latency during inference. So, in this work, we focus on inducing the MatFormer's nested sub-structure in the FFN block. We then stack the

individual blocks (for l layers) to form g nested models ($\mathcal{M}_{1...g}$) with shared parameters i.e., $\mathcal{M}_i \subset \mathcal{M}_{i+1}$. Finally, we jointly train these g models by combining each model's loss.

This leads to a natural question: can one extract more than g models after inducing the MatFormer structure? Yes, in fact, it is possible to extract exponentially many models. Using the trained MatFormer blocks $T_1, \ldots, T_g$ at each layer, one can form new models by Mix'n'Match, i.e., by taking an arbitrary combination of these blocks across layers. For example, in the first layer, one can select T_g, the largest block, choose T_2 in the second layer, and so on, forming g^l different models. As we explicitly optimized only for g models, instead of the exponentially many models, are the extracted models accurate? Surprisingly, in multiple settings, and for a various model sizes, we observe that the extracted models indeed are accurate, with accuracy scaling with the size of the extracted model.

We train Matformer-based decoder-only Language Models (MatLM) up to 2.6B parameters and observe that: (a) MatLMs explicitly trained with g exponentially spaced granularities almost match validation loss and one-shot downstream evals of respective g baseline models trained independently from scratch, (b) our extracted models using Mix'n'Match lie on the accuracy-vs-parameters trade-off curve generated by the g explicitly trained models, (c) through scaling experiments we observe that the loss vs compute law for different MatFormer models remains similar to vanilla Transformer models across different granularities and (d) the submodels extracted from MatLM have highly consistent behavior that is highly desirable for inference optimizations and deployment across scales.

We further studied MatFormer-based ViT models (MatViT) and have similar observations as MatLM. For example, MatViT-L/16 improves the accuracy of the standard ViT-L/16 model on ImageNet-1K, and the extracted sub-models all match or even perform better than the independently trained baselines. Furthermore, we demonstrate that, due to high consistency, MatViT models can be used as "elastic encoders" for adaptive image retrieval. That is, the metric-space of an image encoded by the universal (i.e. the largest) MatViT model is roughly preserved by the nested submodels. Hence, based on query complexity, system load, and various other considerations, we can use one of the extracted MatViT encoders at inference time for retrieval on a fixed corpus encoded by the universal model – providing over 40% lesser compute overhead with $< 0.5\%$ drop in accuracy.

We make these key contributions:

1. We introduce MatFormer, which incorporates a nested sub-structure within the standard Transformer and jointly optimizes all the g granularities to produce a single, universal elastic model.
2. Employing Mix'n'Match of granularities across layers in a universal MatFormer model yields hundreds of accurate and consistent submodels without any additional training cost (Section 4.4).
3. MatFormer generalizes effectively to both decoder-only language models (MatLM) and vision encoders (MatViT), scaling as reliably and accurately as the standard Transformer, while enabling significantly faster autoregressive generation and large-scale adaptive dense retrieval (Section 4.5).

4.3 Related Work

A standard Transformer [245] has become the unifying model architecture for foundation models [24] across modalities like language [27], vision [53] and audio [203]. While extremely powerful, the standard Transformer block is not natively elastic in a way that enables large-scale adaptive and flexible deployment across various resource constraints. To cater to the plethora of deployment requirements, existing solutions include training a family of models of varying sizes [9, 240], post-hoc efficiency techniques like quantization [60], pruning [149], distillation [220] and mixture of varying capacity experts (MoE) [278]. However, these solutions often are specific to the single constraint at hand, and require additional training or trade-off memory/compute during inference making them far from being a truly elastic solution for adaptive deployment. Lastly, Transformer based LLMs are often sped-up during inference with techniques like speculative decoding [157, 40] – that benefits from the smaller draft & the larger verifier models having similar behavior – or early exiting [222] to enable real-time deployment.

Obtaining multiple smaller models from a single model has been explored in the past [275, 274, 30, 87, 31] with most works focusing on CNN encoders. Specifically, OFA [30] creates a universal CNN model which is used to extract and finetune submodels for a handful of deployment constraints while slimmable networks [275] optimize for limited preset widths and require explicit training to interpolate for a few more intermediate widths [274]. NAS techniques that sample random (not nested) subnetworks during training at each step, and then find the subnetwork architecture to retrain from scratch before deployment have been explored [255]. These techniques fall short of being truly elastic and come with significant training overheads. More recently some of them have been extended to Transformer encoders [39, 110, 218] for extracting

sub-models in both static or dynamic settings but fail at extending further to decoder-only language models. While not in the weight space, matryoshka representation learning [147] & FlexiViT [21] showcase elasticity in output & input spaces respectively by smoothly spanning deployment constraints with minimal overhead. MatFormer, in contrast, builds upon these works by nested the weight space instead to enable truly elastic and adaptive Transformer-based (decoder & encoder) models that span all the accuracy-vs-compute tradeoff (statically or dynamically) with minimal changes and training overhead (Figure 4.1). Finally, we also point the readers to SortedNet [241], a concurrent work with similar goals applied to encoders, which optimizes many sampled submodels (akin to prior works) unlike MatFormer's joint optimization of a few (typically 4) nested submodels.

4.4 MatFormer

In this section, we define MatFormer's nested substructure (Section 4.4.1) and discuss its training procedure for a chosen g model granularities (Section 4.4.2). We then discuss elastic inference using Mix'n'Match models (Section 4.4.3) from MatFormer along with its deployment considerations.

4.4.1 MatFormer Structure

MatFormer defines g Transformer blocks T_i, such that, $T_1 \subset T_2 \subset \cdots \subset T_g$ where $T_i \subset T_{i+1}$ indicates that the parameters of T_i are contained in those of T_{i+1}. While it is possible to impose such a structure on any part of the Transformer, we select the FFN block to define our method and present our experiments, as the model size and computational cost of a Transformer is dominated (around 60% for LLMs and ViTs) by the FFN block (see Appendix B of Devvrit et al. [62]).

The Transformer FFN block has a single hidden layer with d_{ff} neurons and both input and outputs in $\mathbb{R}^{d_{\text{model}}}$, and fixed FFN ratio $:= d_{\text{ff}}/d_{\text{model}}$ (typically ≥ 4). MatFormer introduces the matryoshka nested structure with g granularities on the hidden representation of the FFN block. Concretely, a nested sub-block of the Transformer, T_i contains the first m_i neurons of the FFN and $1 \leq m_1 \leq \cdots \leq m_g = d_{\text{ff}}$ represent the number of neurons for each granularity or sub-model. So, depending on the chosen granularity the FFN

operation of T_i i.e., T_i^{FFN} on an input $x \in \mathbb{R}^{d_{\text{model}}}$ is:

$$T_i^{\text{FFN}}(x) = \sigma(x \cdot \mathbf{W}_1[0 : m_i]^\top) \cdot \mathbf{W}_2[0 : m_i], \tag{4.1}$$

where the weight matrices of FFN are $\mathbf{W}_1, \mathbf{W}_2 \in \mathbb{R}^{d_{\text{ff}} \times d_{\text{model}}}$ and bias terms are omitted for simplicity. $\mathbf{W}_1[0 : k]$ denotes the submatrix with the first k rows of $\mathbf{W}_1$. Finally, σ is a non-linearity often set to GELU [105] or squared ReLU [231]. In this work, we chose the $g = 4$ exponentially spaced granularities with FFN ratios of $\{0.5, 1, 2, 4\}$ i.e., the nested hidden neurons are of the sizes $\{\frac{d_{ff}}{8}, \frac{d_{ff}}{4}, \frac{d_{ff}}{2}, d_{ff}\}$.

With the nested MatFormer blocks $T_1, T_2 \ldots T_g$, we can combine these to form a MatFormer model, with g nested submodels $\mathcal{M}_1 \subset \mathcal{M}_2 \ldots, \subset \mathcal{M}_g$ where $\mathcal{M}_i \leftarrow [T_i]^{\times l}$, i.e., $\mathcal{M}_i$ is formed by stacking T_i for l layers. The input and output embedding matrices are shared across the models.

4.4.2 Training

For a Transformer model $\mathcal{M}$, the forward pass on an input x is denoted by $\mathcal{M}(x)$ and let $\mathcal{L}$ denote the loss function between the output and the target y: $\mathcal{L}(\mathcal{M}(x), y)$.

MatFormer relies on a simple training strategy of jointly optimizing all the g nested submodels together. To this end, we set the MatFormer loss as a weighted average of loss of g submodels and train for it using the standard stochastic gradient-based optimizers [225]:

$$\mathcal{L}_{\text{JOINT}}(x, y) = \sum_{i=1}^{g} \lambda_i \cdot \mathcal{L}(\mathcal{M}_i(x), y), \tag{4.2}$$

where $\lambda_i > 0$ is the weight of i-th granular submodel. In this work, we set $\{\lambda_i\}_{i=1\ldots g}$ to be uniform i.e., $1/g$ but explore tuning $\{\lambda_i\}_{i=1\ldots g}$ in Appendix D.4 of Devvrit et al. [62] to further improve MatFormer.

The joint training in MatFormer involves one forward pass per each of the g submodels and benefits from portions of shared computation during backpropagation. MatFormer training results in g accurate nested submodels $\mathcal{M}_{1\ldots g}$ inside the universal MatFormer model ($\mathcal{M}_g$). Note that this simple strategy outperforms various other training techniques (Appendix D.2 of Devvrit et al. [62]). Finally, instead of pretraining models with MatFomer structure, we can also induce this structure via finetuning.

MatFormer training is $\sim 15\%$ faster (for $g = 4$) than training all the Transformer based equivalent

submodels independently (Appendix B of Devvrit et al. [62]). However, MatFormer also enables the extraction of hundreds of smaller submodels along the accuracy-vs-compute curve traced by the g explicitly optimized submodels (Section 4.4.3). These models emerge for free using Mix'n'Match during inference and drastically reduce the amortized training cost per model obtained through MatFormer. The joint optimization, even without self-distillation from $\mathcal{M}_g$, results in smaller submodels that have highly consistent behavior (Section 4.4.4) with the universal model. Finally, in Appendix B.1 of Devvrit et al. [62], we argue that the training efficiency of MatFormer can be significantly improved through various optimizations.

4.4.3 Mix'n'Match

At inference time, it is trivial to extract one of the g submodels $\mathcal{M}_1 \subset \mathcal{M}_2 \ldots, \subset \mathcal{M}_g$ by stacking the corresponding Transformer block T_i across layers. However, by selecting different granularity for each MatFormer layer, it is possible to generate a combinatorially large number of accurate smaller models for free. We call this simple procedure *Mix'n'Match* and observe that these additional model granularities –which were never explicitly optimized – are highly performant.

In fact, we can further increase the number of extracted models by generating interpolating blocks between fixed granulaties [147]. For example, we can generate a $\widetilde{T}$ block that uses first $\frac{1}{2}(m_i + m_{i+1})$ neurons in the FFN layer which still tends to be highly accurate.

To summarize, given a computational budget, we can extract a highly accurate model with Mix'n'Match for the constraints rather than using a smaller less accurate model or training a model for this specific constraint (Sections 4.5.1 & 4.5.2). We note that a compute constraint can be satisfied by various Mix'n'Match models with different accuracies, making identifying the best Mix'n'Match configurations without downstream validation is an exciting direction for future work.

4.4.4 Deployment

During deployment, all we need to store is the single universal MatFormer model for different types of elastic inference depending on the constraints. In the case of static workloads, where compute resources are known beforehand and the inputs remain relatively similar in difficulty, one can choose the most accurate static submodel for the constraints using Mix'n'Match. This eliminates the usage of a less accurate preexisting

model or training of a new one for the specific constraints.

For dynamic workloads, where the compute resources or the input hardness change on the fly, we can use the universal MatFormer model to dynamically extract the optimal submodel for token-based routing in LLMs akin to MoE [138, 162] and elastic encoders in dense retrieval (Section 4.5.2). This works largely because all the extracted submodels have high behavioral *consistency* with universal MatFormer model (Section 4.5.1) – minimizing the drift across predictions from various submodels. We measure the consistency between two generative models as the *percentage of matching tokens* generated by them for the same prefix or using the *KL divergence* of the smaller model outputs with the larger model outputs – this accounts for potential sampling strategies in decoding. This highly consistent nature of MatFormer results in superior inference time speedups for techniques like speculative decoding [157] (Section 4.5.1) and can assist in reducing prediction drift between cross platform deployments. We also show that higher model consistency also aids metric-space structure preservation in encoder models (Section 4.5.2).

4.5 Experiments

In this section, we empirically evaluate MatFormer across modalities (language and vision), model classes (decoder and encoder), and scales (up to 2.6B parameters). Specifically, we train and analyze MatFormer-based decoder-only Language Models – MatLMs (Section 4.5.1) – and encoder-only Vision Transformers – MatViT (Section 4.5.2) models with $g = 4$ nested granularities across various model sizes. For a fair comparison, we also independently train the Transformer baseline for the submodel of each granularity across model sizes for the same tasks. We primarily focus on the elastic deployment of MatFormer-based models (Sections 4.5.1 & 4.5.2) for tasks spanning from one-shot generative evals to adaptive image retrieval. Additionally, we also investigate the reliable scaling behavior [132] of the MatFormer models (Section 4.5.1).

4.5.1 MatLM: MatFormer Language Models

We build MatFormer-based decoder-only Language Models – MatLMs – and contrast them to their vanilla Transformer counterparts (LMs) [169]. The LMs broadly follow the training pipeline and procedure outlined by Thoppilan et al. [238]. For each MatLM model with a set d_{model}, we jointly optimize for $g = 4$ nested granularities represented by FFN ratios of $\{0.5, 1, 2, 4\}$ – i.e., only the hidden representation size of the FFN

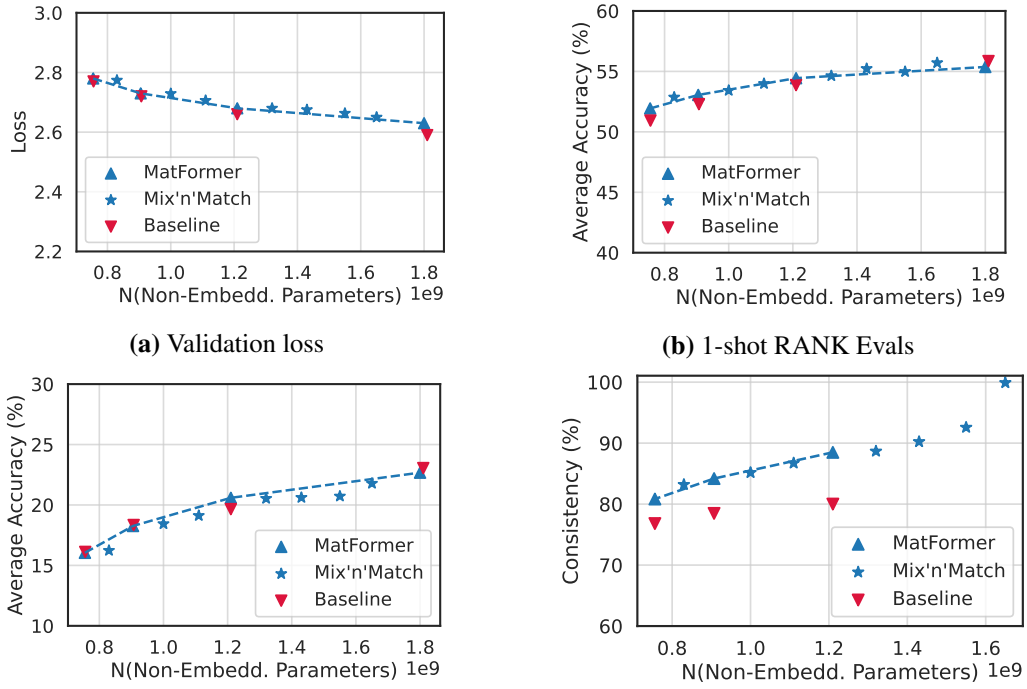
block changes. We denote these submodels as MatLM – {S, M, L, XL} in increasing order of model size and refer to MatLM-XL as the universal MatLM. For baselines, we train vanilla Transformer models with comparable architectures. That is, for each MatLM, we train 4 separate baseline models with FFN ratios of $\{0.5, 1, 2, 4\}$ for a fixed d_{model} denoted as Baseline – {S, M, L, XL}. We evaluate these models on validation loss (= log perplexity) and average accuracy on 26 English tasks similar to [27, 68, 9]. Of these 26 tasks, we group 5 tasks that require generating multiple tokens under "GEN" and the remaining tasks that involve choosing an option from the input text under "RANK". Please see Appendix A of Devvrit et al. [62] for further details on training, evaluation, and the datasets.

Elastic Inference with MatLM

To showcase elastic inference, we evaluate the 2.6B parameter MatLM models on its ability (a) to provide models spanning the accuracy-vs-compute curve using Mix'n'Match (Section 4.4.3) and (b) to improve post-hoc inference optimization techniques like Speculative Decoding [157] to further speed-up accurate auto-regressive generation.

Accurate MatLM submodels for every constraint for free with Mix'n'Match. Leveraging Mix'n'Match, a MatLM can provide accurate models for every compute constraint (between S and XL), not just the explicitly optimized granularities {S, M, L, XL}. We evaluate the impact of Mix'n'Match on the 2.6B parameter MatLM in Figure 4.2 through validation loss and downstream evals and contrast them to four granularities {S, M, L, XL} of the 2.6B baseline LM (all trained independently). In Figures 4.2a, 4.2b & 4.2c, we show that all MatLM – {S, M, L, XL} models all perform as well as their corresponding baselines – with marginal improvements and drops across the scale.

In Figure 4.2a we see that Mix'n'Match helps obtain many models on the optimal loss-vs-compute curve at zero cost. Moreover, downstream eval tasks on these Mix'n'Match models also mimic this trend, as shown in Figures 4.2c & 4.2b. In a deployment setting that only has 55% of the required compute resources needed for the MatLM-XL model, it is now possible to have a Mix'n'Match submodel with $< 2\%$ accuracy drop on RANK evals. Without elastic deployment due to Mix'n'Match, we would see a $> 2.5\%$ accuracy drop due to the use of the MatLM-M model. Note that we highlight only a few of the hundreds of accurate Mix'n'Match models along the curves. We discuss additional details and results on the Mix'n'Match procedure in Appendix



(a) Validation loss

(b) 1-shot RANK Evals

(c) 1-shot GEN Evals

(d) Consistency with the XL model

Figure 4.2: Validation loss & one-shot downstream evaluation scores for the 2.6B MatLM & baseline models. Mix'n'Match helps generate accurate and more consistent models from MatLM that lie on the performance-vs-compute curve spanned by the explicitly optimized submodels.

of Devvrit et al. [62].

MatLM submodels speed up speculative decoding. Speculative decoding leverages an accurate lightweight LM as a draft model to autoregressively generate a few tokens, followed by verifying these drafts with a larger model through parallel decoding on the generated tokens. When the draft is inaccurate, the draft model is rolled back and reset to the larger model's output. This results in considerable inference speed-up for the *same accuracy as the large model*. We point the reader to the original paper for a more detailed explanation [157].

Slow down of this algorithm stems from cases where the smaller model's predictions disagree with the larger model. A draft model that is significantly more consistent with the larger verifier model would lead to less rollbacks of the draft predictions and therefore lower latency. As seen in Figure 4.2d the MatLM submodels can be up to 8.5% more consistent than the baselines to their corresponding XL model. The significant gap persists even in the KL divergence variant of consistency with the XL model's outputs (see Figure 6 in Appendix of Devvrit et al. [62]). This improved consistency along with the need for only a single

universal model positions MatLM favorably to improve techniques that require draft and verifier models such as speculative decoding.

Table 4.1 shows the inference time speed-ups from speculative decoding using the S and XL sub-models of the 2.6B language model for drafting and verification respectively. Speculative decoding with independently trained baseline LMs results in a speed-up of up to 10% over the standard autoregressive decoding of the 2.6B-XL model. But MatLM-based speculative decoding is up to 6% faster than traditional speculative decoding. This additional speed-up can be primarily attributed to the more consistent nature of MatLM-based drafter and verifier models and is further boosted by the ability to share attention cache across models from MatLM which is infeasible for the baselines (see Appendix B.2 of Devvrit et al. [62]). Finally, MatLM further reduces the memory overhead for inference by removing the need to have two models during resource-constrained deployment.

Table 4.1: Inference time speed-ups over a standard 2.6B model through speculative decoding using a 1.5B (S) draft and 2.6B (XL) verifier model.

Speculative Decoding	LAMBADA	TriviaQA
Baseline	1.10×	1.08×
MatLM	1.14×	1.11×
+ shared attention cache	1.16×	1.14×

MatLM Scales as well as Vanilla Transformer LMs

Now that we have established that a 2.6B MatLM model and its submodels are as accurate as the baseline Transformer LMs, we want to examine the scalability of training MatLM models. So, we study the scaling properties [132, 108] of MatLMs and compare them to vanilla Transformer baseline LMs trained for the same number of tokens. We train models ranging from 78M to 2.6B parameters on 10B to 160B tokens and plot the validation loss for MatLM – {S, M, L, XL} compared against their baselines in Figure 7 in Appendix of Devvrit et al. [62].

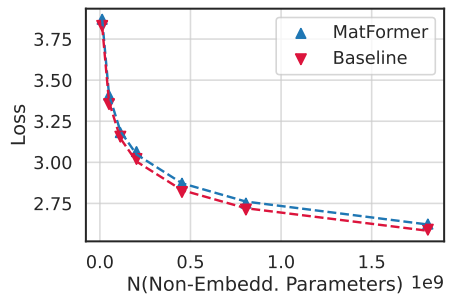
First, in Figure 4.3a, we observe that the training of MatLM-XL models across model sizes scale as reliably as the Baseline-XL LMs for loss vs. number of parameters. However, Figure 4.3b interestingly shows that it is not just the XL models but rather all the nested submodels, irrespective of granularity {S, M, L, XL}, of MatLM and Baseline that follow the same scaling trend. Therefore, we fit

a scaling law according to the number of non-embedding parameters (N) and training tokens (D) for all possible submodels for both MatLMs and the baselines in Table 4.2. We observe that the fitted parameters are extremely similar, suggesting that MatLMs scale similarly to vanilla Transformer LMs. In Figures 4.3c & 4.3d we also find that the downstream evals for MatLM are within 0.5% of the baselines, with the smaller submodels even outperforming the baselines at scale. Finally, Figure 7f in the Appendix of Devvrit et al. [62] shows that the MatLM submodels are more consistent with their XL model compared to the baseline counterparts across scales.

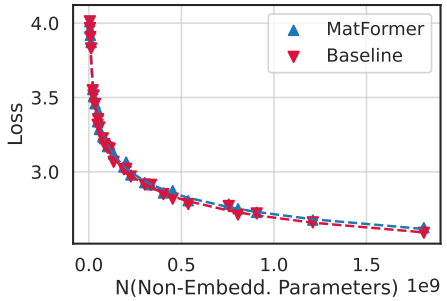
Table 4.2: Fitted parameters for the scaling equation: $\text{Loss}(N, D) = a \cdot (ND)^b + c$

	a	b	c
Baseline	20.917	-0.119	1.868
Matformer	17.516	-0.114	1.845

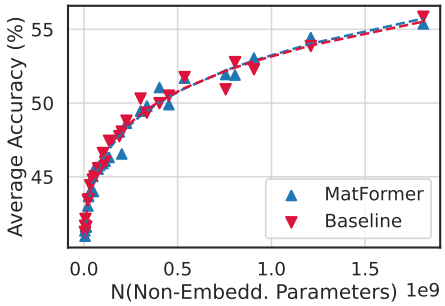
We note that the scaling law equation does not capture how (1) MatLMs have been optimized for multiple submodels and even have performant submodels that have not been explicitly optimized for (Section 4.5.1),



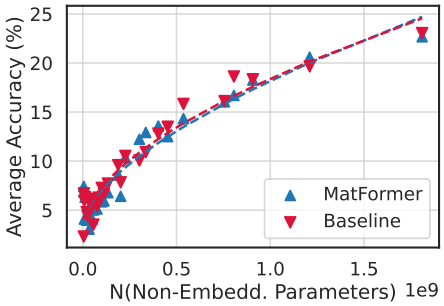
(a) Validation loss for XL-models



(b) Validation loss for all models



(c) 1-shot RANK Evals



(d) 1-shot GEN Evals

Figure 4.3: We train various decoder-only MatLM models at a range of sizes from 78M to 2.6B parameters and observe the scaling trends of all granularities (S, M, L, XL) for validation loss and 1-shot downstream evaluation scores. We find that the MatLM-XL models across scales mimic the training trends of Baseline-XL models. Interestingly, we also note that that validation loss and downstream evaluations follow the *scaling trends of the XL-models across all granularities*.

and (2) MatLMs and baselines of the same size have different training FLOPs per step. We leave formulations that capture these subtleties to future work and further discuss this in Appendix C.1 of Devvrit et al. [62]. We provide full results split by granularity in Appendix C of Devvrit et al. [62].

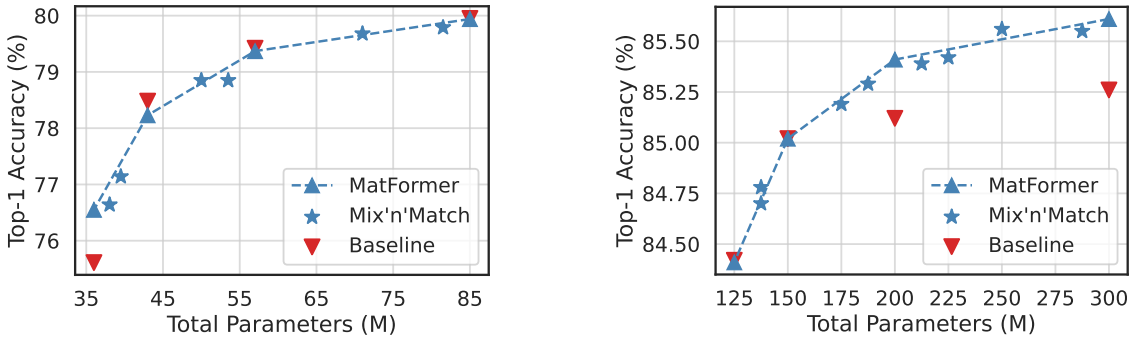
4.5.2 MatViT: MatFormer Vision Transformers

In this section, we extend MatFormer to Vision Transformer (ViT) [67] based computer vision encoder models. MatFormer-based ViT – MatViT – enables elastic inference for fundamental tasks like image classification and retrieval. To this end, we train the MatFormer variant of the standard ViT-B/16 and ViT-L/16 models – MatViT-B/16 and MatViT-L/16 that are trained with $g = 4$ prechosen nested granularities (FFN ratios of $\{0.5, 1, 2, 4\}$). B/16 models are trained on ImageNet-1K [215] with AugReg [233] while L/16 models are pretrained on ImageNet-21K [55] followed by finetuning on ImageNet-1K. All models are trained with the training setup and optimal hyperparameters of the standard ViT variants from the Scenic library [52].

Image Classification

For image classification, we evaluate both ViT & MatViT models on ImageNet-1K. Figure 4.4a shows that the explicitly optimized granularities in MatViT result in as accurate models as the independently trained baselines for the B/16. However for L/16, as shown in Figure 4.4b, we see that the MatViT models are up to 0.35% more accurate than the baseline for the same inference cost.

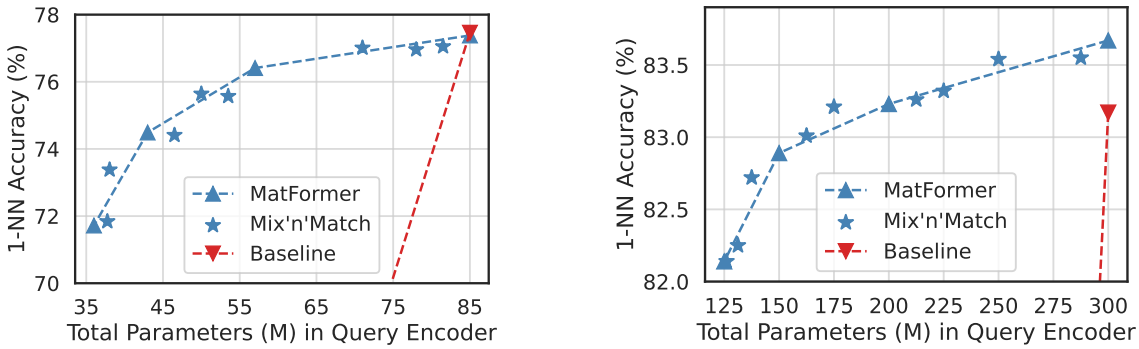
We then explore using MatFormer at different training stages with a 2×2 grid of pretraining-finetuning



(a) B/16 trained on ImageNet-1K with AugReg

(b) L/16 pretrained on IN-21K $\rightarrow$ ImageNet-1K.

Figure 4.4: MatViT variants match or outperform standard ViT models on ImageNet-1K classification and provide free extracted models that span the accuracy-compute curve through Mix'n'Match.



(a) B/16 trained on ImageNet-1K with AugReg

(b) L/16 pretrained on IN-21K $\rightarrow$ ImageNet-1K.

Figure 4.5: MatViT natively enables elastic encoders for adaptive retrieval that can be used for real-time query side computation while retaining strong accuracy on ImageNet-1K, unlike the baselines.

pairs (Table 7 in Appendix E.1 of Devvrit et al. [62]) and find that using a MatFormer during pretraining helps bring more accurate and flexible encoders for downstream use. Further, finetuning using MatFormer enhances elastic deployment depending on the constraints at hand through Mix'n'Match.

Adaptive Encoders with Mix'n'Match. Furthermore, our Mix'n'match models' accuracy almost lies on the line joining accuracy of explicitly trained granularities. In scenarios where, say, an application can host 50M parameter B/16 model, MatViT can provide 0.8% more accurate model than the current approach which would host the largest baseline model with $\leq$ 50M parameters.

During deployment, the universal MatViT model can be stored in memory and depending on the compute constraints be used to extract an adaptable smaller model to maximize accuracy with the available resources at that moment. Currently, we find the Mix'n'Match models on the accuracy-compute curve through a quick inference on the validation set. While relatively scalable, this points to the need for optimal budget allocation across layers in neural networks [145].

Adaptive Image Retrieval

The goal of image retrieval is to find semantically similar images – e.g. images from the same class – using representations obtained from a pretrained encoder [43]. Standard approach is to encode the database images as well as query image with same encoder and run nearest neighbor retrieval for the query embedding. While we can embed database images with an expensive encoder, the query encoder generally has to be real-time. Furthermore, the setting of query encoding might be varied, e.g., on-device vs. cloud processing, varying

query load and query complexity. Current solutions have to stick to a fixed encoder thus compromising on accuracy or cost for various settings.

Given the elastic nature of MatViT, it is a good candidate for query encoder. However, retrieval also requires that submodels preserve distances between fixed database (with large encoder) and query embeddings across all the granularities. If we use smaller baseline ViT models only for query encoding, these distances are not preserved and lead to nearly 0 retrieval accuracy (see Figure 4.5).

We evaluate both ViT and MatViT encoders on ImageNet-1K for image retrieval. We compute 1-nearest neighbor (NN) accuracy using the representation vector of the [CLS] token (also see Appendix E.2 of Devvrit et al. [62]). Figure 4.5 shows that submodels extracted from MatViT can approximately preserve distances and provide significantly more flexibility. For example, with a loss of $< 0.5\%$ accuracy, MatViT-L/16 can reduce compute cost by 40%. To our knowledge, this is the first result of its kind and opens up a wide variety of adaptive inference strategies for large-scale semantic search.

4.6 Conclusions

In this work we presented MatFormer, a natively elastic Transformer architecture that allows training a single universal model which can be used to extract hundreds of smaller accurate submodels at zero additional cost at deployment time We find that the MatFormer Language Model (MatLM) matches the perplexity & 1-shot accuracy of independently trained models. In fact, MatLM demonstrates an interesting loss-vs-compute scaling curve that is nearly *independent* of trained granularity indicating robust generalization to *extremely* large models as well. Finally, MatFormer submodels enable diverse inference time speedups like faster autoregressive generation with speculative decoding and elastic query encoders for adaptive dense retrieval across modalities.

Chapter 5

Soft Threshold Weight Reparameterization for Learnable Sparsity

5.1 Overview

Sparsity in Deep Neural Networks (DNNs) is studied extensively with the focus of maximizing prediction accuracy given an overall parameter budget. Existing methods rely on uniform or heuristic non-uniform sparsity budgets which have sub-optimal layer-wise parameter allocation resulting in a) lower prediction accuracy or b) higher inference cost (FLOPs). This chapter proposes Soft Threshold Reparameterization (STR), a novel use of the soft-threshold operator on DNN weights. STR smoothly induces sparsity while *learning* pruning thresholds thereby obtaining a non-uniform sparsity budget. Our method achieves state-of-the-art accuracy for unstructured sparsity in CNNs (ResNet50 and MobileNetV1 on ImageNet-1K), and, additionally, learns non-uniform budgets that empirically reduce the FLOPs by up to 50%. Notably, STR boosts the accuracy over existing results by up to 10% in the ultra sparse (99%) regime and can also be used to induce low-rank (structured sparsity) in RNNs. In short, STR is a simple mechanism which learns effective sparsity budgets that contrast with popular heuristics. Code, pretrained models and sparsity budgets are at https://github.com/RAIVNLab/STR.

5.2 Introduction

Deep Neural Networks (DNNs) are the state-of-the-art models for many important tasks in the domains of Computer Vision, Natural Language Processing, etc. To enable highly accurate solutions, DNNs require large model sizes resulting in huge inference costs, which many times become the main bottleneck in the real-world deployment of the solutions. During inference, a typical DNN model stresses the following aspects of the compute environment: 1) RAM - working memory, 2) Processor compute - Floating Point Operations (FLOPs[1]), and 3) Flash - model size. Various techniques are proposed to make DNNs efficient including model pruning (sparsity) [93], knowledge distillation [28], model architectures [111] and quantization [206].

Sparsity of the model, in particular, has potential for impact across a variety of inference settings as it reduces the model size and inference cost (FLOPs) without significant change in training pipelines. Naturally, several interesting projects address inference speed-ups via sparsity on existing frameworks [166, 69] and commodity hardware [10]. On-premise or Edge computing is another domain where sparse DNNs have potential for deep impact as it is governed by billions of battery limited devices with single-core CPUs. These devices, including mobile phones [8] and IoT sensors [197, 213], can benefit significantly from sparsity as it can enable real-time on-device solutions.

Sparsity in DNNs, surveyed extensively in Section 5.3, has been the subject of several papers where new algorithms are designed to obtain models with a given parameter budget. But state-of-the-art DNN models tend to have a large number of layers with highly non-uniform distribution both in terms of the number of parameters as well as FLOPs required per layer. Most existing methods rely either on uniform sparsity across all parameter tensors (layers) or on heuristic non-uniform sparsity budgets leading to a sub-optimal weight allocation across layers and can lead to a significant loss in accuracy. Furthermore, if the budget is set at a global level, some of the layers with a small number of parameters would be fully dense as their contribution to the budget is insignificant. However, those layers can have significant FLOPs, e.g., in an initial convolution layer, a simple tiny 3×3 kernel would be applied to the entire image. Hence, while such models might decrease the number of non-zeroes significantly, their FLOPs could still be large.

Motivated by the above-mentioned challenges, this works addresses the following question: "*Can we design a method to learn non-uniform sparsity budget across layers that is optimized per-layer, is stable, and*

[1] One Multiply-Add is counted as one FLOP

is accurate?".

Most existing methods for learning sparse DNNs have their roots in the long celebrated literature of high-dimension statistics and, in particular, sparse regression. These methods are mostly based on well-known Hard and Soft Thresholding techniques, which are essentially projected gradient methods with explicit projection onto the set of sparse parameters. However, these methods require a priori knowledge of sparsity, and as mentioned above, mostly heuristic methods are used to set the sparsity levels per layer.

We propose Soft Threshold Reparameterization (STR) to address the aforementioned issues. We use the fact that the projection onto the sparse sets is available in closed form and propose a novel reparameterization of the problem. That is, for forward pass of DNN, we use soft-thresholded version [66] of a weight tensor $\mathbf{W}_l$ of the l-th layer in the DNN: $\mathcal{S}(\mathbf{W}_l, \alpha_l) := \mathrm{sign}\,(\mathbf{W}_l) \cdot \mathrm{ReLU}(|\mathbf{W}_l| - \alpha_l)$ where α_l is the pruning threshold for the l-th layer. As the DNN loss can be written as a continuous function of α_l's, we can use backpropagation to learn layer-specific α_l to smoothly induce sparsity. Typically, each layer in a neural network is distinct unlike the interchangeable weights and neurons making it interesting to learn layer-wise sparsity.

Due to layer-specific thresholds and sparsity, STR is able to achieve state-of-the-art accuracy for unstructured sparsity in CNNs across various sparsity regimes. STR makes even small-parameter layers sparse resulting in models with significantly lower inference FLOPs than the baselines. For example, STR for 90% sparse MobileNetV1 on ImageNet-1K results in a 0.3% boost in accuracy with 50% fewer FLOPs. Empirically, STR's learnt non-uniform budget makes it a very effective choice for ultra (99%) sparse ResNet50 as well where it is $\sim$10% more accurate than baselines on ImageNet-1K. STR can also be trivially modified to induce structured sparsity, demonstrating its generalizability to a variety of DNN architectures across domains. Finally, STR's learnt non-uniform sparsity budget transfers across tasks thus discovering an efficient sparse backbone of the model.

The 3 major contributions of this work are:

- Soft Threshold Reparameterization (STR), for the weights in DNNs, to induce sparsity via learning the per-layer pruning thresholds thereby obtaining a better non-uniform sparsity budget across layers.
- Extensive experimentation showing that STR achieves the state-of-the-art accuracy for sparse CNNs (ResNet50 and MobileNetV1 on ImageNet-1K) along with a significant reduction in inference FLOPs.

- Extension of STR to structured sparsity, that is useful for the direct implementation of fast inference in practice.

5.3 Related Work

This section covers the spectrum of work on sparsity in DNNs. The sparsity in the discussion can be characterized as (a) unstructured and (b) structured while sparsification techniques can be (i) dense-to-sparse, and (ii) sparse-to-sparse. Finally, the sparsity budget in DNNs can either be (a) uniform, or (b) non-uniform across layers. This will be a key focus of this paper, as different budgets result in different inference compute costs as measured by FLOPs. This section also discusses the recent work on learnable sparsity.

5.3.1 Unstructured and Structured Sparsity

Unstructured sparsity does not take the structure of the model (e.g. channels, rank, etc.,) into account. Typically, unstructured sparsity is induced in DNNs by making the parameter tensors sparse directly based on heuristics (e.g. weight magnitude) thereby creating sparse tensors that might not be capable of leveraging the speed-ups provided by commodity hardware during training and inference. Unstructured sparsity has been extensively studied and includes methods which use gradient, momentum, and Hessian based heuristics [73, 155, 152, 95, 59], and magnitude-based pruning [93, 90, 282, 77, 79, 186, 15, 183, 189, 144, 266]. Unstructured sparsity can also be induced by L_0, L_1 regularization [173], and Variational Dropout (VD) [184].

Gradual Magnitude Pruning (GMP), proposed in [282], and studied further in [79], is a simple magnitude-based weight pruning applied gradually over the course of the training. Discovering Neural Wirings (DNW) [266] also relies on magnitude-based pruning while utilizing a straight-through estimator for the backward pass. GMP and DNW are the state-of-the-art for unstructured pruning in DNNs (especially in CNNs) demonstrating the effectiveness of magnitude pruning. VD gets accuracy comparable to GMP [79] for CNNs but at a cost of $2\times$ memory and $4\times$ compute during training making it hard to be used ubiquitously.

Structured sparsity takes structure into account making the models scalable on commodity hardware with the standard computation techniques/architectures. Structured sparsity includes methods which make parameter tensors low-rank [119, 6, 174], prune out channels, filters and induce block/group sparsity [170,

262, 161, 175, 85, 273]. Even though structured sparsity can leverage speed-ups provided by parallelization, the highest levels of model pruning are only possible with unstructured sparsity techniques.

5.3.2 Dense-to-sparse and Sparse-to-sparse Training

Until recently, most sparsification methods were dense-to-sparse i.e., the DNN starts fully dense and is made sparse by the end of the training. Dense-to-sparse training in DNNs encompasses the techniques presented in [93, 282, 184, 77, 210].

The lottery ticket hypothesis [77] sparked an interest in training sparse neural networks end-to-end. This is referred to as sparse-to-sparse training and a lot of recent work [186, 15, 73, 155, 59] aims to do sparse-to-sparse training using techniques which include re-allocation of weights to improve accuracy.

Dynamic Sparse Reparameterization (DSR) [186] heuristically obtains a global magnitude threshold along with the re-allocation of the weights based on the non-zero weights present at every step. Sparse Networks From Scratch (SNFS) [59] utilizes momentum of the weights to re-allocate weights across layers and the Rigged Lottery (RigL) [73] uses the magnitude to drop and the periodic dense gradients to regrow weights. SNFS and RigL are state-of-the-art in sparse-to-sparse training but fall short of GMP for the same experimental settings. It should be noted that, even though sparse-to-sparse can reduce the training cost, the existing frameworks [196, 1] consider the models as dense resulting in minimal gains.

DNW [266] and Dynamic Pruning with Feedback (DPF) [164] fall between both as DNW uses a fully dense gradient in the backward pass and DPF maintains a copy of the dense model in parallel to optimize the sparse model through feedback. Note that DPF is complementary to most of the techniques discussed here.

5.3.3 Uniform and Non-uniform Sparsity

Uniform sparsity implies that all the layers in the DNN have the same amount of sparsity in proportion. Quite a few works have used uniform sparsity [79], given its ease and lack of hyperparameters. However, some works keep parts of the model dense, including the first or the last layers [164, 186, 282]. In general, making the first or the last layers dense benefits all the methods. GMP typically uses uniform sparsity and achieves state-of-the-art results.

Non-uniform sparsity permits different layers to have different sparsity budgets. Weight re-allocation

heuristics have been used for non-uniform sparsity in DSR and SNFS. It can be a fixed budget like the ERK (Erdos-Renyi-Kernel) heuristic described in RigL [73]. A global pruning threshold [93] can also induce non-uniform sparsity and has been leveraged in Iterative Magnitude Pruning (IMP) [77, 210]. A good non-uniform sparsity budget can help in maintaining accuracy while also reducing the FLOPs due to a better parameter distribution. The aforementioned methods with non-uniform sparsity do not reduce the FLOPs compared to uniform sparsity in practice. Very few techniques like AMC [102], using expensive reinforcement learning, minimize FLOPs with non-uniform sparsity.

Most of the discussed techniques rely on intelligent heuristics to obtain non-uniform sparsity. Learning the pruning thresholds and in-turn learning the non-uniform sparsity budget is the main contribution of this work.

5.3.4 Learnable Sparsity

Concurrent to our work, [221, 168, 156, 269, 12] have proposed learnable sparsity methods through training of the sparse masks and weights simultaneously with minimal heuristics. The reader is urged to review these works for a more complete picture of the field. Note that, while STR is proposed to induce layer-wise unstructured sparsity, it can be easily adapted for global, filter-wise, or per-weight sparsity as discussed in Appendix A.5 of Kusupati et al. [145].

5.4 Method - STR

Optimization under sparsity constraint on the parameter set is a well studied area spanning more than three decades [66, 33, 123], and is modeled as:

$$\min_{\mathcal{W}} \mathcal{L}(\mathcal{W}; \mathcal{D}), \text{ s.t. } \|\mathcal{W}\|_0 \leq k,$$

where $\mathcal{D} \coloneqq \left\{\mathbf{x}_i \in \mathbb{R}^d, y_i \in \mathbb{R}\right\}_{i \in [n]}$ is the observed data, $\mathcal{L}$ is the loss function, $\mathcal{W}$ are the parameters to be learned and $\|\cdot\|_0$ denotes the L_0-norm or the number of non-zeros, and k is the parameter budget. Due to non-convexity and combinatorial structure of the L_0 norm constraint, it's convex relaxation L_1 norm has been studied for long time and has been at the center of a large literature on high-dimensional learning.

In particular, several methods have been proposed to solve the two problems including projected gradient descent, forward/backward pruning etc.

Projected Gradient Descent (PGD) in particular has been popular for both the problems as the projection onto both L_0 as well as the L_1 ball is computable in almost closed form [14, 123]; L_0 ball projection is called Hard Thresholding while L_1 ball projection is known as Soft Thresholding. Further, these methods have been the guiding principle for many modern DNN model pruning (sparsity) techniques [93, 282, 189].

However, projection-based methods suffer from the problem of dense gradient and intermediate parameter structure, as the gradient descent iterate can be arbitrarily out of the set and is then projected back onto L_0 or L_1 ball. At a scale of billions of parameters, computing such dense gradients and updates can be daunting. More critically, the budget parameter k is set at the global level, so it is not clear how to partition the budget for each layer, as the importance of each layer can be significantly different.

In this work, we propose a reparameterization, Soft Threshold Reparameterization (STR) based on the soft threshold operator [66], to alleviate both the above mentioned concerns. That is, instead of first updating $\mathcal{W}$ via gradient descent and then computing its projection, we directly optimize over projected $\mathcal{W}$. Let $\mathcal{S}_g(\mathcal{W}; s)$ be the projection of $\mathcal{W}$ parameterized by s and function g. $\mathcal{S}$ is applied to each element of $\mathcal{W}$ and is defined as:

$$\mathcal{S}_g(w, s) := \text{sign}\,(w) \cdot \text{ReLU}(|w| - g(s)), \tag{5.1}$$

where s is a learnable parameter, $g : \mathbb{R} \to \mathbb{R}$, and $\alpha = g(s)$ is the pruning threshold. $\text{ReLU}(a) = \max(a, 0)$. That is, if $|w| \leq g(s)$, then $\mathcal{S}_g(w, s)$ sets it to 0.

Reparameterizing the optimization problem with $\mathcal{S}$ *modifies* (note that it is not equivalent) it to:

$$\min_{\mathcal{W}} \mathcal{L}(\mathcal{S}_g(\mathcal{W}, \boldsymbol{s}), \mathcal{D}). \tag{5.2}$$

For L-layer DNN architectures, we divide $\mathcal{W}$ into: $\mathcal{W} = [\mathbf{W}_l]_{l=1}^{L}$ where $\mathbf{W}_l$ is the parameter tensor for the l-th layer. As mentioned earlier, different layers of DNNs are unique can have significantly different number of parameters. Similarly, different layers might need different sparsity budget for the best accuracy. So, we set the trainable pruning parameter for each layer as s_l. That is, $\mathbf{s} = [s_1, \ldots, s_L]$.

Now, using the above mentioned reparameterization for each $\mathbf{W}_l$ and adding a standard L_2 regularization

per layer, we get the following Gradient Descent (GD) update equation at the t-th step for $\mathbf{W}_l,\ \forall\, l \in [L]$:

$$\begin{aligned}\mathbf{W}_l^{(t+1)} &\leftarrow (1 - \eta_t \cdot \lambda)\mathbf{W}_l^{(t)} \\ &- \eta_t \nabla_{\mathcal{S}_g(\mathbf{W}_l, s_l)} \mathcal{L}(\mathcal{S}_g(\mathcal{W}^{(t)}, \boldsymbol{s}), \mathcal{D}) \odot \nabla_{\mathbf{W}_l} \mathcal{S}_g(\mathbf{W}_l, s_l),\end{aligned} \tag{5.3}$$

where η_t is the learning rate at the t-th step, and λ is the L_2 regularization (weight-decay) hyper-parameter. $\nabla_{\mathbf{W}_l} \mathcal{S}_g(\mathbf{W}_l, s_l)$ is the gradient of $\mathcal{S}_g(\mathbf{W}_l, s_l)$ w.r.t. $\mathbf{W}_l$.

Now, $\mathcal{S}$ is non-differentiable, so we use sub-gradient which leads to the following update equation:

$$\begin{aligned}\mathbf{W}_l^{(t+1)} &\leftarrow (1 - \eta_t \cdot \lambda)\mathbf{W}_l^{(t)} \\ &- \eta_t \nabla_{\mathcal{S}_g(\mathbf{W}_l, s_l)} \mathcal{L}(\mathcal{S}_g(\mathcal{W}^{(t)}, \boldsymbol{s}), \mathcal{D}) \odot \mathbf{1}\left\{\mathcal{S}_g(\mathbf{W}_l^{(t)}, s_l) \neq 0\right\},\end{aligned} \tag{5.4}$$

where $\mathbf{1}\{\cdot\}$ is the indicator function and $A \odot B$ denotes element-wise (Hadamard) product of tensors A and B.

Now, if g is a continuous function, then using the STR (5.2) and (5.1), it is clear that $\mathcal{L}(\mathcal{S}_g(\mathcal{W}, \boldsymbol{s}), \mathcal{D})$ is a continuous function of $\boldsymbol{s}$. Further, sub-gradient of $\mathcal{L}$ w.r.t. $\boldsymbol{s}$, can be computed and uses for gradient descent on $\boldsymbol{s}$ as well; see Appendix A.2 of Kusupati et al. [145]. Algorithm 1 in the Appendix of Kusupati et al. [145] shows the implementation of STR on 2D convolution along with extensions to global, per-filter & per-weight sparsity. STR can be modified and applied on the eigenvalues of a parameter tensor, instead of individual entries mentioned above, resulting in low-rank tensors; see Section 5.5.2 for further details. Note that $\boldsymbol{s}$ also has the same weight-decay parameter λ.

Naturally, g plays a critical role here, as a sharp g can lead to an arbitrary increase in threshold leading to poor accuracy while a flat g can lead to slow learning. Practical considerations for choice of g are discussed in Appendix A.1 of Kusupati et al. [145]. For the experiments, g is set as the Sigmoid function for unstructured sparsity and the exponential function for structured sparsity. Typically, $\{s_l\}_{l\in[L]}$ are initialized with s_{init} to ensure that the thresholds $\{\alpha_l = g(s_l)\}_{l\in[L]}$ start close to 0. Figure 5.1 shows that the thresholds' dynamics are guided by a combination of gradients from $\mathcal{L}$ and the weight-decay on $\boldsymbol{s}$. Further, the overall sparsity budget for STR is not set explicitly. Instead, it is controlled by the weight-decay parameter (λ), and can be further fine-tuned using s_{init}. Interestingly, this curve is similar to the handcrafted heuristic for thresholds

defined in [189]. Figure 5.2 shows the overall learnt sparsity budget for ResNet50 during training. The curve looks similar to GMP [282] sparsification heuristic, however, STR learns it via backpropagation and SGD.

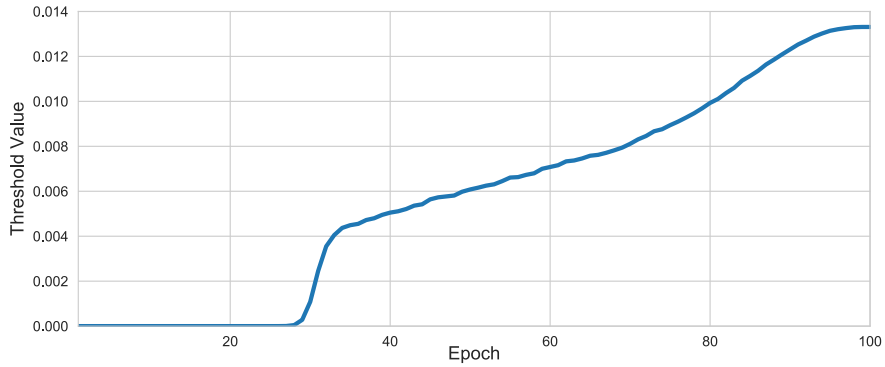


Figure 5.1: The learnt threshold parameter, $\alpha = g(s)$, for layer 10 in 90% sparse ResNet50 on ImageNet-1K over the course of training.

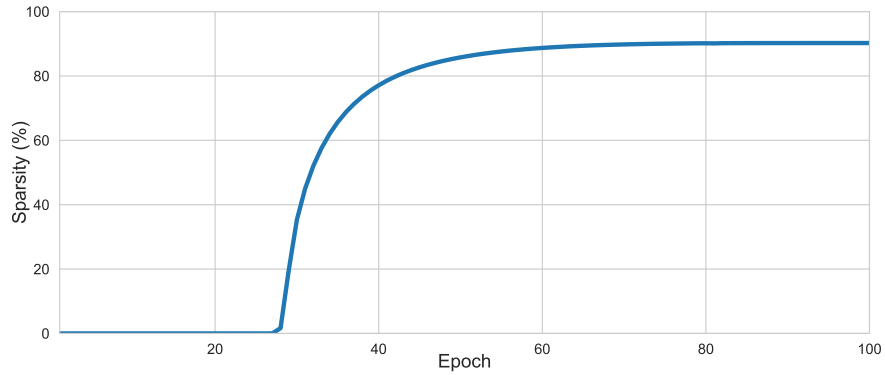


Figure 5.2: The progression of the learnt overall budget for 90% sparse ResNet50 on ImageNet-1K over the course of training.

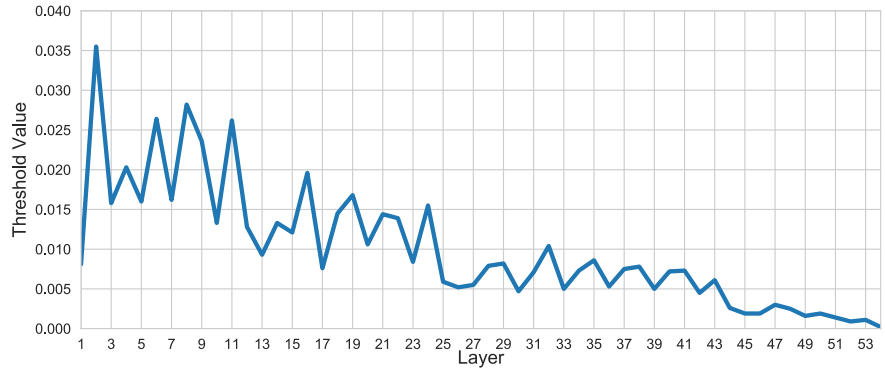


Figure 5.3: The final learnt threshold values, $[\alpha_l]_{l=1}^{54} = [g(s_l)]_{l=1}^{54}$, for all the layers in 90% sparse ResNet50 on ImageNet-1K.

Finally, each parameter tensor learns a different threshold value, $\{\alpha_l\}_{l\in[L]}$, resulting in unique final thresholds across the layers, as shown in Figure 5.3 for ResNet50. This, in turn, results in the non-uniform sparsity budget (see Figure 5.6) which is empirically shown to be effective in increasing prediction accuracy while reducing FLOPs. Moreover, (5.4) shows that the gradient update itself is sparse as gradient of $\mathcal{L}$ is multiplied with an indicator function of $\mathcal{S}_g(\mathbf{W}_l) \neq 0$ which gets sparser over iterations (Figure 5.2). So STR addresses both the issues with standard PGD methods (Hard/Soft Thresholding) that we mentioned above.

5.4.1 Analysis

The reparameterization trick using the projection operator's functional form can be used for standard constrained optimization problems as well (assuming the projection operator has a closed-form). However, it is easy to show that in general, such a method need not converge to the optimal solution even for convex functions over convex sets. This raises a natural question about the effectiveness of the technique for sparse weights learning problem. It turns out that for sparsity constrained problems, STR is very similar to backward pruning [96] which is a well-known technique for sparse regression. Note that, similar to Hard/Soft Thresholding, standard backward pruning also does not support differentiable tuning thresholds which makes it challenging to apply it to DNNs.

To further establish this connection, let's consider a standard sparse regression problem where $\mathbf{y} = \mathbf{X}\mathbf{w}^*$, $\mathbf{X}_{ij} \sim \mathcal{N}(0,1)$, and $\mathbf{X} \in \mathbb{R}^{n\times d}$. $\mathbf{w}^* \in \{0,1\}^d$ has $r \ll d$ non-zeros, and $d \gg n \gg r\log d$. Due to the initialization, $g(s) \approx 0$ in initial few iterations. So, gradient descent converges to the least ℓ_2-norm regression solution. That is, $\mathbf{w} = \mathbf{U}\mathbf{U}^T\mathbf{w}^*$ where $\mathbf{U} \in \mathbb{R}^{d\times n}$ is the right singular vector matrix of $\mathbf{X}$ and is a random n-dimensional subspace. As $\mathbf{U}$ is a random subspace. Since $n \gg r\log d$, $\mathbf{U}_S\mathbf{U}_S^T \approx \frac{r}{d}\cdot I$ where $S = \text{supp}(\mathbf{w}^*)$, and $\mathbf{U}_S$ indexes rows of $\mathbf{U}$ corresponding to S. That is, $\min_{j\in S}\left|\mathbf{U}_j\cdot\mathbf{U}^T\mathbf{w}^*\right| \geq 1 - o(1)$. On the other hand, $\left|\mathbf{U}_j\cdot\mathbf{U}_S^T\mathbf{w}^*\right| \lesssim \frac{\sqrt{nr}}{d}\sqrt{\log d}$ with high probability for $j \notin S$. As $n \gg r\log d$, almost all the elements of $\text{supp}(\mathbf{w}^*)$ will be in top $\mathcal{O}(n)$ elements of $\mathbf{w}$. Furthermore, $\mathbf{X}\mathcal{S}_g(\mathbf{w}, s) = \mathbf{y}$, so $|s|$ would decrease significantly via weight-decay and hence $g(s)$ becomes large enough to prune all but say $\mathcal{O}(n)$ elements. Using a similar argument as above, leads to further pruning of $\mathbf{w}$, while ensuring recovery of almost all elements in $\text{supp}(\mathbf{w}^*)$.

5.5 Experiments

This section showcases the experimentation followed by the observations from applying STR for (a) unstructured sparsity in CNNs and (b) structured sparsity in RNNs.

5.5.1 Unstructured Sparsity in CNNs

Experimental Setup

ImageNet-1K [55] is a widely used large-scale image classification dataset with 1K classes. All the CNN experiments presented are on ImageNet-1K. ResNet50 [98] and MobileNetV1 [111] are two popular CNN architectures. ResNet50 is extensively used in literature to show the effectiveness of sparsity in CNNs. Experiments on MobileNetV1 argue for the generalizability of the proposed technique (STR). Dataset and models' details can be found in Appendix A.7 of Kusupati et al. [145].

STR was compared against strong state-of-the-art baselines in various sparsity regimes including GMP [79], DSR [186], DNW [266], SNFS [59], RigL [73] and DPF [164]. GMP and DNW always use a uniform sparsity budget. RigL, SNFS, DSR, and DPF were compared in their original form. Exceptions for the uniform sparsity are marked in Table 5.1. The "+ ERK" suffix implies the usage of ERK budget [73] instead of the original sparsity budget. Even though VD [184] achieves state-of-the-art results, it is omitted due to the $2\times$ memory and $4\times$ compute footprint during training. Typically VD and IMP use a global threshold for global sparsity (GS) [93] which can also be learnt using STR. The unstructured sparsity experiments presented compare the techniques which induce layer-wise sparsity. Note that STR is generalizable to other scenarios as well. Open-source implementations, pre-trained models, and reported numbers of the available techniques were used as the baselines. Experiments were run on a machine with 4 NVIDIA Titan X (Pascal) GPUs.

All baselines use the hyperparameter settings defined in their implementations/papers. The experiments for STR use a batch size of 256, cosine learning rate routine and are trained for 100 epochs following the hyperparameter settings in [266] using SGD + momentum. STR has weight-decay (λ) and s_{init} hyperparameters to control the overall sparsity in CNNs and can be found in Appendix A.6 of Kusupati et al. [145]. $\text{GMP}_{1.5\times}$ [79] and $\text{RigL}_{5\times}$ [73] show that training the networks longer increases accuracy. However, due to

the limited compute and environmental concerns [223], all the experiments were run only for around 100 epochs (∼3 days each). Unstructured sparsity in CNNs with STR is enforced by learning one threshold per-layer as shown in Figure 5.3. PyTorch STRConv code can be found in Algorithm 1 of Appendix of Kusupati et al. [145].

ResNet50 on ImageNet-1K

A fully dense ResNet50 trained on ImageNet-1K has 77.01% top-1 validation accuracy. STR is compared extensively to other baselines on ResNet50 in the sparsity ranges of 80%, 90%, 95%, 96.5%, 98%, and 99%. Table 5.1 shows that DNW and GMP are state-of-the-art among the baselines across all the aforementioned sparsity regimes. As STR might not be able to get exactly to the sparsity budget, numbers are reported for the models which nearby. Note that the 90.23% sparse ResNet50 on ImageNet-1K with STR is referred to as the 90% sparse ResNet50 model learnt with STR.

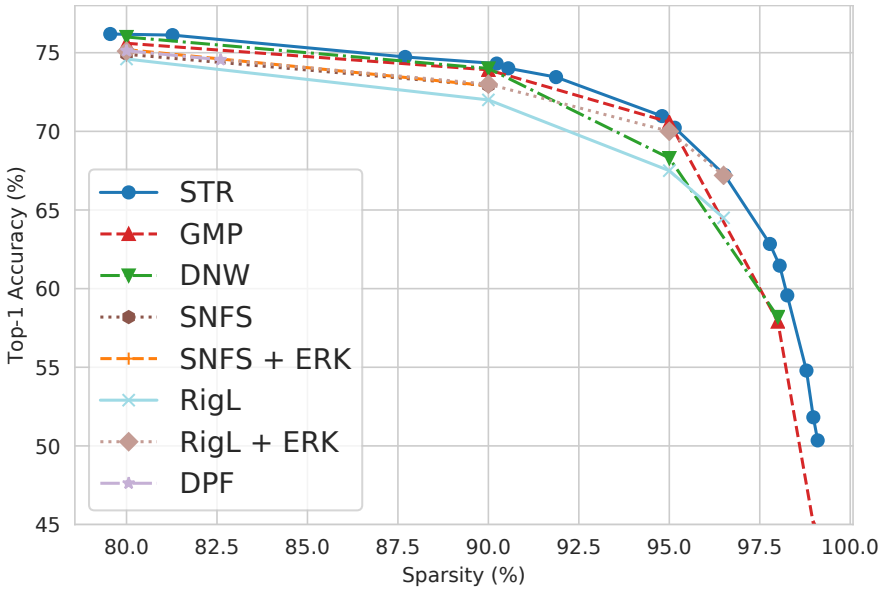


Figure 5.4: STR forms a frontier curve over all the baselines in all sparsity regimes showing that it is the state-of-the-art for unstructured sparsity in ResNet50 on ImageNet-1K.

STR comfortably beats all the baselines across all the sparsity regimes as seen in Table 5.1 and is the state-of-the-art for unstructured sparsity. Figure 5.4 shows that STR forms a frontier curve encompassing all the baselines at all the levels of sparsity. Very few methods are stable in the ultra sparse regime of 98-99% sparsity and GMP can achieve 99% sparsity. STR is very stable even in the ultra sparse regime, as shown in

Table 5.1 and Figure 5.4, while being up to 10% higher in accuracy than GMP at 99% sparsity.

STR induces non-uniform sparsity across layers, Table 5.1 and Figure 5.5 show that STR produces

Table 5.1: STR is the state-of-the-art for unstructured sparsity in ResNet50 on ImageNet-1K while having lesser inference cost (FLOPs) than the baselines across all the sparsity regimes. * and $^\#$ imply that the first and last layer are dense respectively. Baseline numbers reported from their respective papers/open-source implementations and models. FLOPs do not include batch-norm.

Method	Top-1 Acc (%)	Params	Sparsity (%)	FLOPs
ResNet-50	77.01	25.6M	0.00	4.09G
GMP	75.60	5.12M	80.00	818M
DSR$^{*\#}$	71.60	5.12M	80.00	1.23G
DNW	76.00	5.12M	80.00	818M
SNFS	74.90	5.12M	80.00	-
SNFS + ERK	75.20	5.12M	80.00	1.68G
RigL*	74.60	5.12M	80.00	920M
RigL + ERK	75.10	5.12M	80.00	1.68G
DPF	75.13	5.12M	80.00	818M
STR	**76.19**	5.22M	79.55	**766M**
STR	**76.12**	**4.47M**	**81.27**	**705M**
GMP	73.91	2.56M	90.00	409M
DNW	74.00	2.56M	90.00	409M
SNFS	72.90	2.56M	90.00	1.63G
SNFS + ERK	72.90	2.56M	90.00	960M
RigL*	72.00	2.56M	90.00	515M
RigL + ERK	73.00	2.56M	90.00	960M
DPF$^\#$	74.55	4.45M	82.60	411M
STR	**74.73**	3.14M	87.70	**402M**
STR	**74.31**	**2.49M**	**90.23**	**343M**
STR	**74.01**	**2.41M**	**90.55**	**341M**
GMP	70.59	1.28M	95.00	204M
DNW	68.30	1.28M	95.00	204M
RigL*	67.50	1.28M	95.00	317M
RigL + ERK	70.00	1.28M	95.00	∼600M
STR	**70.97**	1.33M	94.80	**182M**
STR	70.40	**1.27M**	**95.03**	**159M**
STR	70.23	**1.24M**	**95.15**	**162M**
RigL*	64.50	0.90M	96.50	257M
RigL + ERK	67.20	0.90M	96.50	∼500M
STR	**67.78**	0.99M	96.11	**127M**
STR	**67.22**	**0.88M**	**96.53**	**117M**
GMP	57.90	0.51M	98.00	82M
DNW	58.20	0.51M	98.00	82M
STR	**62.84**	0.57M	97.78	**80M**
STR	**61.46**	**0.50M**	**98.05**	**73M**
STR	**59.76**	**0.45M**	**98.22**	**68M**
GMP	44.78	0.26M	99.00	41M
STR	**54.79**	0.31M	98.79	54M
STR	**51.82**	0.26M	98.98	47M
STR	**50.35**	**0.23M**	**99.10**	44M

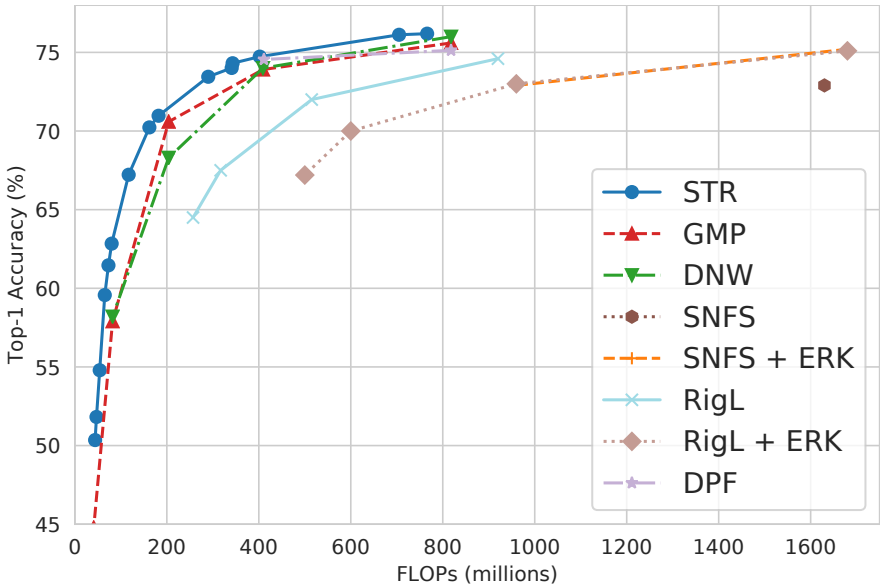


Figure 5.5: STR results in ResNet50 models on ImageNet-1K which have the lowest inference cost (FLOPs) for any given accuracy.

models which have lower or similar inference FLOPs compared to the baselines while having better prediction accuracy in all the sparsity regimes. This hints at the fact that STR could be redistributing the parameters thereby reducing the FLOPs. In the 80% sparse models, STR is at least 0.19% better in accuracy than the baselines while having at least 60M (6.5%) lesser FLOPs. Similarly, STR has state-of-the-art accuracy in 90%, 95%, and 96.5% sparse regimes while having at least 68M (16.5%), 45M (22%) and 140M (54%) lesser FLOPs than the best baselines respectively. In the ultra sparse regime of 98% and 99% sparsity, STR has similar or slightly higher FLOPs compared to the baselines but is up to 4.6% and 10% better in accuracy respectively. Table 5.1 summarizes that the non-uniform sparsity baselines like SNFS, SNFS+ERK, and RigL+ERK can have up to 2-4× higher inference cost (FLOPs) due to non-optimal layer-wise distribution of the parameter weights.

Observations: STR on ResNet50 shows some interesting observations related to sparsity and inference cost (FLOPs). These observations will be further discussed in Section 5.6:

1. STR is state-of-the-art for unstructured sparsity.
2. STR minimizes inference cost (FLOPs) while maintaining accuracy in the 80-95% sparse regime.
3. STR maximizes accuracy while maintaining inference cost (FLOPs) in 98-99% ultra sparse regime.
4. STR learns a non-uniform layer-wise sparsity, shown in Figure 5.6, which shows that the initial layers of

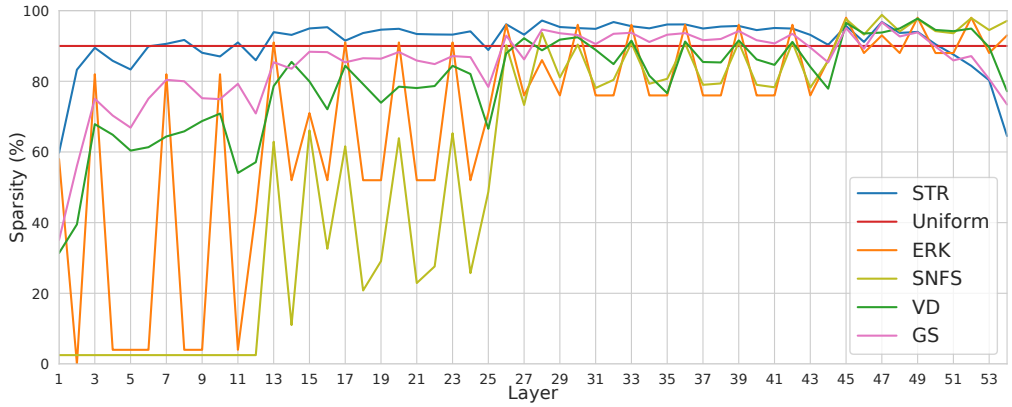


Figure 5.6: Layer-wise sparsity budget for the 90% sparse ResNet50 models on ImageNet-1K using various sparsification techniques.

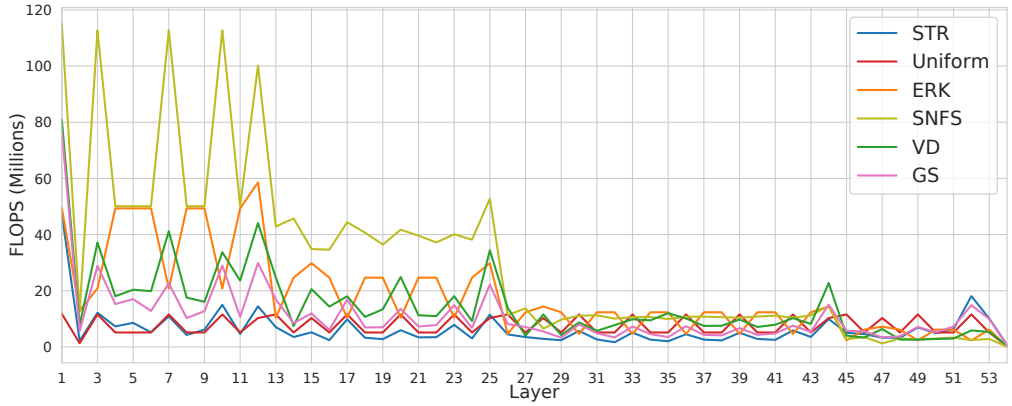


Figure 5.7: Layer-wise FLOPs budget for the 90% sparse ResNet50 models on ImageNet-1K using various sparsification techniques.

the CNN can be sparser than that of the existing non-uniform sparsity methods. All the learnt non-uniform budgets through STR can be found in Appendix A.3 of Kusupati et al. [145].

5. Figure 5.6 also shows that the last layers through STR are denser than that of the other methods which is contrary to the understanding in the literature of non-uniform sparsity [186, 59, 73, 79]. This leads to a sparser backbone for transfer learning. The backbone sparsities can be found in Appendix A.3 of Kusupati et al. [145].

6. Figure 5.7 shows the layer-wise FLOPs distribution for the non-uniform sparsity methods. STR adjusts the FLOPs across layers such that it has lower FLOPs than the baselines. Note that the other non-uniform sparsity budgets lead to heavy compute overhead in the initial layers due to denser parameter tensors.

STR can also induce global sparsity (GS) [93] with similar accuracy at $\sim 2\times$ FLOPs compared to layer-wise

Table 5.2: STR is up to 3% higher in accuracy while having 33% lesser inference cost (FLOPs) for MobileNetV1 on ImageNet-1K.

Method	Top-1 Acc (%)	Params	Sparsity (%)	FLOPs
MobileNetV1	71.95	4.21M	0.00	569M
GMP	67.70	1.09M	74.11	163M
STR	**68.35**	**1.04M**	**75.28**	**101M**
STR	66.52	**0.88M**	**79.07**	**81M**
GMP	61.80	0.46M	89.03	82M
STR	**64.83**	0.60M	85.80	**55M**
STR	**62.10**	0.46M	89.01	**42M**
STR	61.51	**0.44M**	**89.62**	**40M**

for 90-98% sparsity (details in Appendix A.5.1 of Kusupati et al. [145]).

MobileNetV1 on ImageNet-1K

MobileNetV1 was trained on ImageNet-1K for unstructured sparsity with STR to ensure generalizability. Since GMP is the state-of-the-art baseline as shown earlier, STR was only compared to GMP for 75% and 90% sparsity regimes. A fully dense MobileNetV1 has a top-1 accuracy of 71.95% on ImageNet-1K. GMP [282] has the first layer and depthwise convolution layers dense for MobileNetV1 to ensure training stability and maximize accuracy.

Table 5.2 shows the STR is at least 0.65% better than GMP for 75% sparsity, while having at least 62M (38%) lesser FLOPs. More interestingly, STR has state-of-the-art accuracy while having up to 50% (40M) lesser FLOPs than GMP in the 90% sparsity regime. All the observations made for ResNet50 hold for MobileNetV1 as well. The sparsity and FLOPs distribution across layers can be found in Appendix A.4 of Kusupati et al. [145].

5.5.2 Structured Sparsity in RNNs

Experimental Setup

Google-12 is a speech recognition dataset that has 12 classes made from the Google Speech Commands dataset [259]. HAR-2 is a binarized version of the 6-class Human Activity Recognition dataset [8]. These two datasets stand as compelling cases for on-device resource-efficient machine learning at the edge. Details about the datasets can be found in Appendix A.7 of Kusupati et al. [145].

FastGRNN [144] was proposed to enable powerful RNN models on resource-constrained devices. FastGRNN relies on making the RNN parameter matrices low-rank, sparse and quantized. As low-rank is a form of structured sparsity, experiments were done to show the effectiveness of STR for structured sparsity. The input vector to the RNN at each timestep and hidden state have D & $\hat{D}$ dimensionality respectively. FastGRNN has two parameter matrices, $\mathbf{W} \in \mathbb{R}^{D\times\hat{D}}$, $\mathbf{U} \in \mathbb{R}^{\hat{D}\times\hat{D}}$ which are reparameterized as product of low-rank matrices, $\mathbf{W} = \mathbf{W}_1\mathbf{W}_2$, and $\mathbf{U} = \mathbf{U}_1\mathbf{U}_2$ where $\mathbf{W}_1 \in \mathbb{R}^{D\times r_W}$, $\mathbf{W}_2 \in \mathbb{R}^{r_W\times\hat{D}}$, and $(\mathbf{U}_1)^\top, \mathbf{U}_2 \in \mathbb{R}^{r_U\times\hat{D}}$. r_W, r_U are the ranks of the respective matrices. In order to apply STR, the low-rank reparameterization can be changed to $\mathbf{W} = (\mathbf{W}_1 \odot \mathbf{1}\mathbf{m}_\mathbf{W}^\top)\mathbf{W}_2$, and $\mathbf{U} = (\mathbf{U}_1 \odot \mathbf{1}\mathbf{m}_\mathbf{U}^\top)\mathbf{U}_2$ where $\mathbf{m}_\mathbf{W} = \mathbf{1}_D$, and $\mathbf{m}_\mathbf{U} = \mathbf{1}_{\hat{D}}$, $\mathbf{W}_1 \in \mathbb{R}^{D\times D}$, $\mathbf{W}_2 \in \mathbb{R}^{D\times\hat{D}}$, and $\mathbf{U}_1, \mathbf{U}_2 \in \mathbb{R}^{\hat{D}\times\hat{D}}$. To learn the low-rank, STR is applied on the $\mathbf{m}_\mathbf{W}$, and $\mathbf{m}_\mathbf{U}$ vectors. Learning low-rank with STR on $\mathbf{m}_\mathbf{W}$, $\mathbf{m}_\mathbf{U}$ can be thought as inducing unstructured sparsity on the two trainable vectors aiming for the right r_W, and r_U.

The baseline is low-rank FastGRNN where the ranks of the matrices are preset [144]. EdgeML [57] FastGRNN was used for the experiments with the hyperparameters suggested in the paper and is referred to as vanilla training. Hyperparameters for the models can be found in Appendix A.6 of Kusupati et al. [145].

FastGRNN on Google-12 and HAR-2

Table 5.3 presents the results for low-rank FastGRNN with vanilla training and STR. Full-rank non-reparameterized FastGRNN has an accuracy of 92.60% and 96.10% on Google-12 and HAR-2 respectively. STR outperforms vanilla training by up to 1.67% in four different model-size reducing rank settings on Google-12. Similarly, on HAR-2, STR is better than vanilla training in all the rank settings by up to 2.47%. Note that the accuracy of the low-rank models obtained by STR is either better or on-par with the full rank

Table 5.3: STR can induce learnt low-rank in FastGRNN resulting in up to 2.47% higher accuracy than the vanilla training.

Google-12			HAR-2		
(r_W, r_U)	Accuracy (%)		(r_W, r_U)	Accuracy (%)	
	Vanilla Training	STR		Vanilla Training	STR
Full rank (32, 100)	92.30	-	Full rank (9, 80)	96.10	-
(12, 40)	92.79	**94.45**	(9, 8)	94.06	**95.76**
(11, 35)	92.86	**94.42**	(9, 7)	93.15	**95.62**
(10, 31)	92.86	**94.25**	(8, 7)	94.88	**95.59**
(9, 24)	93.18	**94.45**			

models while being around 50% and 70% smaller in size (low-rank) for Google-12 and HAR-2 respectively.

These experiments for structured sparsity in RNNs show that STR can be applied to obtain low-rank parameter tensors. Similarly, STR can be extended for filter/channel pruning and block sparsity [101, 115, 170] and details for this adaptation can be found in Appendix A.5.2 of Kusupati et al. [145].

5.6 Discussion and Drawbacks

STR's usage for unstructured sparsity leads to interesting observations as noted in Section 5.5.1. It is clear from Table 5.1 and Figures 5.4, 5.5 that STR achieves state-of-the-art accuracy for all the sparsity regimes and also reduces the FLOPs in doing so. STR helps in learning non-uniform sparsity budgets which are intriguing to study as an optimal non-uniform sparsity budget can ensure minimization of FLOPs while maintaining accuracy. Although it is not clear why STR's learning dynamics result in a non-uniform budget that minimizes FLOPs, the reduction in FLOPs is due to the better redistribution of parameters across layers.

Non-uniform sparsity budgets learnt by STR have the initial and middle layers to be sparser than the other methods while making the last layers denser. Conventional wisdom suggests that the initial layers should be denser as the early loss of information would be hard to recover, this drives the existing non-uniform sparsity heuristics. As most of the parameters are present in the deeper layers, the existing methods tend to make them sparser while not affecting the FLOPs by much. STR, on the other hand, balances the FLOPs and sparsity across the layers as shown in Figures 5.6, 5.7 making it a lucrative and efficient choice. The denser final layers along with sparser initial and middle layers point to sparser CNN backbones obtained using STR. These sparse backbones can be viable options for efficient representation/transfer learning for downstream tasks.

Table 5.4: Effect of various layer-wise sparsity budgets when used with DNW for ResNet50 on ImageNet-1K.

Method	Top-1 Acc (%)	Params	Sparsity (%)	FLOPs
Uniform	74.00	2.56M	90.00	409M
ERK	**74.10**	2.56M	90.00	960M
Budget from STR	74.01	**2.49M**	**90.23**	**343M**
Uniform	68.30	1.28M	95.00	204M
Budget from STR	**69.72**	1.33M	94.80	182M
Budget from STR	68.01	**1.24M**	**95.15**	**162M**

Table 5.4 shows the effectiveness/transferability of the learnt non-uniform budget through STR for 90% sparse ResNet50 on ImageNet-1K using DNW [266]. DNW typically takes in a uniform sparsity budget and has an accuracy of 74% for a 90% sparse ResNet50. Using ERK non-uniform budget for 90% sparsity results in a 0.1% increase in accuracy at the cost 2.35× inference FLOPs. Training DNW with the learnt budget from STR results in a reduction of FLOPs by 66M (16%) while maintaining accuracy. In the 95% sparsity regime, the learnt budget can improve the accuracy of DNW by up to 1.42% over uniform along with a reduction in FLOPs by at least 22M (11%).

Table 5.5: Effect of various layer-wise sparsity budgets when used with GMP for ResNet50 on ImageNet-1K.

Method	Top-1 Acc (%)	Params	Sparsity (%)	FLOPs
Uniform	73.91	2.56M	90.00	409M
Budget from STR	**74.13**	**2.49M**	**90.23**	**343M**
Uniform	57.90	0.51M	98.00	82M
Budget from STR	**59.47**	**0.50M**	**98.05**	**73M**

Similarly, these budgets can also be used for other methods like GMP [282]. Table 5.5 shows that the learnt sparsity budgets can lead to an increase in accuracy by 0.22% and 1.57% in 90% and 98% sparsity regimes respectively when used with GMP. Accuracy gains over uniform sparsity are also accompanied by a significant reduction in inference FLOPs. Note that the learnt non-uniform sparsity budgets can also be obtained using smaller representative datasets instead of expensive large-scale experiments.

The major drawback of STR is the tuning of the weight-decay parameter, λ and finer-tuning with s_{init} to obtain the targeted overall sparsity. One way to circumvent this issue is to freeze the non-uniform sparsity distribution in the middle of training when the overall sparsity constraints are met and train for the remaining epochs. This might not potentially give the best results but can give a similar budget which can be then transferred to methods like GMP or DNW. Another drawback of STR is the function g for the threshold. The stability, expressivity, and sparsification capability of STR depends on g. However, it should be noted that sigmoid and exponential functions work just fine, as g, for STR.

5.7 Conclusions

This paper proposed Soft Threshold Reparameterization (STR), a novel use of the soft-threshold operator, for the weights in DNN, to smoothly induce sparsity while learning layer-wise pruning thresholds thereby obtaining a non-uniform sparsity budget. Extensive experimentation showed that STR is state-of-the-art for unstructured sparsity in CNNs for ImageNet-1K while also being effective for structured sparsity in RNNs. Our method results in sparse models that have significantly lesser inference costs than the baselines. In particular, STR achieves the same accuracy as the baselines for 90% sparse MobileNetV1 with 50% lesser FLOPs. STR has $\sim$10% higher accuracy than the existing methods in ultra sparse (99%) regime for ResNet50 showing the effectiveness of the learnt non-uniform sparsity budgets. STR can also induce low-rank structure in RNNs while increasing the prediction accuracy showing the generalizability of the proposed reparameterization. Finally, STR is easy to adapt and the learnt budgets are transferable.

Chapter 6

LLC: Accurate, Multi-purpose Learnt Low-dimensional Binary Codes

6.1 Overview

Learning binary representations of instances and classes is a classical problem with several high potential applications. In modern settings, the compression of high-dimensional neural representations to low-dimensional binary codes is a challenging task and often require large bit-codes to be accurate. In this chapter, we propose a novel method for **L**earning **L**ow-dimensional binary **C**odes (LLC) for instances as well as classes. Our method does *not* require any side-information, like annotated attributes or label meta-data, and learns extremely low-dimensional binary codes ($\approx$ 20 bits for ImageNet-1K). The learnt codes are super-efficient while still ensuring *nearly optimal* classification accuracy for ResNet50 on ImageNet-1K. We demonstrate that the learnt codes capture intrinsically important features in the data, by discovering an intuitive taxonomy over classes. We further quantitatively measure the quality of our codes by applying it to the efficient image retrieval as well as out-of-distribution (OOD) detection problems. For ImageNet-100 retrieval problem, our learnt binary codes outperform 16 bit HashNet using only 10 bits and also are as accurate as 10 dimensional real representations. Finally, our learnt binary codes can perform OOD detection, out-of-the-box, as accurately as a baseline that needs $\approx$ 3000 samples to tune its threshold, while we require *none*. Code is open-sourced at https://github.com/RAIVNLab/LLC.

6.2 Introduction

Embedding data in low-dimensional binary space is a long-standing machine learning problem [256]. The problem has received a lot of interest in the computer vision (CV) domain, where the goal is to find binary codes that capture the key semantics of the image, like, objects present in the image or interpretable attributes. Section 6.3 covers the literature on learning binary codes and their applications.

In addition to learning semantically meaningful representations of the instances, low-dimensional binary codes allow efficiency in a variety of large-scale machine learning (ML) applications. Low-dimensional codes are crucial in extreme classification with millions of classes [23, 122, 48] and also critical in efficient large-scale retrieval settings [167, 50, 264].

Compressing information into binary codes is challenging due to its highly non-smooth nature while requiring the preservation of relevant information in an instance/class. This might explain the lack of good classification accuracy for existing classification-based embedding techniques [113, 45]. To address that, traditional methods often relied on side-information like attributes to construct class codes and then use that to learn the instance codes [56, 4].

Learning binary embeddings can be posed in a variety of formulations like pairwise optimization [142] or unsupervised learning [36, 216], in this work we focus on learning binary codes using a given labeled multi-class dataset, e.g., ImageNet-1K. This allows us to couple the representation (code) learning of both *instances* and *classes* thus enabling us to capture the underlying semantic structure efficiently to assist in downstream tasks like classification, retrieval etc.

We propose LLC, a method to learn *both* class and instance codes via the standard classification task and its setup *without any side-information*. Our Learning Low-dimensional binary Codes (LLC) technique, formulates the embedding (code) learning problem as that of learning a low-dimensional binary embedding of a standard deep neural "backbone". Instead of directly training for the low-dimensional binary instance codes, we propose a two-phase approach. In the first phase, LLC learns low-dimensional (k-bit) binary codes for classes that capture semantic information through a surrogate classification task. Then in the second phase, LLC uses these learnt class codes as an efficient alternative to learning instance codes in sub-linear cost (in the number of classes, L) using the Error-Correcting Output Codes (ECOC) approach [64]. This two-phase pipeline helps in the effective distillation of required semantic similarity between instances through the learnt

class codes. For example, on ImageNet-1K with ResNet50, LLC is able to learn tight 20-bit codes that can be used for *efficient classification* and achieve 74.5% accuracy compared to the standard baseline 77% on ImageNet-1K (Section 6.5.1). Furthermore, we observe that the learnt 20-bit class codes capture intuitive taxonomy over classes (Figure 6.1) while the instance codes retain the distilled class similarity information useful in efficient retrieval and OOD detection.

Retrieval. To further study, the effectiveness of our learnt binary codes, we apply them to hashing-based efficient retrieval, where the goal is to retrieve a large number of similar instances with the same class label in top retrieved samples. Deep supervised hashing is a widely studied problem with several recent results [34, 234] which are designed *specifically* for the learnt hashing-based retrieval. Interestingly, our learnt instance codes through the LLC routine provide strikingly better performance while not being learnt explicitly for hashing. For eg., using AlexNet, with just 32-bit codes we are can provide 5.4% more accurate retrieval than HashNet's 64-bit codes on ImageNet-100 (Section 6.5.2).

OOD Detection. We similarly apply LLC based learnt binary codes to detect OOD instances [104]. We adopt a simple approach based on our binary codes: if an instance is not within a Hamming distance of 1 to any class codes, we classify it as OOD. That is, we do not fine-tune our OOD detector for the new domain, which is critical in practical settings. In contrast, baseline techniques for OOD detection require a few samples (eg., ≈ 3000 for ImageNet-750) from the OOD domain to fine-tune thresholds, while we require *no* samples yet reaching similar OOD detection (Section 6.5.3).

In this work, we make the following key contributions:

- LLC method to learn semantically-rich low-dimensional binary codes for both classes & instances.
- Show that the learnt codes enable accurate & efficient classification: ImageNet-1K with 20-bits.
- Apply LLC to image retrieval task, and demonstrate that it comfortably outperforms the instance code learning methods for hashing-based retrieval on ImageNet-100.
- Finally, use codes from LLC for strong & sample efficient OOD detection in practical settings.

6.3 Related Work

Binary class codes were originally aimed at sub-linear training and prediction for multi-class classification. The Error-Correcting Output Codes (ECOC) framework [64, 7, 72] reformulated multi-class classification as multi-label classification using k-bit codes per class (codebook). The learning of optimal codebook is NP-complete [47] which lead to use of random codebooks [113, 45] in traditional ML. However, there were a few codebook learning [16, 280, 263, 13] and construction schemes using side-information from other modalities [4]. The lack of a strong learnable feature extractor often deterred the gains these codebooks provide for the classification and effective learning of instance codes. Attribute annotations can also help in constructing class codes [5]. These binary codes are either explicitly annotated [75] or discovered [205, 76]. Attributed-based learning also ties into leveraging the class codes for zero/few-shot learning [150, 151, 4, 193] expecting some form of interpretability.

Most methods that use class codes as supervision can produce instance codes [64]. However, the standalone literature of instance codes comes from requirements in large-scale application like retrieval (hashing). In the past, most hashing techniques that created instance codes were based on random projections [50, 45, 44], semantics [217, 56] or learnt through metric learning [142, 141, 192, 140], clustering [261, 216] and quantization [84]. Deep learning further helped in learning more accurate hashing functions to generate instance codes either in an unsupervised [36, 227] or supervised [167, 34, 234, 276] fashion. We refer to [176, 279, 256] for a more thorough review on deep hashing methods.

Finally, embedding-based classification [45, 271, 23, 88] enables joint low-dimensional representation learning for both classes and instances with an eye on sub-linear training and prediction costs. After distilling the key ideas from the literature, we aim to a) learn semantically rich low-dimensional representations for both classes and instances together, b) have these representations in the binary space, and c) do this with minimal dependence on side-information or metadata.

LLC, to the best of our knowledge - for the first time, jointly learns low-dimensional binary codes for both classes and instances using a surrogate classification task, without any side-information (Section 6.4). The learnt class codes capture intrinsic information at the semantic level that helps in discovering an intuitive taxonomy over classes (Figure 6.1). The learnt class codes then anchor the instance code learning which results in tight and accurate low-dimensional instance codes further used in retrieval (Section 6.5.2). Finally,

both the learnt class and instance codes power extremely efficient yet accurate classification (Section 6.5.1) and out-of-distribution detection (Section 6.5.3).

6.4 Learning Low-dimensional Binary Codes

The goal is to learn a binary embedding (code) function $g\colon \mathcal{X} \to \{-1, 1\}^k$ where $\mathcal{X}$ is the input domain and k is the dimensionality of the code. We focus on learning embeddings using a labelled multi-class data [113]. That is, suppose we are given a labelled dataset $\mathcal{D} = \{(x_1, y_1), \ldots, (x_n, y_n)\}$ where $x_i \in \mathcal{X}$ is an input point and $y_i \in [L]$ is the label of x_i for all $i \in [n]$. Then, the goal is to learn an instance embedding function $g\colon \mathcal{X} \to \{+1, -1\}^k$ *and* class embeddings $h_q \in \{+1, -1\}^k$ for all $q \in [L]$ such that $g(x_i) = h_{y_i}$ and $g(x_i) = g(x_j)$ if and only if $y_i = y_j$.

Intuitively, for large-scale datasets, $g(x)$ and h_q should capture key semantic information to provide accurate classification, thus allowing their use in application domains like retrieval or OOD detection. Note that while we present our technique for learning embeddings using multi-class datasets, it applies more generally to multi-labeled datasets as well.

Instance and Class Code Parameterization. For learning such embedding function, we assume access to a deep neural architecture $F(\,\cdot\,; \theta_F)\colon \mathcal{X} \to \mathbb{R}^d$ that maps the input $x \in \mathcal{X}$ to a d-dimensional real-valued representation. θ_F is a learnable parameterization of the network; we drop θ_F from F wherever the meaning is clear from the context. For example, ResNet50 is one such network that encodes 224×224 RGB images into $d = 2048$ dimensions.

Now, given a network F and $x \in \mathcal{X}$, we formulate embedding function of x and the corresponding multiclass prediction scores $\hat{y} \in \mathbb{Z}^L$ as:

$$g(x) \coloneqq B\left(\mathbf{P} \cdot F(x; \theta_F)\right), \quad \hat{y} \coloneqq B(\mathbf{C}) \cdot g(x)\,, \tag{6.1}$$

where $\mathbf{P} \in \mathbb{R}^{k \times d}$ maps $F(x)$ into k-dimensions and $B(a) = \operatorname{sign}(a) \in \{+1, -1\}$ is the standard binarization/sign operator applied elementwise (with the assumption $\operatorname{sign}(0) = +1$). Finally, $\mathbf{C} \in \mathbb{R}^{L \times k}$, and $\hat{y} = B(\mathbf{C}) \cdot g(x)$ represents the scores of each class for an input x. Note that for a class $\ell \in [L]$, $B(\mathbf{C}_\ell)$

(where $\mathbf{C}_\ell$ represents the ℓ-th row of $\mathbf{C}$) is the learnt binary class embedding (code) of class $\ell \in [L]$, and $g(x) = B(\mathbf{P} \cdot F(x; \theta_F))$ is the learnt instance embedding (code) of instance x. Note that (6.1) is a general purpose formulation for the problem of learning class and instance codes.

6.4.1 The LLC Method

Phase 1: *Codebook Learning* – $B(\mathbf{C})$. Given labelled examples $\mathcal{D}$, we use standard empirical risk minimization to learn a multi-class classifier, i.e., we solve

$$\min_{\mathbf{C},\mathbf{P},\theta_F} \sum_{(x_i,y_i)\in\mathcal{D}} \mathcal{L}\left(B(\mathbf{C}) \cdot (\mathbf{P} \cdot F(x_i; \theta_F))\ ;\ y_i\right)\ , \tag{6.2}$$

where $\mathcal{L}\colon \mathbb{R}^L \times [L] \to \mathbb{R}_+$ is the standard multi-class softmax cross-entropy loss function. This is a standard optimization problem that can be solved using standard gradient descent methods or other sub-gradient based optimizers. However, one challenge is that $B(\mathbf{C})$ is a binary matrix and B is a binary function, so the gradients are 0 almost everywhere. Instead, we use the Straight-Through Estimator (STE) [18] technique popular in binary neural networks domain [206], to optimize for $\mathbf{C}$ through the binarization. Intuitively, STE uses binarization/sign function in the forward pass, but in the backpropagation phase, it allows the gradients to flow straight-through as if it were real-valued. The codebook, $B(\mathbf{C})$ refers to the collection of k-bit class codes learnt in this process.

For ImageNet-1K, we learnt unique binary codes, $B(\mathbf{C}_\ell)$, for every class $\ell \in [L]$ of the 1000 classes using only 20-bits, only twice the information-theoretic limit. As with the class representations from a linear classifier, these class codes do capture intrinsically important features that help in discovering intuitive taxonomy over classes (Section 6.4.2) among various applications (Section 6.5).

Phase 2: *Instance Code Learning* – $B(\mathbf{P} \cdot F(x; \theta_F))$. Several existing techniques model $\mathbf{C}$ and $\mathbf{P}$ in different ways to learn an embedding function similar to (6.1). However, these methods often try to only learn instance codes and have challenges in maintaining high accuracy [34, 36] in a variety of applications because optimization problem (6.2) is challenging and might lead to significantly sub-optimal classification error. For example, for ImageNet-1K classification with ResNet50, the accuracy for our trained model (20-bits) at this stage is 72.5% compared to the standard 77%.

To remedy this, we further optimize our embeddings using the ECOC framework [64] for multi-class classification, which essentially transforms the multi-class problem into a multi-label problem, which in turn is k independent binary classification problems. That is, we use the k-bit codes learnt for each class as the supervision to further train $F(\,\cdot\,;\theta_F)$ and $\mathbf{P}$:

$$\min_{\theta_F,P} \sum_{(x_i,y_i)\in\mathcal{D}} \sum_{j=1}^{k} \text{BCE}\left(\sigma(\mathbf{P}_j \cdot F(x_i;\theta_F))\;;\; (B(\mathbf{C}_{y_i,j})+1)/2\right), \tag{6.3}$$

where σ is the sigmoid/logistic function, BCE is the binary cross-entropy loss between the j-th bit of instance x_i's embedding, and the j-th bit extracted from the class embedding of it's label y_i (the function $z \mapsto (z+1)/2$ is used to map $\{+1,-1\}$ to $\{1,0\}$ to make it a simple binary classification problem per each bit). We use gradient based optimization to learn θ_F and $\mathbf{P}$. As mentioned earlier, ECOC framework allows us to correct errors in classification. For example, with just 20 bits on ImageNet-1K dataset, the method now achieves 74.5% accuracy with ResNet50 backbone.

The advantage of this two-phase pipeline where we start with a codebook learning for classes is that the cost of learning instances codes reduces to a bottleneck of k-dims ($\ll L$) instead of the usual L . Furthermore, these learnt low-dimensional binary codes for both classes and instances help in large-scale applications via efficient classification and retrieval (see Section 6.5). Note that, unlike attribute-based methods [151], we do *not* require additional meta-data, but learn binary codes by only using the standard classification task. This also circumvents the potential instabilities of pairwise optimization in instance binary code learning which often leads to poor class codes due to codebook collapse. At the end of LLC routine, we have learnt the instance codes, $B(\mathbf{P}\cdot F(x;\theta_F))$, and class codes, $B(\mathbf{C})$ to be used for downstream applications. Algorithm 1 presents LLC in full.

Overall, we present a simple yet scalable method to learn low-dimensional (exact) binary codes for both classes and instances which in turn could power multi-class classification with sub-linear costs (in terms of L) and efficient retrieval for large-scale applications. Using our method, we can consistently learn unique low-dimensional binary codes for all 1000 classes in ImageNet-1K using only 20-bits (which is twice the information-theoretic limit of $\lceil \log_2(1000) \rceil$). Next, we discuss the learnt codebook's intrinsic information about the classes and their structure.

6.4.2 Discovered Taxonomy and Visualizations

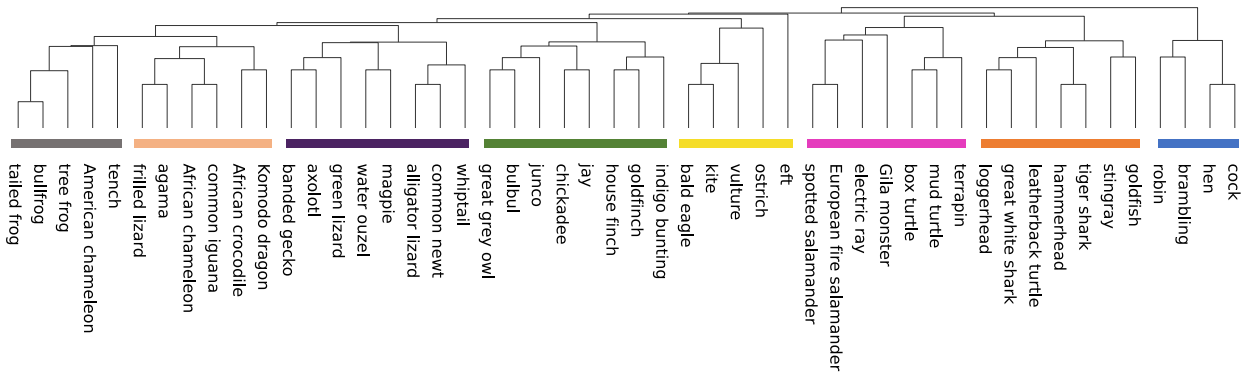


Figure 6.1: Discovered taxonomy over 50 classes of ImageNet-1K using the learnt 20-bit class codes. Related species are well clustered while pushing away unrelated ones. Figure 3 in Appendix D of Kusupati et al. [146] contains the codebook.

After learning the 20-bit binary codebook for 1000 classes of ImageNet-1K, we used the class representation from $B(\mathbf{C})$ of the first 50 classes to discover an intuitive taxonomy through agglomerative clustering [187]. Figure 6.1 shows the discovered hierarchy. This hierarchy effectively separates birds from amphibians; frogs and chickens are on extremes of the taxonomy and brings species with shared similarities closer (lizards & crocodiles; marine life). While the taxonomy is not perfect, the 20-bits do capture enough important information that can be used downstream.

Figure 6.2 shows the pair-wise inner-product heat maps for all the 1000 classes using 20-bits and 2048-dimensional real representation; the comparison reveals that 20-bits indeed highlights the same substructures

Algorithm 1 The LLC Method

Input: $\mathcal{D}$, F and B

Output: $\mathbf{C}$, $\mathbf{P}$ and θ_F

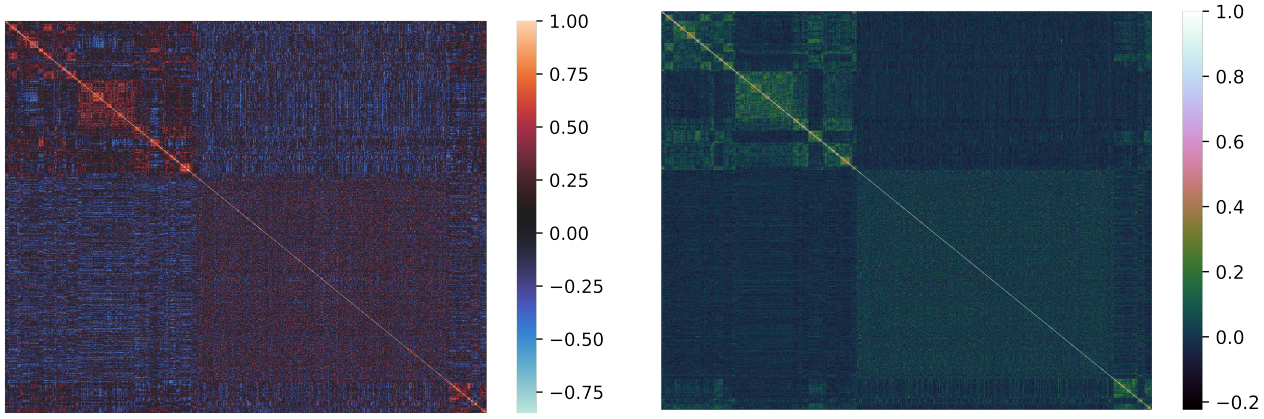
1: **Codebook Learning –** $B(\mathbf{C})$: Solve (6.2) using ERM and STE to get $\mathbf{C}$, $\mathbf{P}$ and θ_F -

$$\mathbf{C}, \mathbf{P}, \theta_F \leftarrow \underset{\mathbf{C},\mathbf{P},\theta_F}{\arg\min} \sum_{(x_i,y_i)\in\mathcal{D}} \mathcal{L}\left(B(\mathbf{C}) \cdot (\mathbf{P} \cdot F(x_i;\theta_F))\ ;\ y_i\right)\ .$$

2: **Instance Code Learning –** $B(\mathbf{P} \cdot F(x;\theta_f))$: Further optimize $\mathbf{P}$ and θ_F by solving (6.3) using ECOC framework and ERM by fixing $\mathbf{C}$ -

$$\theta_F, P \leftarrow \underset{\theta_F,P}{\arg\min} \sum_{(x_i,y_i)\in\mathcal{D}} \sum_{j=1}^{k} \text{BCE}\left(\sigma(\mathbf{P}_j \cdot F(x_i;\theta_F))\ ;\ (B(\mathbf{C}_{y_i,j}) + 1)/2\right)\ .$$

as the higher dimensional real-valued embeddings. Appendix D of Kusupati et al. [146] has a more detailed discussion about quantitatively evaluating the discovered hierarchy and more visualizations.



(a) 20-bit codes (b) 2048-d real representations

Figure 6.2: The pair-wise inner product heat maps of class representations a) learnt 20-bit codes & b) learnt 2048 dimensional real representations for the 1000 classes in ImageNet-1K. Similar sub structures are highlighted in both heatmaps and often correspond to local hierarchy present in the classes thus making a case that 20-bit codes distill enough information to capture hierarchy of the classes.

6.5 Applications

In this section, we discuss three applications of the learnt low-dimensional binary codes: 1) efficient multi-class classification (Section 6.5.1), 2) efficient retrieval (Section 6.5.2), and 3) out-of-the-box out-of-distribution (OOD) detection (Section 6.5.3). We also present ablation studies on codebook learning, feature separability and classification (Section 6.5.4).

6.5.1 Efficient Multi-class Classification

Recall that the proposed LLC algorithm outputs a) the learnt class codes (codebook), $B(\mathbf{C})$ and b) an *encoder* that produces instance codes, $B(\mathbf{P} \cdot F(\mathbf{x}; \theta_F))$ for x. We define a class codebook as a collection of L binary vectors, one for each class in the dataset, that can then be used for classification of a test instance x. We can use several "decoding" routines to classify an instance x, given its encoding and the learnt codebook. Below we discuss two decoding schemes that are diametrically opposite in terms of the inference cost. Also, note that the standard linear classification with real-valued representation and classifiers scale as $O(L)$ in terms of

computational complexity and model size.

Decoding Schemes

Exact Decoding (ED). Exact Decoding scheme expects the Hamming distance between the generated instance code, $B(\mathbf{P}F(\mathbf{x};\theta_F))$, and the ground truth class code, $B(\mathbf{C}_i)$ to be *exactly* 0. That is, we can hash the class codes in a table, and then ED requires only a $O(1)$ hash-table lookup for a given instance. Consequently, the inference time for ED is nearly *independent* of L. Naturally, the decoding scheme is highly stringent and would misclassify an instance if the instance binary code and the ground truth code do not match in even a single *bit*. Surprisingly, this highly efficient decoding scheme still provides non-trivial accuracy (see Table 6.1 and Section 6.5.1).

Minimum Hamming Decoding (MHD). Minimum Hamming Decoding is akin to the Maximum Dot Product used by standard linear classifiers. For an instance code, we evaluate the Hamming distance with all the L class codes and output the class with the least Hamming distance. Note that the Hamming distance over binary codes can be computed using XOR operations that are implemented significantly more efficiently than the floating-point operations [206]. Even though, technically, computational complexity and model size of MHD scales as $O(L)$ but the real-world implementations should be an order of magnitude faster than standard classifiers. In fact, for large number of classes L, the efficiency of MHD can be further improved by using approximate nearest neighbour search [50, 19, 179]. Appendix A of Kusupati et al. [146] has the mathematical presentation of the decoding schemes.

See Section 6.6 for more discussion on potential decoding schemes. Also see Section 6.5.4 for ablation studies about the two decoding schemes along with feature separability (linear vs Hamming).

Empirical Evaluation

ImageNet-1K [215] is a widely used image classification dataset with 1000 hierarchical classes. Our classification experiments use ResNet50 [98] and are trained using the $\sim$1.3M training images. Images were transformed & augmented with standard procedures [145, 266]. All the implementations were in PyTorch [196] and experimented on a machine with 4 NVIDIA Titan X (Pascal) GPUs.

Table 6.1: Classification performance on ImageNet-1K with ResNet50 using various class codebooks for training.

Codebook	Unique Codes	ED Accuracy (%)	MHD Accuracy (%)
Random 20-bits	1000	64.07	66.91
CCA 20-bits	813	55.17	57.03
SVD 20-bits	969	65.12	69.18
LLC 20-bits (Ours)	1000	**68.82**	**74.57**

Table 6.2: Classification accuracy on ImageNet-1K vs. bit length of the learnt class codebooks (§6.5.4).

LLC Length	Unique Codes	ED Accuracy (%)	MHD Accuracy (%)
15 bits	990	67.20	71.03
20 bits	1000	**68.82**	74.57
25 bits	1000	67.93	74.79
30 bits	1000	67.51	**75.13**

When applied to ImageNet-1K, the first phase of LLC, learnt a 20-bit codebook with 1000 unique class codes, i.e., every class has its own *distinct* binary code. We warm start the second phase of LLC by the learnt ResNet50 backbone along with the 20 dimensional projection layer. See Appendix C of Kusupati et al. [146] for the hyperparameter values and other training details.

A key feature of LLC is that it jointly learns both the class codebook as well as instance codes. Several existing techniques decouple this learning process where the codebook is constructed separately and is then used to train the instance codes [113, 47, 7, 72, 4, 263]. We evaluate the advantage of the *joint* learning approach of LLC by comparing its performance against three strong baselines: i) Random codebook of 20-bits, ii) 20-bit CCA codebook [4, 263, 280] & iii) 20-bit SVD codebook. Previous works [113, 47, 7] argued that random codebooks are competitive to the ones constructed using side-information. 20-bit CCA and SVD codebooks aim to capture the hierarchy that is amiss in the random codebook. The 20-bit SVD codebook is built using the SVD of 2048 dimensional linear classifiers (for each class) in the pre-trained ResNet50, and binarizing it. 20-bit CCA codebook is the binarized version of the transformed label embedding projected on to 20 components learnt using CCA between 2048 dimensional representations of 50K samples from the ImageNet train set and their one-hot label embeddings. Despite being able to capture the hierarchy information, both 20-bit CCA/SVD codebooks suffer from clashes reducing their overall effectiveness.

Next, using the baselines codebooks and the corresponding learnt instance codes, we compute class predictions for each test instance using the Exact Decoding (ED) & Minimum Hamming Decoding (MHD) schemes mentioned in the previous section. We evaluate all the methods using top-1 accuracy on the ImageNet-1K validation set. Baseline ResNet50 architecture represents the maximum accuracy we can hope to achieve using binarized instance+class codes. Note that this baseline classifier requires $O(L)$ computation over 16-bit real numbers, and achieves Top-1 accuracy of 77%.

Table 6.1 compares the accuracy of LLC (with 20-bit codebook) against baseline codebooks mentioned

above. Note that MHD with LLC codebook is 74.5% accurate, i.e., despite using only 20-dimensional *binary* representation it is only about 2.5% less accurate than standard ResNet50 that uses 2048 dimensional real-valued representation. Furthermore, we observe that compared to standard codebooks like SVD, our jointly learnt codebook is 5% more accurate.

Interestingly, Exact Decoding (ED) – which is $O(1)$ inference scheme – with LLC codebook is nearly as accurate as the SVD codebook with MHD scheme and is about 12% more accurate than the CCA codebook with ED scheme. Naturally, codebook length/dimensionality plays a critical role in classification accuracy; see Section 6.5.4 for a detailed ablative study on this aspect. Finally, the gains in efficiency should be even more compelling for problems with millions of classes [244].

6.5.2 Efficient Retrieval

The goal in retrieval is to find instances from a database that are most similar to a given *query*. Traditional retrieval approaches, use a *fixed metric* to retrieve "similar points", with data structures like LSH for efficient retrieval. Recent progress in Deep Supervised Hashing (DSH) [167] offer significantly more compelling solutions by learning the hashing function itself. That is, DSH aims to learn binary codes for each instance s.t. a pair of instances are embedded closely iff they belong to the same class, and then learns the hashing function end-to-end using a small train set.

As LLC also learns instance codes to reflect class membership, we can directly use our learnt encoder as a hashing function for given instances. For each query, the most relevant samples from the database are retrieved based on the minimum Hamming distance. Similar to the decoding schemes in classification, the retrieval can be optimized using approximate nearest neighbor search. Finally, the efficiency gains provided by using bits instead of real numbers should enable deployment of LLC based retrieval for efficient high recall portions of retrieval pipelines.

Empirical Evaluation

Following DSH literature, we evaluate hashing-based image retrieval on ImageNet-100, a benchmark dataset created by Cao et al. [34]. ImageNet-100 has 100 classes randomly sampled from ImageNet-1K. All the validation images of these classes are used as query images, all the training images ($\sim$ 1300 per class) of

Table 6.3: Efficient image retrieval on ImageNet-100 using AlexNet compared using MAP@1000 (Appendix B of Kusupati et al. [146]) across 16 – 64 bits.

Method	10 bits	16 bits	32 bits	48 bits	64 bits
HashNet [34]	0.1995	0.2815	0.4300	0.5270	0.5124
Greedy Hash [234]	0.2860	0.4247	0.5412	0.5720	0.5895
LLC (Ours)	**0.3086**	**0.4305**	**0.5565**	**0.5749**	**0.6000**

Table 6.4: Comparison of LLC based retrieval vs real-valued representations with ResNet50 on ImageNet-100 using MAP@1000.

Representation	8 dims	10 dims	64 dims
LLC (1 bit)	-	0.6458	0.6773
Real (16 bits)	0.5041	0.6657	0.7794

these 100 classes are used as database images. Finally, 130 samples per class from the database are used as the training set for learning binary codes or hashing functions.

We compare against HashNet [34] and Greedy Hash [234] for image retrieval using learnt instance codes. HashNet learns the bit representations of instances using a pairwise optimization with positive and negative instance pairs. HashNet is a representative baseline for an alternative way of learning binary instance codes compared to LLC. On the other hand, Greedy Hash learns only the instance codes using straight-through-estimator via the classification task. Note that LLC learns both class codes as well as instance codes differentiating it from Greedy Hash style methods. Learnt instance codes are a byproduct of efficient classification as opposed to baselines that optimize for them.

We use the Mean Average Precision (MAP@1000) metric for evaluation. The MAP@1000 calculation code of HashNet [34] is erroneous and has propagated to several papers in the literature. We use the corrected version, hence the accuracy numbers are different from the original paper. Please see Appendix B of Kusupati et al. [146] for the corrected version, the changes required along with an example and a brief discussion. We used the publicly available pre-trained HashNet models [35] and Greedy Hash [234] code to recompute the MAP@1000.

Following HashNet [34], we use AlexNet [137] as the backbone and warm-start it with a pre-trained model on ImageNet-1K. We add a projection layer to the backbone and learn the instance and class codes. We also report retrieval numbers with ResNet50 [98] and compare LLC based retrieval numbers to learnt real-valued representations. Please see Appendix C of Kusupati et al. [146] for the training details and hyperparameters of efficient retrieval pipelines.

Table 6.3 shows the performance (evaluated using MAP@1000) for HashNet, Greedy Hash, and LLC across various code lengths. LLC outperforms HashNet across all code lengths (16 – 64) by at least 4.79% on MAP@1000. LLC is also better than Greedy Hash across all the bit lengths. LLC also outperforms 16-bit HashNet by 2% & 15% using only 10 & 16 bits respectively. Finally, 32-bit LLC comfortably outperforms

both 48 & 64-bit HashNet showcasing the effectiveness of our learnt tight bit codes. Note that LLC, learning both instance and class codes, is effective in retrieval even though it was designed for classification.

We repeat the retrieval experiments with ResNet50. Table 6.4 shows the MAP@1000 for LLC with 10 and 64 bits along with the same dimensional real-valued representations. The 10-bit LLC is only 2% lower than 10 dimensional real-valued representation even though theoretically, the cost associated with 10-bit LLC based retrieval is about $256\times$ less than 10 dimensional real representations.

The 64 bit and 10 bit LLC outperforms 10 and 8 dimensional real-valued representations respectively at a much cheaper retrieval cost, at least by an order of magnitude. More discussion about the use of binary codes for retrieval at a large scale can be found in Section 6.6. Finally, 10-bit LLC with ResNet50 outperforms the best performing AlexNet based models for the same task, suggesting ResNet50 is a more appropriate architecture for benchmarking DSH literature.

6.5.3 Out-of-Distribution (OOD) Detection

For a multi-class classifier, detecting an OOD sample is very important for robustness [104] and sequential learning [249]. Multi-class classifiers are augmented with OOD detection capability by setting a threshold on heuristics like maximum logit which is tuned using a validation set.

We focus on the scenario where the ratio of in-distribution to out-of-distribution samples in the validation set is not representative of the deployment. This throws off the methods that try to maximize metrics, F1, using a validation set. Our learnt class codebook from LLC comes with over-provisioning (for ease of optimization) resulting in unassigned codes. These unassigned codes can be treated as OOD out-of-the-box with no tuning whatsoever. That is, we classify an instance as OOD if its instance code does not match *exactly* with the code of a class in our learnt codebook.

Appendix E of Kusupati et al. [146] discusses the OOD detection experiments on ImageNet-750 [249] & MIT Places [281]. At a high level, LLC based out-of-the-box OOD detection (with a learnt 20-bit codebook on ImageNet-1K) achieves nearly the same OOD detection accuracy as a baseline [104] that tries to maximize F1 using a validation set. We would like to stress that while such a method needs ≈ 3000 points in the validation set, our method requires *no* samples, which is critical in several practical settings.

6.5.4 Ablation Studies

Classification Accuracy vs Number of Bits. Table 6.2 shows the trade-off in classification accuracy with the variation in the length of the learnt codebook for ImageNet-1K. LLC learns a 15-bit codebook with only 990 unique codes leading to a loss of accuracy due to code collapse in both ED and MHD schemes (1.62% & 3.54% compared to 20-bit codebook respectively). An interesting observation is that the ED accuracy gradually goes down after 20-bits while the MHD accuracy keeps on increasing. The phenomenon of increasing accuracy with MHD is probably due to the increase in the capacity of both instance and class codes. However, the decrease in ED accuracy after 20-bits can be explained through the hardness in exactly predicting every bit in the instance code to match the ground truth class code. Our classification model with 20-bits on average gets 19.2 bits correct but the model with 30-bits only gets 28.5 bits right. This increase in uncertainty coupled with the stringent ED scheme leads to a slight dip in accuracy as the code length increases. However, this also provides us with a path for more accurate decoding schemes while being efficient as discussed in Section 6.6.

Classification Accuracy vs Faster Codebook Learning. Codebook learning phase of LLC is expensive, this motivated us to speed up codebook learning at a minimal loss in accuracy. One way is to warm-start the codebook using the ones built with SVD/CCA (see Section 6.5.1). While these codebooks suffer from code collapse, with further training, they start to learn 1000 unique codes quickly. Using these final codebooks gets to a comparable (1% drop) accuracy as the 20-bit learnt LLC codebook but at a relatively cheaper training. Another option is to use only a portion of the data and a much smaller network to learn the codebook. We sampled 50K training images and use a MobileNetV1 [111] (which has about $6\times$ less parameters and compute than ResNet50) to learn a 20-bit codebook which gets to ED and MHD accuracy of 66.62% & 72.55% which is only 2% lower than the end-to-end learnt codebook but at a fraction of the training cost (3 hrs vs 2 days).

Linear vs Hamming Separability. Fitting a deep neural network to the learnt codebook for classification results in warping of the feature space considerably. The final classification space is a hypercube with the vertices being apart by Hamming distance of 1. To verify linear separability, we take the learnt, frozen ResNet50 trained for the 20-bit classification problem and fit a linear classifier on top of the 2048 dimensional

features. Linear classifier quickly reaches a top-1 accuracy of 75.51%.

The opposite does not seem to be true. We extract and freeze the backbone of a pre-trained ResNet50 and train a projection layer to fit the 20-bit learnt codebook. This gets to top-1 accuracy of only about 21% with the ED scheme. However, we also observed that unfreezing and finetuning the last 3 layers of the backbone recovers the top-1 ED accuracy to roughly 68%.

These experiments show 1) Hamming separability inherently enables linear separability, 2) Linear separability does not imply Hamming separability & 3) with enough overparameterization, linearly separable space can be warped to support Hamming separability. Hamming separability automatically provides linear separability with increased accuracy of $\sim 1\%$ over the MHD scheme which allows for an option for using a more powerful yet simple classifier, in case of accuracy requirements.

Independent vs Nested Codebook Learning. Consider a scenario with varied computational budgets for classification. We could either train independent k-bit models (eg., $k = 20, 25, 30$) and use them according to the budget, or we could learn a single nested codebook-based model that can be readily adapted to any of these settings. While training a codebook of larger bit length like $k = 30$, we can ensure that the first m-bits, $m < k$, also form a codebook at minimal additional cost. We were able to stably train a 30-bit codebook and also extract 20, 25-bit codebooks from it all of which are as accurate as independently trained codebooks. These nested codebooks have the potential to be used based on the computational resource availability for efficient classification without having to retrain. All this capabilities are enabled by Matryoshka Representation Learning [147] presented in Chapter 2.

6.6 Discussion and Conclusions

We designed LLC to learn low-dimensional binary codes for instances as well as classes, and utilized them in applications like efficient classification, retrieval, OOD detection. A key finding is that combining class code learning with ECOC framework to learn instance code leads to a stable training system that can accurately capture the semantics of the class data despite just 20-dimensional code. Traditionally, methods like HashNet, KLSH [141] attempt to learn hashing function using pairwise loss functions by embedding instances such that points from the same class are embedded closely and points from different classes are

far. But such formulations are hard to optimize, due to the risk of embedding collapse. We observe that LLC by using instance-wise formulation can train stably with significantly higher performance. Another fascinating observation is that while architectures like ResNet50 have large intermediate (2048 dimensional real) representations, they can be compressed to just 20 bits without significant loss in accuracy! Even though quantization [206] literature demonstrates strong compression of representations, we believe such stark compression has been elusive so far and is worth further exploration from the efficient inference viewpoint.

Limitations. Our visualization (see Figure 4 in Appendix D of Kusupati et al. [146]) indicates that each bit does not correspond to some easily interpretable attributes, unlike DBC [205]. We believe incorporating priors with weak supervision as well as cross-modal learning could help LLC get past this limitation.

ED and MHD schemes are on two ends of the computation vs accuracy spectrum and do not transition smoothly. Designing decoding schemes that can compromise between these two extreme decoding schemes might be able to address this limitation.

Strong encoders are needed to warp the feature space to ensure Hamming separability. For example, using a 20-bit learnt codebook with ED scheme, ResNet50 gets to 68.8% top-1 accuracy whereas MobileNetV1 can only reach 53.23%. This also ties into the argument that classification is a trade-off between encoder and decoder costs. Making decoders efficient and cheap, puts the burden on encoding the information in the right way and higher expressivity often helps in that cause.

Future Work. There are several exciting directions that we would like to explore. In principle, LLC can easily incorporate side-information when needed with simple additional losses during training. The additional regularization losses can also help in incorporating natural constraints on the codebook [47, 7] or can enable attribute-based class codes for interpretablity [75, 76, 151] making them exciting directions to explore. LLC algorithm can also be used to encode instances of multiple modalities like audio, visual, language to the same learnt low-dimensional binary space. This might help in effective cross-modal supervision along with retrieval among various other applications.

Potential in Large-Scale Applications. While our focus was on designing low-dimensional accurate binary codes, several studies [206, 116] have shown that efficiency afforded by bit-wise computation over

floating-point computation can lead to almost an order of magnitude speed-up. Furthermore, as the number of classes increases, the learning of class codebooks helps in training representations in sublinear costs [45] along with sublinear inference (in L). We expect LLC algorithm to have its efficiency benefits outweigh the accuracy drop for large multi-class/multi-label problems, like objection recognition using ImageNet-22K [55], document tagging [244, 200] and instance classification [264]. The efficiency aspect of the binary codes has not been fully explored in this paper as the main computational bottleneck for ImageNet-1K classification is the deep neural network featurizer.

Lastly, LLC based efficient retrieval can be used for the initial high-recall shortlisting of a search pipeline, which is followed by high precision models operating on more expressive yet expensive embeddings. We leave practical demonstration of such a system at web-scale for future work.

Chapter 7

Discussion, Conclusion and Future Work

I believe that the works presented in this thesis are a right step *towards adaptive intelligence* in web-scale AI/ML systems. A testament to this is the wide-spread adoption of these works in real-world deployments at the internet scale across both industry and open-source.

Matryoshka representation learning (MRL) [147] presented in Chapter 2 is now a default design choice for quite a few dense embedding applications. Most notable are the state-of-the-art text embedding models ada-v3 by OpenAI [194] and gecko by Google [154] that help power a wide range of retrieval and retrieval augmented generation (RAG) [159] across millions of users. Google also made MRL a default for many of its embedding models powering web-scale image products – serving over a billion users daily. Pinterest has used MRL as part of their LinkSage [199] recommendation system. Further, the simplicity and scalability of the idea attracted a lot of interest from the open-source community through Hugging Face [74] where several of the most popular embedding models [3, 2] are powered by MRL. This further led to vector database and retrieval communities implement naive version of adaptive retrieval at scale [236]. Adding to this, improved training dynamics due to MRL, led to state of the art cross modal retreival models [247]. Overall, MRL, either through pretraining or fine-tuning, has become a drop in design choice for any dense vector representation based models and systems – owing to inherent adaptivity.

In a similar vein other works have been used extensively or showed promise at scale. AdANNS [209], presented in Chapter 3, built to levergae MRL for web-scale search systems has show to scale to tens of billions of documents while enabling adaptive inference on a fixed ANNS index for the first time. MatFormer [63],

presented in Chapter 4, is tested for web-scale applications as the next generation Transformer architecture for seamless compute adaptivity and also has introduced adaptive query processing for web-scale search, for the first time. Further, STR [145] presented in Chapter 5 has been used extensively for efficient models through pruning at companies like Apple while also helping the community rethink fundamentals of sparsity. Finally, LLC [146] presented in Chapter 6 when combined with MRL was able to encode 1 billion images with 8 bytes per image – totaling to 8 GB, at least $4\times$ cheaper than exisiting techniques – while retraining the recall during retrieval. Owing to the nested structure and binary precision, the representation acts as its own search index and can enable near $O(1)$ search on a regular smartphone, thus paving the way for web-scale offline search. This further helps us in "Indexing the World" in a differentiable fashion, allowing for data and interactions to control the knowledge organization for an accurate and efficient knowledge discovery.

I envision a future where AI systems cater to every user accurately, reliably and equitably based on their needs in real-time. To realize this goal, I will explore the following directions in the near future.

End-to-end elastic search. Bringing together everything I have developed towards elastic modelling, representations and ANNS will result in a truly elastic end-to-end learned search system that maintains accuracy for *head* tasks at fraction of the cost while being able to spend lot more resources to cater the rare *tail* queries to not leave any data or user behind.

Indexing the world. Rethinking search through end-to-end representation learning and compression opens up a new and on-the-fly way to index the entire world, not just the web. Imagine a robot that is moving around and perceiving its surroundings, at the moment it can rarely remember everything it saw, heard or felt. Enabling efficient representation that doubles as index of the perceived states would help any embodied or intelligent agent make a more informed decision like what a human or even a crow would. This also helps in improving privacy by enabling on-device indexing and search without compromising on accuracy for smart devices [117]. Beyond perception, this representation learning paradigm assists in any setup that requires accurate and fast search on all the candidates like in drug discovery [49] or protein structure generation [243]. I wish to expand on potential applications of efficient large-scale search and storage across natural sciences for a more grounded use of generative foundation models.

Contextual foundation models. Hallucinations are the Achilles' heel of modern generative models, especially for tail tasks. While post-hoc retrieval augmentation [159, 208] can fix some of the issues and make the generations more grounded and diverse, I look forward to building contextual foundation models that are explicitly designed and optimized to leverage retrieval of relevant contextual data and external memory banks as core components in their inference [25].

Continually learning intelligent systems. While human learning is never-ending, the machine equivalent, continual learning has hit a road block owing to the issues in evaluation [251]. Loosely drawing parallels to human brain or to a great extent emulating the modern-day computer architecture, elastic models can act as hierarchical information packing and learning entities. I would like to revisit continual learning, through the lens of elastic and contextual modeling, in real-world to capture trends across temporal scales while discovering new things along the way [252] to eventually build a world model alongside fast local models across time-scales [131].

In sum, my research focused on designing fundamental ML algorithms with strong empirical performance and real-world deployability geared towards enabling *efficient, elastic and contextual – **adaptive** – intelligence* that can bring the systems ever so close to the efficiency of the human brain [185].

Bibliography

[1] M. Abadi, P. Barham, J. Chen, Z. Chen, A. Davis, J. Dean, M. Devin, S. Ghemawat, G. Irving, M. Isard, et al. Tensorflow: A system for large-scale machine learning. In *12th {USENIX} Symposium on Operating Systems Design and Implementation ({OSDI} 16)*, pages 265–283, 2016.

[2] M. AI. 64 bytes per embedding, yee-haw. 2024. URL https://www.mixedbread.ai/blog/binary-mrl.

[3] N. AI. Unboxing nomic embed v1.5: Resizable production embeddings with matryoshka representation learning. 2024. URL https://blog.nomic.ai/posts/nomic-embed-matryoshka.

[4] Z. Akata, F. Perronnin, Z. Harchaoui, and C. Schmid. Label-embedding for image classification. *IEEE transactions on pattern analysis and machine intelligence*, 38(7):1425–1438, 2015.

[5] Z. Akata, S. Reed, D. Walter, H. Lee, and B. Schiele. Evaluation of output embeddings for fine-grained image classification. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 2927–2936, 2015.

[6] K. Alizadeh, A. Farhadi, and M. Rastegari. Butterfly transform: An efficient fft based neural architecture design. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2020.

[7] E. L. Allwein, R. E. Schapire, and Y. Singer. Reducing multiclass to binary: A unifying approach for margin classifiers. *Journal of machine learning research*, 1(Dec):113–141, 2000.

[8] D. Anguita, A. Ghio, L. Oneto, X. Parra, and J. L. Reyes-Ortiz. Human activity recognition on smartphones using a multiclass hardware-friendly support vector machine. In *International Workshop*

on Ambient Assisted Living, pages 216–223. Springer, 2012. URL https://archive.ics.uci.edu/ml/datasets/human+activity+recognition+using+smartphones.

[9] R. Anil, A. M. Dai, O. Firat, M. Johnson, D. Lepikhin, A. Passos, S. Shakeri, E. Taropa, P. Bailey, Z. Chen, et al. Palm 2 technical report. *arXiv preprint arXiv:2305.10403*, 2023.

[10] M. Ashby, C. Baaij, P. Baldwin, M. Bastiaan, O. Bunting, A. Cairncross, C. Chalmers, L. Corrigan, S. Davis, N. van Doorn, et al. Exploiting unstructured sparsity on next-generation datacenter hardware.

[11] M. Aumüller, E. Bernhardsson, and A. Faithfull. Ann-benchmarks: A benchmarking tool for approximate nearest neighbor algorithms. *Information Systems*, 87:101374, 2020.

[12] K. Azarian, Y. Bhalgat, J. Lee, and T. Blankevoort. Learned threshold pruning. *arXiv preprint arXiv:2003.00075*, 2020.

[13] M. Á. Bautista Martín et al. *Learning error-correcting representations for multi-class problems*. PhD thesis, Universitat de Barcelona, 2016.

[14] A. Beck and M. Teboulle. A fast iterative shrinkage-thresholding algorithm for linear inverse problems. *SIAM journal on imaging sciences*, 2(1):183–202, 2009.

[15] G. Bellec, D. Kappel, W. Maass, and R. Legenstein. Deep rewiring: Training very sparse deep networks. In *International Conference on Learning Representations*, 2018.

[16] S. Bengio, J. Weston, and D. Grangier. Label embedding trees for large multi-class tasks. *Advances in Neural Information Processing Systems*, 23, 2010.

[17] Y. Bengio. Deep learning of representations for unsupervised and transfer learning. In *Proceedings of ICML workshop on unsupervised and transfer learning*, pages 17–36. JMLR Workshop and Conference Proceedings, 2012.

[18] Y. Bengio, N. Léonard, and A. Courville. Estimating or propagating gradients through stochastic neurons for conditional computation. *arXiv preprint arXiv:1308.3432*, 2013.

[19] J. L. Bentley. K-d trees for semidynamic point sets. In *Proceedings of the sixth annual symposium on Computational geometry*, pages 187–197, 1990.

[20] E. Bernhardsson. *Annoy: Approximate Nearest Neighbors in C++/Python*, 2018. URL https://pypi.org/project/annoy/. Python package version 1.13.0.

[21] L. Beyer, P. Izmailov, A. Kolesnikov, M. Caron, S. Kornblith, X. Zhai, M. Minderer, M. Tschannen, I. Alabdulmohsin, and F. Pavetic. Flexivit: One model for all patch sizes. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 14496–14506, 2023.

[22] A. Beygelzimer, S. Kakade, and J. Langford. Cover trees for nearest neighbor. In *Proceedings of the 23rd international conference on Machine learning*, pages 97–104, 2006.

[23] K. Bhatia, H. Jain, P. Kar, M. Varma, and P. Jain. Sparse local embeddings for extreme multi-label classification. In *NIPS*, volume 29, pages 730–738, 2015.

[24] R. Bommasani, D. A. Hudson, E. Adeli, R. Altman, S. Arora, S. von Arx, M. S. Bernstein, J. Bohg, A. Bosselut, E. Brunskill, et al. On the opportunities and risks of foundation models. *arXiv preprint arXiv:2108.07258*, 2021.

[25] S. Borgeaud, A. Mensch, J. Hoffmann, T. Cai, E. Rutherford, K. Millican, G. B. Van Den Driessche, J.-B. Lespiau, B. Damoc, A. Clark, et al. Improving language models by retrieving from trillions of tokens. In *International conference on machine learning*. PMLR, 2022.

[26] S. Brin and L. Page. The anatomy of a large-scale hypertextual web search engine. *Computer networks and ISDN systems*, 30(1-7):107–117, 1998.

[27] T. Brown, B. Mann, N. Ryder, M. Subbiah, J. D. Kaplan, P. Dhariwal, A. Neelakantan, P. Shyam, G. Sastry, A. Askell, et al. Language models are few-shot learners. *Advances in neural information processing systems*, 33:1877–1901, 2020.

[28] C. Buciluǎ, R. Caruana, and A. Niculescu-Mizil. Model compression. In *Proceedings of the 12th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 535–541, 2006.

[29] D. Cai. A revisit of hashing algorithms for approximate nearest neighbor search. *IEEE Transactions on Knowledge and Data Engineering*, 33(6):2337–2348, 2021. doi: 10.1109/TKDE.2019.2953897.

[30] H. Cai, C. Gan, T. Wang, Z. Zhang, and S. Han. Once-for-all: Train one network and specialize it for efficient deployment. *arXiv preprint arXiv:1908.09791*, 2019.

[31] H. Cai, C. Gan, J. Lin, and S. Han. Network augmentation for tiny deep learning. *arXiv preprint arXiv:2110.08890*, 2021.

[32] R. Cai, S. Muralidharan, G. Heinrich, H. Yin, Z. Wang, J. Kautz, and P. Molchanov. Flextron: Many-in-one flexible large language model. In *International Conference on Machine Learning*. PMLR, 2024.

[33] E. Candes, T. Tao, et al. The dantzig selector: Statistical estimation when p is much larger than n. *The annals of Statistics*, 35(6):2313–2351, 2007.

[34] Z. Cao, M. Long, J. Wang, and P. S. Yu. Hashnet: Deep learning to hash by continuation. In *Proceedings of the IEEE international conference on computer vision*, pages 5608–5617, 2017.

[35] Z. Cao, M. Long, J. Wang, and P. S. Yu. Hashnet: Deep learning to hash by continuation, 2017. URL https://github.com/thuml/HashNet.

[36] M. A. Carreira-Perpinán and R. Raziperchikolaei. Hashing with binary autoencoders. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 557–566, 2015.

[37] W.-C. Chang, F. X. Yu, Y.-W. Chang, Y. Yang, and S. Kumar. Pre-training tasks for embedding-based large-scale retrieval. *arXiv preprint arXiv:2002.03932*, 2020.

[38] W.-C. Chang, D. Jiang, H.-F. Yu, C. H. Teo, J. Zhang, K. Zhong, K. Kolluri, Q. Hu, N. Shandilya, V. Ievgrafov, et al. Extreme multi-label learning for semantic matching in product search. In *Proceedings of the 27th ACM SIGKDD Conference on Knowledge Discovery & Data Mining*, pages 2643–2651, 2021.

[39] A. Chavan, Z. Shen, Z. Liu, Z. Liu, K.-T. Cheng, and E. P. Xing. Vision transformer slimming: Multi-dimension searching in continuous optimization space. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 4931–4941, 2022.

[40] C. Chen, S. Borgeaud, G. Irving, J.-B. Lespiau, L. Sifre, and J. Jumper. Accelerating large language model decoding with speculative sampling. *arXiv preprint arXiv:2302.01318*, 2023.

[41] T. Chen, S. Kornblith, M. Norouzi, and G. Hinton. A simple framework for contrastive learning of visual representations. In *International conference on machine learning*, pages 1597–1607. PMLR, 2020.

[42] T. Chen, L. Li, and Y. Sun. Differentiable product quantization for end-to-end embedding compression. In *International Conference on Machine Learning*, pages 1617–1626. PMLR, 2020.

[43] W. Chen, Y. Liu, W. Wang, E. M. Bakker, T. Georgiou, P. Fieguth, L. Liu, and M. S. Lew. Deep learning for instance retrieval: A survey. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2022.

[44] A. Choromanska, K. Choromanski, M. Bojarski, T. Jebara, S. Kumar, and Y. LeCun. Binary embeddings with structured hashed projections. In *International Conference on Machine Learning*, pages 344–353. PMLR, 2016.

[45] M. Cissé, N. Usunier, T. Artieres, and P. Gallinari. Robust bloom filters for large multilabel classification tasks. In *Advances in Neural Information Processing Systems 26*, pages 1851–1859, 2013.

[46] K. L. Clarkson. An algorithm for approximate closest-point queries. In *Proceedings of the tenth annual symposium on Computational geometry*, pages 160–164, 1994.

[47] K. Crammer and Y. Singer. On the algorithmic implementation of multiclass kernel-based vector machines. *Journal of machine learning research*, 2(Dec):265–292, 2001.

[48] K. Dahiya, D. Saini, A. Mittal, A. Shaw, K. Dave, A. Soni, H. Jain, S. Agarwal, and M. Varma. Deepxml: A deep extreme multi-label learning framework applied to short text documents. In *Proceedings of the 14th ACM International Conference on Web Search and Data Mining*, pages 31–39, 2021.

[49] S. Dara, S. Dhamercherla, S. S. Jadav, C. M. Babu, and M. J. Ahsan. Machine learning in drug discovery: a review. *Artificial Intelligence Review*, 55(3):1947–1999, 2022.

[50] M. Datar, N. Immorlica, P. Indyk, and V. S. Mirrokni. Locality-sensitive hashing scheme based on p-stable distributions. In *Proceedings of the twentieth annual symposium on Computational geometry*, pages 253–262, 2004.

[51] J. Dean. Challenges in building large-scale information retrieval systems. In *Keynote of the 2nd ACM International Conference on Web Search and Data Mining (WSDM)*, volume 10, 2009.

[52] M. Dehghani, A. Gritsenko, A. Arnab, M. Minderer, and Y. Tay. Scenic: A jax library for computer vision research and beyond. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 21393–21398, 2022.

[53] M. Dehghani, J. Djolonga, B. Mustafa, P. Padlewski, J. Heek, J. Gilmer, A. P. Steiner, M. Caron, R. Geirhos, I. Alabdulmohsin, et al. Scaling vision transformers to 22 billion parameters. In *International Conference on Machine Learning*, pages 7480–7512. PMLR, 2023.

[54] M. Deitke, R. Liu, M. Wallingford, H. Ngo, O. Michel, A. Kusupati, A. Fan, et al. Objaverse-XL: A universe of 10m+ 3d objects. *Advances in Neural Information Processing Systems, Datasets and Benchmarks*, 2023.

[55] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei. Imagenet: A large-scale hierarchical image database. In *2009 IEEE conference on computer vision and pattern recognition*, pages 248–255. Ieee, 2009.

[56] J. Deng, A. C. Berg, and L. Fei-Fei. Hierarchical semantic indexing for large scale image retrieval. In *CVPR 2011*, pages 785–792. IEEE, 2011.

[57] D. K. Dennis, Y. Gaurkar, S. Gopinath, C. Gupta, M. Jain, A. Kumar, A. Kusupati, C. Lovett, S. G. Patil, and H. V. Simhadri. EdgeML: Machine Learning for resource-constrained edge devices. URL https://github.com/Microsoft/EdgeML.

[58] K. Desai and J. Johnson. Virtex: Learning visual representations from textual annotations. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 11162–11173, 2021.

[59] T. Dettmers and L. Zettlemoyer. Sparse networks from scratch: Faster training without losing performance. *arXiv preprint arXiv:1907.04840*, 2019.

[60] T. Dettmers and L. Zettlemoyer. The case for 4-bit precision: k-bit inference scaling laws. In *International Conference on Machine Learning*, pages 7750–7774. PMLR, 2023.

[61] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*, 2018.

[62] Devvrit, S. Kudugunta, A. Kusupati, T. Dettmers, K. Chen, I. Dhillon, Y. Tsvetkov, H. Hajishirzi, S. Kakade, A. Farhadi, P. Jain, et al. Matformer: Nested transformer for elastic inference. *arXiv preprint arXiv:2310.07707*, 2023.

[63] Devvrit, S. Kudugunta, A. Kusupati, T. Dettmers, K. Chen, I. Dhillon, Y. Tsvetkov, H. Hannaneh, S. Kakade, A. Farhadi, and P. Jain. Matformer: Nested transformer for elastic inference. *arXiv preprint arxiv:2310.07707*, 2023.

[64] T. G. Dietterich and G. Bakiri. Solving multiclass learning problems via error-correcting output codes. *Journal of artificial intelligence research*, 2:263–286, 1994.

[65] S. K. Divvala, A. Farhadi, and C. Guestrin. Learning everything about anything: Webly-supervised visual concept learning. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 3270–3277, 2014.

[66] D. L. Donoho. De-noising by soft-thresholding. *IEEE transactions on information theory*, 41(3): 613–627, 1995.

[67] A. Dosovitskiy, L. Beyer, A. Kolesnikov, D. Weissenborn, X. Zhai, T. Unterthiner, M. Dehghani, M. Minderer, G. Heigold, S. Gelly, et al. An image is worth 16x16 words: Transformers for image recognition at scale. *arXiv preprint arXiv:2010.11929*, 2020.

[68] N. Du, Y. Huang, A. M. Dai, S. Tong, D. Lepikhin, Y. Xu, M. Krikun, Y. Zhou, A. W. Yu, O. Firat, B. Zoph, L. Fedus, M. Bosma, Z. Zhou, T. Wang, Y. E. Wang, K. Webster, M. Pellat, K. Robinson,

K. Meier-Hellstern, T. Duke, L. Dixon, K. Zhang, Q. V. Le, Y. Wu, Z. Chen, and C. Cui. Glam: Efficient scaling of language models with mixture-of-experts, 2022.

[69] E. Elsen, M. Dukhan, T. Gale, and K. Simonyan. Fast sparse convnets. *arXiv preprint arXiv:1911.09723*, 2019.

[70] J. J. Engelsma, A. K. Jain, and V. N. Boddeti. Hers: Homomorphically encrypted representation search. *IEEE Transactions on Biometrics, Behavior, and Identity Science*, 4(3):349–360, 2022.

[71] L. Engstrom, A. Ilyas, H. Salman, S. Santurkar, and D. Tsipras. Robustness (python library), 2019. URL https://github.com/MadryLab/robustness.

[72] S. Escalera, O. Pujol, and P. Radeva. Error-correcting ouput codes library. *The Journal of Machine Learning Research*, 11:661–664, 2010.

[73] U. Evci, T. Gale, J. Menick, P. S. Castro, and E. Elsen. Rigging the lottery: Making all tickets winners. In *International Conference on Machine Learning*, 2020.

[74] H. Face. Introduction to matryoshka embedding models. 2024. URL https://huggingface.co/blog/matryoshka.

[75] A. Farhadi, I. Endres, D. Hoiem, and D. Forsyth. Describing objects by their attributes. In *2009 IEEE Conference on Computer Vision and Pattern Recognition*, pages 1778–1785. IEEE, 2009.

[76] V. Ferrari and A. Zisserman. Learning visual attributes. *Advances in neural information processing systems*, 20:433–440, 2007.

[77] J. Frankle and M. Carbin. The lottery ticket hypothesis: Finding sparse, trainable neural networks. In *International Conference on Learning Representations*, 2019.

[78] J. H. Friedman, J. L. Bentley, and R. A. Finkel. An algorithm for finding best matches in logarithmic expected time. *ACM Transactions on Mathematical Software (TOMS)*, 3(3):209–226, 1977.

[79] T. Gale, E. Elsen, and S. Hooker. The state of sparsity in deep neural networks. *arXiv preprint arXiv:1902.09574*, 2019.

[80] T. Ge, K. He, Q. Ke, and J. Sun. Optimized product quantization for approximate nearest neighbor search. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 2946–2953, 2013.

[81] A. Gholami, S. Kim, Z. Dong, Z. Yao, M. W. Mahoney, and K. Keutzer. A survey of quantization methods for efficient neural network inference. *arXiv preprint arXiv:2103.13630*, 2021.

[82] G. Golub and W. Kahan. Calculating the singular values and pseudo-inverse of a matrix. *Journal of the Society for Industrial and Applied Mathematics, Series B: Numerical Analysis*, 2(2):205–224, 1965.

[83] S. Gong, V. N. Boddeti, and A. K. Jain. On the intrinsic dimensionality of image representations. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 3987–3996, 2019.

[84] Y. Gong, S. Lazebnik, A. Gordo, and F. Perronnin. Iterative quantization: A procrustean approach to learning binary codes for large-scale image retrieval. *IEEE transactions on pattern analysis and machine intelligence*, 35(12):2916–2929, 2012.

[85] A. Gordon, E. Eban, O. Nachum, B. Chen, H. Wu, T.-J. Yang, and E. Choi. Morphnet: Fast & simple resource-constrained structure learning of deep networks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 1586–1595, 2018.

[86] R. Gray. Vector quantization. *IEEE Assp Magazine*, 1(2):4–29, 1984.

[87] M. Grimaldi, L. Mocerino, A. Cipolletta, and A. Calimera. Dynamic convnets on tiny devices via nested sparsity. *IEEE Internet of Things Journal*, 10(6):5073–5082, 2022.

[88] C. Guo, A. Mousavi, X. Wu, D. Holtmann-Rice, S. Kale, S. J. Reddi, and S. Kumar. Breaking the glass ceiling for embedding-based classifiers for large output spaces. In *NeurIPS*, 2019.

[89] R. Guo, P. Sun, E. Lindgren, Q. Geng, D. Simcha, F. Chern, and S. Kumar. Accelerating large-scale inference with anisotropic vector quantization. In *International Conference on Machine Learning*, pages 3887–3896. PMLR, 2020.

[90] Y. Guo, A. Yao, and Y. Chen. Dynamic network surgery for efficient dnns. In *Advances In Neural Information Processing Systems*, pages 1379–1387, 2016.

[91] N. Gupta, P. H. Chen, H.-F. Yu, C.-J. Hsieh, and I. S. Dhillon. End-to-end learning to index and search in large output spaces. *arXiv preprint arXiv:2210.08410*, 2022.

[92] M. Gutmann and A. Hyvärinen. Noise-contrastive estimation: A new estimation principle for unnormalized statistical models. In *Proceedings of the thirteenth international conference on artificial intelligence and statistics*, pages 297–304. JMLR Workshop and Conference Proceedings, 2010.

[93] S. Han, J. Pool, J. Tran, and W. Dally. Learning both weights and connections for efficient neural network. In *Advances in neural information processing systems*, pages 1135–1143, 2015.

[94] M. G. Harris and C. D. Giachritsis. Coarse-grained information dominates fine-grained information in judgments of time-to-contact from retinal flow. *Vision research*, 40(6):601–611, 2000.

[95] B. Hassibi and D. G. Stork. Second order derivatives for network pruning: Optimal brain surgeon. In *Advances in neural information processing systems*, pages 164–171, 1993.

[96] T. Hastie, R. Tibshirani, and J. Friedman. *The elements of statistical learning: data mining, inference, and prediction*. Springer Science & Business Media, 2009.

[97] J. He, S. Kumar, and S.-F. Chang. On the difficulty of nearest neighbor search. In *International Conference on Machine Learning (ICML)*, 2012.

[98] K. He, X. Zhang, S. Ren, and J. Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016.

[99] K. He, H. Fan, Y. Wu, S. Xie, and R. Girshick. Momentum contrast for unsupervised visual representation learning. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 9729–9738, 2020.

[100] K. He, X. Chen, S. Xie, Y. Li, P. Dollár, and R. Girshick. Masked autoencoders are scalable vision learners. *arXiv preprint arXiv:2111.06377*, 2021.

[101] Y. He, X. Zhang, and J. Sun. Channel pruning for accelerating very deep neural networks. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 1389–1397, 2017.

[102] 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 *Proceedings of the European Conference on Computer Vision (ECCV)*, pages 784–800, 2018.

[103] J. Hegdé. Time course of visual perception: coarse-to-fine processing and beyond. *Progress in neurobiology*, 84(4):405–439, 2008.

[104] D. Hendrycks and K. Gimpel. A baseline for detecting misclassified and out-of-distribution examples in neural networks. *arXiv preprint arXiv:1610.02136*, 2016.

[105] D. Hendrycks and K. Gimpel. Gaussian error linear units (gelus). *arXiv preprint arXiv:1606.08415*, 2016.

[106] D. Hendrycks, S. Basart, N. Mu, S. Kadavath, F. Wang, E. Dorundo, R. Desai, T. Zhu, S. Parajuli, M. Guo, et al. The many faces of robustness: A critical analysis of out-of-distribution generalization. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 8340–8349, 2021.

[107] D. Hendrycks, K. Zhao, S. Basart, J. Steinhardt, and D. Song. Natural adversarial examples. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 15262–15271, 2021.

[108] J. Hoffmann, S. Borgeaud, A. Mensch, E. Buchatskaya, T. Cai, E. Rutherford, D. d. L. Casas, L. A. Hendricks, J. Welbl, A. Clark, et al. Training compute-optimal large language models. *arXiv preprint arXiv:2203.15556*, 2022.

[109] H. Hotelling. Analysis of a complex of statistical variables into principal components. *Journal of educational psychology*, 24(6):417, 1933.

[110] L. Hou, Z. Huang, L. Shang, X. Jiang, X. Chen, and Q. Liu. Dynabert: Dynamic bert with adaptive width and depth. *Advances in Neural Information Processing Systems*, 33:9782–9793, 2020.

[111] 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.

[112] J. Howard and S. Ruder. Universal language model fine-tuning for text classification. *arXiv preprint arXiv:1801.06146*, 2018.

[113] D. Hsu, S. M. Kakade, J. Langford, and T. Zhang. Multi-label prediction via compressed sensing. *arXiv preprint arXiv:0902.1284*, 2009.

[114] H. Hu, D. Dey, M. Hebert, and J. A. Bagnell. Learning anytime predictions in neural networks via adaptive loss balancing. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 33, pages 3812–3821, 2019.

[115] Z. Huang and N. Wang. Data-driven sparse structure selection for deep neural networks. In *Proceedings of the European conference on computer vision (ECCV)*, pages 304–320, 2018.

[116] I. Hubara, M. Courbariaux, D. Soudry, R. El-Yaniv, and Y. Bengio. Binarized neural networks. In *Proceedings of the 30th International Conference on Neural Information Processing Systems*, pages 4114–4122, 2016.

[117] Humane. Humane ai pin. *Humane blog*, 2023. URL https://hu.ma.ne/aipin.

[118] P. Indyk and R. Motwani. Approximate nearest neighbors: towards removing the curse of dimensionality. In *Proceedings of the thirtieth annual ACM symposium on Theory of computing*, pages 604–613, 1998.

[119] M. Jaderberg, A. Vedaldi, and A. Zisserman. Speeding up convolutional neural networks with low rank expansions. In *Proceedings of the British Machine Vision Conference. BMVA Press*, 2014.

[120] D. Jain, K. Huynh Anh Nguyen, S. M. Goodman, R. Grossman-Kahn, H. Ngo, A. Kusupati, R. Du, A. Olwal, L. Findlater, and J. E. Froehlich. Protosound: A personalized and scalable sound recognition system for deaf and hard-of-hearing users. In *Proceedings of the 2022 CHI Conference on Human Factors in Computing Systems*, 2022.

[121] G. Jain, N. Hegde, A. Kusupati, A. Nagrani, S. Buch, P. Jain, A. Arnab, and S. Paul. Mixture of nested experts: Adaptive processing of visual tokens. *arXiv preprint*, 2024.

[122] H. Jain, V. Balasubramanian, B. Chunduri, and M. Varma. Slice: Scalable linear extreme classifiers trained on 100 million labels for related searches. In *Proceedings of the Twelfth ACM International Conference on Web Search and Data Mining*, pages 528–536, 2019.

[123] P. Jain, A. Tewari, and P. Kar. On iterative hard thresholding methods for high-dimensional m-estimation. In *Advances in Neural Information Processing Systems*, pages 685–693, 2014.

[124] A. Jaiswal, Z. Gan, X. Du, B. Zhang, Z. Wang, and Y. Yang. Compressing llms: The truth is rarely pure and never simple, 2023.

[125] S. Jayaram Subramanya, F. Devvrit, H. V. Simhadri, R. Krishnawamy, and R. Kadekodi. Diskann: Fast accurate billion-point nearest neighbor search on a single node. *Advances in Neural Information Processing Systems*, 32, 2019.

[126] H. Jegou, M. Douze, and C. Schmid. Product quantization for nearest neighbor search. *IEEE transactions on pattern analysis and machine intelligence*, 33(1):117–128, 2010.

[127] C. Jia, Y. Yang, Y. Xia, Y.-T. Chen, Z. Parekh, H. Pham, Q. Le, Y.-H. Sung, Z. Li, and T. Duerig. Scaling up visual and vision-language representation learning with noisy text supervision. In *International Conference on Machine Learning*, pages 4904–4916. PMLR, 2021.

[128] J. Johnson, M. Douze, and H. Jégou. Billion-scale similarity search with GPUs. *IEEE Transactions on Big Data*, 7(3):535–547, 2019.

[129] W. B. Johnson. Extensions of lipschitz mappings into a hilbert space. *Contemp. Math.*, 26:189–206, 1984.

[130] I. T. Jolliffe and J. Cadima. Principal component analysis: a review and recent developments. *Philosophical Transactions of the Royal Society A: Mathematical, Physical and Engineering Sciences*, 374(2065):20150202, 2016.

[131] D. Kahneman. *Thinking, fast and slow*. macmillan, 2011.

[132] J. Kaplan, S. McCandlish, T. Henighan, T. B. Brown, B. Chess, R. Child, S. Gray, A. Radford, J. Wu, and D. Amodei. Scaling laws for neural language models. *arXiv*, 2020.

[133] V. Karpukhin, B. Oğuz, S. Min, P. Lewis, L. Wu, S. Edunov, D. Chen, and W.-t. Yih. Dense passage retrieval for open-domain question answering. *arXiv preprint arXiv:2004.04906*, 2020.

[134] T. C. Kaz Sato. Vertex ai matching engine. *Microsoft AI Blog*, 2021. URL https://cloud.google.com/blog/topics/developers-practitioners/find-anything-blazingly-fast-googles-vector-search-technology.

[135] S. Kornblith, J. Shlens, and Q. V. Le. Do better imagenet models transfer better? In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 2661–2671, 2019.

[136] T. Kraska, A. Beutel, E. H. Chi, J. Dean, and N. Polyzotis. The case for learned index structures. In *Proceedings of the 2018 international conference on management of data*, pages 489–504, 2018.

[137] A. Krizhevsky, I. Sutskever, and G. E. Hinton. Imagenet classification with deep convolutional neural networks. *Advances in neural information processing systems*, 25, 2012.

[138] S. Kudugunta, Y. Huang, A. Bapna, M. Krikun, D. Lepikhin, M.-T. Luong, and O. Firat. Beyond distillation: Task-level mixture-of-experts for efficient inference. *arXiv preprint arXiv:2110.03742*, 2021.

[139] S. Kudugunta, I. Caswell, B. Zhang, X. Garcia, C. A. Choquette-Choo, K. Lee, D. Xin, A. Kusupati, R. Stella, A. Bapna, et al. MADLAD-400: A multilingual and document-level large audited dataset. *Advances in Neural Information Processing Systems, Datasets and Benchmarks*, 2023.

[140] B. Kulis and T. Darrell. Learning to hash with binary reconstructive embeddings. In *NIPS*, volume 22, pages 1042–1050. Citeseer, 2009.

[141] B. Kulis and K. Grauman. Kernelized locality-sensitive hashing for scalable image search. In *2009 IEEE 12th international conference on computer vision*, pages 2130–2137. IEEE, 2009.

[142] B. Kulis, P. Jain, and K. Grauman. Fast similarity search for learned metrics. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 31(12):2143–2157, 2009.

[143] R. Kumar, A. Mittal, N. Gupta, A. Kusupati, I. Dhillon, and P. Jain. EHI: End-to-end learning of hierarchical index for efficient dense retrieval. *arXiv:2310.08891*, 2023.

[144] A. Kusupati, M. Singh, K. Bhatia, A. Kumar, P. Jain, and M. Varma. Fastgrnn: A fast, accurate, stable and tiny kilobyte sized gated recurrent neural network. *Advances in Neural Information Processing Systems*, 31, 2018.

[145] A. Kusupati, V. Ramanujan, R. Somani, M. Wortsman, P. Jain, S. Kakade, and A. Farhadi. Soft threshold weight reparameterization for learnable sparsity. In *International Conference on Machine Learning*, pages 5544–5555. PMLR, 2020.

[146] A. Kusupati, M. Wallingford, V. Ramanujan, R. Somani, J. S. Park, K. Pillutla, P. Jain, S. Kakade, and A. Farhadi. Llc: Accurate, multi-purpose learnt low-dimensional binary codes. *Advances in Neural Information Processing Systems*, 34, 2021.

[147] A. Kusupati, G. Bhatt, A. Rege, M. Wallingford, A. Sinha, V. Ramanujan, W. Howard-Snyder, K. Chen, S. Kakade, P. Jain, et al. Matryoshka representation learning. *Advances in Neural Information Processing Systems*, 35:30233–30249, 2022.

[148] T. Kwiatkowski, J. Palomaki, O. Redfield, M. Collins, A. Parikh, C. Alberti, D. Epstein, I. Polosukhin, J. Devlin, K. Lee, et al. Natural questions: a benchmark for question answering research. *Transactions of the Association for Computational Linguistics*, 7:453–466, 2019.

[149] F. Lagunas, E. Charlaix, V. Sanh, and A. M. Rush. Block pruning for faster transformers. *arXiv preprint arXiv:2109.04838*, 2021.

[150] C. H. Lampert, H. Nickisch, and S. Harmeling. Learning to detect unseen object classes by between-class attribute transfer. In *2009 IEEE Conference on Computer Vision and Pattern Recognition*, pages 951–958. IEEE, 2009.

[151] C. H. Lampert, H. Nickisch, and S. Harmeling. Attribute-based classification for zero-shot visual object categorization. *IEEE transactions on pattern analysis and machine intelligence*, 36(3):453–465, 2013.

[152] Y. LeCun, J. S. Denker, and S. A. Solla. Optimal brain damage. In *Advances in neural information processing systems*, pages 598–605, 1990.

[153] Y. LeCun, Y. Bengio, and G. Hinton. Deep learning. *nature*, 521(7553):436–444, 2015.

[154] J. Lee, Z. Dai, X. Ren, B. Chen, D. Cer, J. R. Cole, K. Hui, M. Boratko, R. Kapadia, W. Ding, et al. Gecko: Versatile text embeddings distilled from large language models. *arXiv preprint arXiv:2403.20327*, 2024.

[155] N. Lee, T. Ajanthan, and P. Torr. SNIP: Single-shot network pruning based on connection sensitivity. In *International Conference on Learning Representations*, 2019.

[156] Y. Lee. Differentiable sparsification for deep neural networks. *arXiv preprint arXiv:1910.03201*, 2019.

[157] Y. Leviathan, M. Kalman, and Y. Matias. Fast inference from transformers via speculative decoding. *arXiv*, 2023.

[158] D. A. Levin and Y. Peres. *Markov chains and mixing times*, volume 107. American Mathematical Soc., 2017.

[159] P. Lewis, E. Perez, A. Piktus, F. Petroni, V. Karpukhin, N. Goyal, H. Küttler, M. Lewis, et al. Retrieval-augmented generation for knowledge-intensive nlp tasks. *Advances in Neural Information Processing Systems*, 2020.

[160] C. Li, H. Farkhoor, R. Liu, and J. Yosinski. Measuring the intrinsic dimension of objective landscapes. *arXiv preprint arXiv:1804.08838*, 2018.

[161] H. Li, A. Kadav, I. Durdanovic, H. Samet, and H. P. Graf. Pruning filters for efficient convnets. In *International Conference on Learning Representations*, 2017.

[162] M. Li, S. Gururangan, T. Dettmers, M. Lewis, T. Althoff, N. A. Smith, and L. Zettlemoyer. Branch-train-merge: Embarrassingly parallel training of expert language models. *arXiv preprint arXiv:2208.03306*, 2022.

[163] W. Li, Y. Zhang, Y. Sun, W. Wang, W. Zhang, and X. Lin. Approximate nearest neighbor search on high dimensional data—experiments, analyses, and improvement. *IEEE Transactions on Knowledge and Data Engineering*, 2020.

[164] T. Lin, S. U. Stich, L. Barba, D. Dmitriev, and M. Jaggi. Dynamic model pruning with feedback. In *International Conference on Learning Representations*, 2020.

[165] Y. Linde, A. Buzo, and R. Gray. An algorithm for vector quantizer design. *IEEE Transactions on communications*, 28(1):84–95, 1980.

[166] B. Liu, M. Wang, H. Foroosh, M. Tappen, and M. Pensky. Sparse convolutional neural networks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 806–814, 2015.

[167] H. Liu, R. Wang, S. Shan, and X. Chen. Deep supervised hashing for fast image retrieval. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 2064–2072, 2016.

[168] J. Liu, Z. Xu, R. Shi, R. C. C. Cheung, and H. K. So. Dynamic sparse training: Find efficient sparse network from scratch with trainable masked layers. In *International Conference on Learning Representations*, 2020.

[169] P. J. Liu, M. Saleh, E. Pot, B. Goodrich, R. Sepassi, L. Kaiser, and N. Shazeer. Generating wikipedia by summarizing long sequences. *arXiv preprint arXiv:1801.10198*, 2018.

[170] Z. Liu, M. Sun, T. Zhou, G. Huang, and T. Darrell. Rethinking the value of network pruning. In *International Conference on Learning Representations*, 2019.

[171] Z. Liu, H. Mao, C.-Y. Wu, C. Feichtenhofer, T. Darrell, and S. Xie. A convnet for the 2020s. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 11976–11986, 2022.

[172] S. Lloyd. Least squares quantization in pcm. *IEEE transactions on information theory*, 28(2):129–137, 1982.

[173] C. Louizos, M. Welling, and D. P. Kingma. Learning sparse neural networks through l_0 regularization. In *International Conference on Learning Representations*, 2018.

[174] Z. Lu, V. Sindhwani, and T. N. Sainath. Learning compact recurrent neural networks. In *2016 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 5960–5964. IEEE, 2016.

[175] J.-H. Luo, J. Wu, and W. Lin. Thinet: A filter level pruning method for deep neural network compression. In *Proceedings of the IEEE international conference on computer vision*, pages 5058–5066, 2017.

[176] X. Luo, C. Chen, H. Zhong, H. Zhang, M. Deng, J. Huang, and X. Hua. A survey on deep hashing methods. *arXiv preprint arXiv:2003.03369*, 2020.

[177] Y. Malkov, A. Ponomarenko, A. Logvinov, and V. Krylov. Approximate nearest neighbor algorithm based on navigable small world graphs. *Information Systems*, 45:61–68, 2014.

[178] Y. A. Malkov and D. Yashunin. Efficient and robust approximate nearest neighbor search using hierarchical navigable small world graphs. *IEEE Transactions on Pattern Analysis & Machine Intelligence*, 42(04):824–836, 2020.

[179] Y. A. Malkov and D. A. Yashunin. Efficient and robust approximate nearest neighbor search using hierarchical navigable small world graphs. *IEEE transactions on pattern analysis and machine intelligence*, 42(4):824–836, 2018.

[180] K. Mardia, J. Kent, and J. Bibby. Multivariate analysis. *Probability and Mathematical Statistics*, 1979.

[181] J. Masci, U. Meier, D. Cireşan, and J. Schmidhuber. Stacked convolutional auto-encoders for hierarchical feature extraction. In *International conference on artificial neural networks*, pages 52–59. Springer, 2011.

[182] P. Mitra, C. Murthy, and S. K. Pal. Unsupervised feature selection using feature similarity. *IEEE transactions on pattern analysis and machine intelligence*, 24(3):301–312, 2002.

[183] D. C. Mocanu, E. Mocanu, P. Stone, P. H. Nguyen, M. Gibescu, and A. Liotta. Scalable training of artificial neural networks with adaptive sparse connectivity inspired by network science. *Nature communications*, 9(1):2383, 2018.

[184] D. Molchanov, A. Ashukha, and D. Vetrov. Variational dropout sparsifies deep neural networks. In *Proceedings of the 34th International Conference on Machine Learning-Volume 70*, pages 2498–2507. JMLR. org, 2017.

[185] H. Moravec. When will computer hardware match the human brain. *Journal of evolution and technology*, 1998.

[186] H. Mostafa and X. Wang. Parameter efficient training of deep convolutional neural networks by dynamic sparse reparameterization. In *International Conference on Machine Learning*, pages 4646–4655, 2019.

[187] F. Murtagh and P. Contreras. Algorithms for hierarchical clustering: an overview. *Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery*, 2(1):86–97, 2012.

[188] V. Nanda, T. Speicher, J. P. Dickerson, S. Feizi, K. P. Gummadi, and A. Weller. Diffused redundancy in pre-trained representations. *arXiv preprint arXiv:2306.00183*, 2023.

[189] S. Narang, E. Elsen, G. Diamos, and S. Sengupta. Exploring sparsity in recurrent neural networks. In *International Conference on Learning Representations*, 2019.

[190] P. Nayak. Understanding searches better than ever before. *Google AI Blog*, 2019. URL https://blog.google/products/search/search-language-understanding-bert/.

[191] A. Neelakantan, T. Xu, R. Puri, A. Radford, J. M. Han, J. Tworek, Q. Yuan, N. Tezak, J. W. Kim, C. Hallacy, et al. Text and code embeddings by contrastive pre-training. *arXiv preprint arXiv:2201.10005*, 2022.

[192] M. Norouzi, D. J. Fleet, and R. R. Salakhutdinov. Hamming distance metric learning. In *Advances in neural information processing systems*, pages 1061–1069, 2012.

[193] M. Norouzi, T. Mikolov, S. Bengio, Y. Singer, J. Shlens, A. Frome, G. S. Corrado, and J. Dean. Zero-shot learning by convex combination of semantic embeddings. *arXiv preprint arXiv:1312.5650*, 2013.

[194] OpenAI. New embedding models and api updates. 2024. URL https://openai.com/index/new-embedding-models-and-api-updates/.

[195] R. OpenAI. Gpt-4 technical report. *arXiv*, pages 2303–08774, 2023.

[196] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga, et al. Pytorch: An imperative style, high-performance deep learning library. *Advances in neural information processing systems*, 32, 2019.

[197] S. G. Patil, D. K. Dennis, C. Pabbaraju, N. Shaheer, H. V. Simhadri, V. Seshadri, M. Varma, and P. Jain. Gesturepod: Enabling on-device gesture-based interaction for white cane users. In *Proceedings of the 32nd Annual ACM Symposium on User Interface Software and Technology*, pages 403–415, 2019.

[198] M. E. Peters, M. Neumann, M. Iyyer, M. Gardner, C. Clark, K. Lee, and L. Zettlemoyer. Deep contextualized word representations. In *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long Papers)*, pages 2227–2237, New Orleans, Louisiana, June 2018. Association for Computational Linguistics. doi: 10.18653/v1/N18-1202. URL https://aclanthology.org/N18-1202.

[199] Pinterest. Linksage: Gnn-based pinterest off-site content understanding. 2024. URL https://medium.com/pinterest-engineering/linksage-gnn-based-pinterest-off-site-content-understanding-fca14b0d1141.

[200] Y. Prabhu, A. Kusupati, N. Gupta, and M. Varma. Extreme regression for dynamic search advertising. In *Proceedings of the 13th International Conference on Web Search and Data Mining*, pages 456–464, 2020.

[201] A. Radford, K. Narasimhan, T. Salimans, and I. Sutskever. Improving language understanding by generative pre-training. *OpenAI Blog*, 2018. URL https://openai.com/blog/language-unsupervised/.

[202] A. Radford, J. W. Kim, C. Hallacy, A. Ramesh, G. Goh, S. Agarwal, G. Sastry, A. Askell, P. Mishkin, J. Clark, et al. Learning transferable visual models from natural language supervision. In *International Conference on Machine Learning*, pages 8748–8763. PMLR, 2021.

[203] A. Radford, J. W. Kim, T. Xu, G. Brockman, C. McLeavey, and I. Sutskever. Robust speech recognition via large-scale weak supervision. In *International Conference on Machine Learning*, pages 28492–28518. PMLR, 2023.

[204] V. Ramanujan, P. K. A. Vasu, A. Farhadi, O. Tuzel, and H. Pouransari. Forward compatible training for representation learning. *arXiv preprint arXiv:2112.02805*, 2021.

[205] M. Rastegari, A. Farhadi, and D. Forsyth. Attribute discovery via predictable discriminative binary codes. In *European Conference on Computer Vision*, pages 876–889. Springer, 2012.

[206] M. Rastegari, V. Ordonez, J. Redmon, and A. Farhadi. Xnor-net: Imagenet classification using binary convolutional neural networks. In *European conference on computer vision*, pages 525–542. Springer, 2016.

[207] B. Recht, R. Roelofs, L. Schmidt, and V. Shankar. Do imagenet classifiers generalize to imagenet? In *International Conference on Machine Learning*, pages 5389–5400. PMLR, 2019.

[208] A. Rege and A. Kusupati. FReAD: Faithful retrieval augmented diffusion models. *Work in progress*, 2023.

[209] A. Rege, A. Kusupati, A. Fan, Q. Cao, S. Kakade, P. Jain, A. Farhadi, et al. Adanns: A framework for adaptive semantic search. *Advances in Neural Information Processing Systems*, 36, 2023.

[210] A. Renda, J. Frankle, and M. Carbin. Comparing fine-tuning and rewinding in neural network pruning. In *International Conference on Learning Representations*, 2020.

[211] O. Rippel, M. Gelbart, and R. Adams. Learning ordered representations with nested dropout. In *International Conference on Machine Learning*, pages 1746–1754. PMLR, 2014.

[212] J. Rissanen. Modeling by shortest data description. *Automatica*, 14(5):465–471, 1978.

[213] D. Roy, S. Srivastava, A. Kusupati, P. Jain, M. Varma, and A. Arora. One size does not fit all: Multi-scale, cascaded rnns for radar classification. In *Proceedings of the 6th ACM International Conference on Systems for Energy-Efficient Buildings, Cities, and Transportation*, pages 1–10, 2019.

[214] S. Ruder, M. E. Peters, S. Swayamdipta, and T. Wolf. Transfer learning in natural language processing. In *Proceedings of the 2019 conference of the North American chapter of the association for computational linguistics: Tutorials*, pages 15–18, 2019.

[215] 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. *International journal of computer vision*, 115(3):211–252, 2015.

[216] R. Salakhutdinov and G. Hinton. Learning a nonlinear embedding by preserving class neighbourhood structure. In *Artificial Intelligence and Statistics*, pages 412–419. PMLR, 2007.

[217] R. Salakhutdinov and G. Hinton. Semantic hashing. *International Journal of Approximate Reasoning*, 50(7):969–978, 2009.

[218] M. Salehi, S. Mehta, A. Kusupati, A. Farhadi, and H. Hajishirzi. Sharcs: Efficient transformers through routing with dynamic width sub-networks. *Findings of Empirical Methods in Natural Language Processing*, 2023.

[219] J. S. Sánchez, F. Pla, and F. J. Ferri. On the use of neighbourhood-based non-parametric classifiers. *Pattern Recognition Letters*, 18(11-13):1179–1186, 1997.

[220] V. Sanh, L. Debut, J. Chaumond, and T. Wolf. Distilbert, a distilled version of bert: smaller, faster, cheaper and lighter. *arXiv preprint arXiv:1910.01108*, 2019.

[221] P. Savarese, H. Silva, and M. Maire. Winning the lottery with continuous sparsification. *arXiv preprint arXiv:1912.04427*, 2019.

[222] T. Schuster, A. Fisch, J. Gupta, M. Dehghani, D. Bahri, V. Tran, Y. Tay, and D. Metzler. Confident adaptive language modeling. *Advances in Neural Information Processing Systems*, 35:17456–17472, 2022.

[223] R. Schwartz, J. Dodge, N. A. Smith, and O. Etzioni. Green AI. *arXiv preprint arXiv:1907.10597*, 2019.

[224] R. R. Selvaraju, M. Cogswell, A. Das, R. Vedantam, D. Parikh, and D. Batra. Grad-cam: Visual explanations from deep networks via gradient-based localization. In *Proceedings of the IEEE international conference on computer vision*, pages 618–626, 2017.

[225] N. Shazeer and M. Stern. Adafactor: Adaptive learning rates with sublinear memory cost. In *International Conference on Machine Learning*, pages 4596–4604. PMLR, 2018.

[226] E. Shen, A. Fan, S. Pratt, J. S. Park, M. Wallingford, S. Kakade, A. Holtzman, R. Krishna, A. Farhadi, and A. Kusupati. Superposed decoding: Multiple generations from a single autoregressive inference pass. *arXiv preprint*, 2024.

[227] F. Shen, Y. Xu, L. Liu, Y. Yang, Z. Huang, and H. T. Shen. Unsupervised deep hashing with similarity-adaptive and discrete optimization. *IEEE transactions on pattern analysis and machine intelligence*, 40(12):3034–3044, 2018.

[228] H. V. Simhadri, G. Williams, M. Aumüller, M. Douze, A. Babenko, D. Baranchuk, Q. Chen, L. Hosseini, R. Krishnaswamy, G. Srinivasa, et al. Results of the neurips'21 challenge on billion-scale approximate nearest neighbor search. *arXiv preprint arXiv:2205.03763*, 2022.

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

[230] J. Sivic and A. Zisserman. Video google: A text retrieval approach to object matching in videos. In *Computer Vision, IEEE International Conference on*, volume 3, pages 1470–1470. IEEE Computer Society, 2003.

[231] D. R. So, W. Mańke, H. Liu, Z. Dai, N. Shazeer, and Q. V. Le. Primer: Searching for efficient transformers for language modeling. *arXiv preprint arXiv:2109.08668*, 2021.

[232] D. Soudry, E. Hoffer, M. S. Nacson, S. Gunasekar, and N. Srebro. The implicit bias of gradient descent on separable data. *The Journal of Machine Learning Research*, 19(1):2822–2878, 2018.

[233] A. Steiner, A. Kolesnikov, X. Zhai, R. Wightman, J. Uszkoreit, and L. Beyer. How to train your vit? data, augmentation, and regularization in vision transformers. *arXiv preprint arXiv:2106.10270*, 2021.

[234] S. Su, C. Zhang, K. Han, and Y. Tian. Greedy hash: Towards fast optimization for accurate hash coding in cnn. In *Proceedings of the 32nd International Conference on Neural Information Processing Systems*, pages 806–815, 2018.

[235] C. Sun, A. Shrivastava, S. Singh, and A. Gupta. Revisiting unreasonable effectiveness of data in deep learning era. In *Proceedings of the IEEE international conference on computer vision*, pages 843–852, 2017.

[236] Supabase. Matryoshka embeddings: faster openai vector search using adaptive retrieval. 2024. URL https://supabase.com/blog/matryoshka-embeddings.

[237] M. Tan and Q. Le. Efficientnet: Rethinking model scaling for convolutional neural networks. In *International conference on machine learning*, pages 6105–6114. PMLR, 2019.

[238] R. Thoppilan, D. De Freitas, J. Hall, N. Shazeer, A. Kulshreshtha, H.-T. Cheng, A. Jin, T. Bos, L. Baker, Y. Du, et al. Lamda: Language models for dialog applications. *arXiv preprint arXiv:2201.08239*, 2022.

[239] H. Touvron, T. Lavril, G. Izacard, X. Martinet, M.-A. Lachaux, T. Lacroix, B. Rozière, N. Goyal, E. Hambro, F. Azhar, et al. Llama: Open and efficient foundation language models. *arXiv preprint arXiv:2302.13971*, 2023.

[240] H. Touvron, L. Martin, K. Stone, P. Albert, A. Almahairi, Y. Babaei, N. Bashlykov, S. Batra, P. Bhargava, S. Bhosale, et al. Llama 2: Open foundation and fine-tuned chat models. *arXiv preprint arXiv:2307.09288*, 2023.

[241] M. Valipour, M. Rezagholizadeh, H. Rajabzadeh, M. Tahaei, B. Chen, and A. Ghodsi. Sortednet, a place for every network and every network in its place: Towards a generalized solution for training many-in-one neural networks. *arXiv preprint arXiv:2309.00255*, 2023.

[242] L. Van Der Maaten, E. Postma, J. Van den Herik, et al. Dimensionality reduction: a comparative. *J Mach Learn Res*, 10(66-71):13, 2009.

[243] M. van Kempen, S. S. Kim, C. Tumescheit, M. Mirdita, C. L. Gilchrist, J. Söding, and M. Steinegger. Foldseek: fast and accurate protein structure search. *Biorxiv*, pages 2022–02, 2022.

[244] M. Varma. Extreme classification. *Communications of the ACM*, 62(11):44–45, 2019.

[245] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, and I. Polosukhin. Attention is all you need. In *Advances in neural information processing systems*, pages 5998–6008, 2017.

[246] P. Viola and M. Jones. Rapid object detection using a boosted cascade of simple features. In *Proceedings of the 2001 IEEE computer society conference on computer vision and pattern recognition. CVPR 2001*, volume 1, pages I–I. Ieee, 2001.

[247] A. Visheratin. Nllb-siglip-mrl. 2024. URL https://huggingface.co/visheratin/nllb-siglip-mrl-large.

[248] C. Waldburger. As search needs evolve, microsoft makes ai tools for better search available to researchers and developers. *Microsoft AI Blog*, 2019. URL https://blogs.microsoft.com/ai/bing-vector-search/.

[249] M. Wallingford, A. Kusupati, K. Alizadeh-Vahid, A. Walsman, A. Kembhavi, and A. Farhadi. Are we overfitting to experimental setups in recognition? *arXiv preprint arXiv:2007.02519*, 2020.

[250] M. Wallingford, H. Li, A. Achille, A. Ravichandran, C. Fowlkes, R. Bhotika, and S. Soatto. Task adaptive parameter sharing for multi-task learning. *arXiv preprint arXiv:2203.16708*, 2022.

[251] M. Wallingford, A. Kusupati, K. Alizadeh-Vahid, A. Walsman, A. Kembhavi, and A. Farhadi. FLUID: A unified evaluation framework for flexible sequential data. *Transactions on Machine Learning Research*, 2023.

[252] M. Wallingford, A. Kusupati, A. Fang, V. Ramanujan, A. Kembhavi, R. Mottaghi, and A. Farhadi. Neural radiance field codebooks. *International Conference on Learning Representations*, 2023.

[253] M. Wallingford, A. Bhattad, A. Kusupati, V. Ramanujan, M. Deitke, A. Kembhavi, R. Mottaghi, W.-C. Ma, and A. Farhadi. From an image to a scene: Learning to imagine the world from a million 360 videos. *arXiv preprint*, 2024.

[254] H. Wang, S. Ge, Z. Lipton, and E. P. Xing. Learning robust global representations by penalizing local predictive power. In *Advances in Neural Information Processing Systems*, pages 10506–10518, 2019.

[255] H. Wang, Z. Wu, Z. Liu, H. Cai, L. Zhu, C. Gan, and S. Han. Hat: Hardware-aware transformers for efficient natural language processing. *arXiv preprint arXiv:2005.14187*, 2020.

[256] J. Wang, W. Liu, S. Kumar, and S.-F. Chang. Learning to hash for indexing big data—a survey. *Proceedings of the IEEE*, 104(1):34–57, 2015.

[257] M. Wang, X. Xu, Q. Yue, and Y. Wang. A comprehensive survey and experimental comparison of graph-based approximate nearest neighbor search. *Proceedings of the VLDB Endowment*, 14(11): 1964–1978, 2021.

[258] X. Wang, D. Kondratyuk, K. M. Kitani, Y. Movshovitz-Attias, and E. Eban. Multiple networks are more efficient than one: Fast and accurate models via ensembles and cascades. *arXiv preprint arXiv:2012.01988*, 2020.

[259] P. Warden. Speech commands: A dataset for limited-vocabulary speech recognition. *arXiv preprint arXiv:1804.03209*, 2018. URL http://download.tensorflow.org/data/speech_commands_v0.01.tar.gz.

[260] R. Weber, H.-J. Schek, and S. Blott. A quantitative analysis and performance study for similarity-search methods in high-dimensional spaces. In *VLDB*, volume 98, pages 194–205, 1998.

[261] Y. Weiss, A. Torralba, R. Fergus, et al. Spectral hashing. In *Advances in Neural Information Processing Systems*, 2008.

[262] W. Wen, C. Wu, Y. Wang, Y. Chen, and H. Li. Learning structured sparsity in deep neural networks. In *Advances in neural information processing systems*, pages 2074–2082, 2016.

[263] J. Weston, S. Bengio, and N. Usunier. Wsabie: Scaling up to large vocabulary image annotation. In *Twenty-Second International Joint Conference on Artificial Intelligence*, 2011.

[264] T. Weyand, A. Araujo, B. Cao, and J. Sim. Google landmarks dataset v2-a large-scale benchmark for instance-level recognition and retrieval. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 2575–2584, 2020.

[265] I. H. Witten, I. H. Witten, A. Moffat, T. C. Bell, T. C. Bell, E. Fox, and T. C. Bell. *Managing gigabytes: compressing and indexing documents and images*. Morgan Kaufmann, 1999.

[266] M. Wortsman, A. Farhadi, and M. Rastegari. Discovering neural wirings. In *Advances In Neural Information Processing Systems*, pages 2680–2690, 2019.

[267] M. Wortsman, G. Ilharco, M. Li, J. W. Kim, H. Hajishirzi, A. Farhadi, H. Namkoong, and L. Schmidt. Robust fine-tuning of zero-shot models. *arXiv preprint arXiv:2109.01903*, 2021.

[268] Z. Wu, Y. Xiong, S. Yu, and D. Lin. Unsupervised feature learning via non-parametric instance-level discrimination. *arXiv preprint arXiv:1805.01978*, 2018.

[269] X. Xiao, Z. Wang, and S. Rajasekaran. Autoprune: Automatic network pruning by regularizing auxiliary parameters. In *Advances in Neural Information Processing Systems*, pages 13681–13691, 2019.

[270] J. Yosinski, J. Clune, Y. Bengio, and H. Lipson. How transferable are features in deep neural networks? *Advances in neural information processing systems*, 27, 2014.

[271] H.-F. Yu, P. Jain, P. Kar, and I. Dhillon. Large-scale multi-label learning with missing labels. In *International conference on machine learning*, pages 593–601. PMLR, 2014.

[272] H.-F. Yu, K. Zhong, J. Zhang, W.-C. Chang, and I. S. Dhillon. Pecos: Prediction for enormous and correlated output spaces. *Journal of Machine Learning Research*, 23(98):1–32, 2022.

[273] J. Yu and T. Huang. Network slimming by slimmable networks: Towards one-shot architecture search for channel numbers. *arXiv preprint arXiv:1903.11728*, 2019.

[274] J. Yu and T. S. Huang. Universally slimmable networks and improved training techniques. In *Proceedings of the IEEE/CVF international conference on computer vision*, pages 1803–1811, 2019.

[275] J. Yu, L. Yang, N. Xu, J. Yang, and T. Huang. Slimmable neural networks. *arXiv preprint arXiv:1812.08928*, 2018.

[276] X. Yuan, L. Ren, J. Lu, and J. Zhou. Relaxation-free deep hashing via policy gradient. In *Proceedings of the European Conference on Computer Vision (ECCV)*, pages 134–150, 2018.

[277] R. Zellers, J. Lu, X. Lu, Y. Yu, Y. Zhao, M. Salehi, A. Kusupati, J. Hessel, A. Farhadi, and Y. Choi. Merlot reserve: Neural script knowledge through vision and language and sound. *arXiv preprint arXiv:2201.02639*, 2022.

[278] C. Zhang and Y. Ma. *Ensemble machine learning: methods and applications*. Springer, 2012.

[279] X. Zhang. A survey on deep hashing for image retrieval. *arXiv preprint arXiv:2006.05627*, 2020.

[280] Y. Zhang and J. Schneider. Multi-label output codes using canonical correlation analysis. In *Proceedings of the fourteenth international conference on artificial intelligence and statistics*, pages 873–882. JMLR Workshop and Conference Proceedings, 2011.

[281] B. Zhou, A. Lapedriza, J. Xiao, A. Torralba, and A. Oliva. Learning deep features for scene recognition using places database. In *Advances in Neural Information Processing Systems*, 2014.

[282] M. Zhu and S. Gupta. To prune, or not to prune: exploring the efficacy of pruning for model compression. *arXiv preprint arXiv:1710.01878*, 2017.

[283] Y. Zhu, R. Kiros, R. Zemel, R. Salakhutdinov, R. Urtasun, A. Torralba, and S. Fidler. Aligning books and movies: Towards story-like visual explanations by watching movies and reading books. In *Proceedings of the IEEE international conference on computer vision*, pages 19–27, 2015.