Improving Efficiency of Deep Representation Learning

By

KOSSAR POURAHMADI-MEIBODI
DISSERTATION

Submitted in partial satisfaction of the requirements for the degree of

DOCTOR OF PHILOSOPHY

in

Computer Science

in the

OFFICE OF GRADUATE STUDIES

of the

UNIVERSITY OF CALIFORNIA

DAVIS

Approved:

Hamed Pirsiavash, Chair

Zhe Zhao

Lifu Huang

Committee in Charge

2026

© Kossar Pourahmadi-Meibodi, 2026. All rights reserved.

To my homeland.

TABLE OF CONTENTS

Page

1 Introduction **1**

2 CMSF: Constrained Mean Shift Using Distant Yet Related Neighbors **4**
2.1 Related Work . . . 7
2.2 Method . . . 8
2.2.1 Self-Supervised Setting . . . 9
2.2.2 Supervised Setting . . . 11
2.2.3 Semi-Supervised Setting . . . 11
2.3 Experiments . . . 12
2.3.1 Self-Supervised Learning ($CMSF_{self}$) . . . 13
2.3.2 Supervised Learning . . . 15
2.3.3 Semi-Supervised Learning . . . 18
2.4 Conclusion . . . 20

3 PRANC: Pseudo RAndom Networks for Compacting deep models **21**
3.1 Related work . . . 24
3.2 Method . . . 26
3.3 Application . . . 29
3.4 Experiments on image classification . . . 29
3.4.1 Comparison with model pruning methods: . . . 29
3.4.2 Comparison with model distillation methods: . . . 30
3.4.3 Comparison to Dataset Distillation methods: . . . 31
3.4.4 Large-Scale dataset and models . . . 31
3.4.5 Ablation study: . . . 32
3.5 Experiments on image compression . . . 34
3.6 Conclusion . . . 36

4 CompGS: Smaller and Faster Gaussian Splatting **39**
4.1 Related Work . . . 42
4.2 Method . . . 44
4.3 Experiments . . . 47
4.3.1 Comparison of parameter compression methods: . . . 50
4.3.2 Comparison of Gaussian count compression methods: . . . 53
4.3.3 Results on ARKit-200 and DL3DV datasets: . . . 53
4.3.4 Ablations . . . 54
4.4 Conclusion . . . 56

5 LoRA-merging-aware Training **57**
5.1 Related Work . . . 59
5.2 Method . . . 61
5.3 Experiments . . . 63
5.3.1 Per-task Evaluation . . . 64
5.3.2 Ablations . . . 64
5.4 Conclusion . . . 66

6 Discussion **67**

Appendix A CMSF **69**

Appendix B PRANC **78**

Appendix C CompGS **95**

Bibliography **107**

ACKNOWLEDGMENTS

First, I want to thank my PhD advisor, Dr. Hamed Pirsiavash, for trusting me and giving me the opportunity to be his student for the past five years. Throughout my PhD, he was very supportive and understanding. There were many difficult moments that he made easier simply by being patient and considerate. From him, I also learned to always aim for the best and not settle for something just because it is good enough. I also want to thank my dissertation and qualifying exam committee members, Dr. Zhe Zhao, Dr. Lifu Huang, Dr. Jiawei Zhang, and Dr. Iman Soltani. Their support and feedback were invaluable in shaping this dissertation.

I am also grateful to my former labmates, who graduated a few years before me. Having them around made PhD life much more interesting, motivating, and fun. Even the exhausting and stressful paper submission deadlines became a lot more enjoyable when we were going through them together. I am very thankful for the time we shared, both in and outside the lab.

I would also like to thank Golnoosh Samei, Behnoosh Parsa, Prateek Singhal, and Yusan Lin for believing in me and giving me the opportunity to work with them as an intern. I learned a lot from each of these experiences, and they helped me become more confident in my work. My path would have been very different without their trust and mentorship.

Finally, I want to thank my parents. They patiently listened to every worry, frustration, and stressful moment throughout these years, even when there was little they could do from far away other than listen. They always found a way to remind me of the good things, calm me down, and help me see that whatever felt overwhelming at the moment would eventually pass. Knowing that they were always there gave me the peace of mind to focus on what I loved, keep moving forward through the difficult parts, and make it to the end. I could not have done this without their love and support.

Abstract

Modern machine learning has advanced rapidly, but much of this progress depends on large amounts of labeled data, computation, memory, storage, and communication. These requirements make state-of-the-art systems expensive to reproduce, adapt, and deploy, limiting participation to organizations with substantial resources. Even when code and model checkpoints are publicly available, the cost of working with them can remain out of reach for many academic laboratories and smaller research groups. This dissertation studies how to reduce these barriers and make advanced machine learning more accessible.

The central idea of this dissertation is that efficiency should be considered throughout the full lifecycle of a model. Learning should depend less on expensive supervision. Models and learned representations should require fewer parameters and less storage. Systems should be easier to communicate and deploy on devices with limited memory or bandwidth. Task-specific capabilities should also be reusable and easier to combine, rather than requiring repeated training of large models from scratch.

To support these goals, this dissertation develops methods across visual representation learning, compact neural network parameterization, efficient 3D scene representation, and parameter-efficient model adaptation. Although these problems arise in different settings, they are connected by the same objective: reducing resource demands without giving up the performance and flexibility that make modern machine learning useful. The results show that efficiency can often be improved by changing how models learn, how they represent information, and how they share structure across tasks.

The broader goal of this work is to help move machine learning toward systems that are not only capable, but also affordable, reusable, and practical under limited resources. By lowering the cost of training, storing, deploying, and combining models, this dissertation aims to support a wider and more inclusive research community.

Chapter 1

Introduction

As machine learning models become larger and more capable, their demand for computation, memory, storage, and communication continues to grow. Training a modern model may require large datasets, many accelerators, and long optimization schedules. Deploying the same model can also be expensive, especially when it must run on devices with limited memory or communicate over constrained networks. These costs increasingly determine who can participate in machine learning research. Although code and model checkpoints are often publicly released, reproducing or extending recent work may still be beyond the resources of many academic laboratories.

Improving efficiency is therefore important not only for reducing cost, but also for making machine learning research more accessible. Efficiency can take several forms. A model may require fewer annotations, fewer trainable parameters, less storage, or less communication. It may also be designed so that several independently learned capabilities can be combined without retraining a large model. The goal of my research is to develop methods that improve these different forms of efficiency while preserving the performance and flexibility of modern representation-learning systems.

This dissertation studies four problems that arise at different stages. The first concern is supervision: how can a model learn useful visual representations without relying on large annotated datasets? The second is model representation: can a large neural network be described using only a small number of learned values? The third is compact 3D scene representations: can Gaussian Splatting retain its fast rendering while using much less storage? The final problem concerns model adaptation

and reuse: can task-specific LoRA experts be trained so that they can later be merged into one model with less interference? Although these problems involve different applications, they share the same objective of reducing the resources needed to train, store, communicate, and combine learned models.

The first direction of my research is **efficiency of supervision**. The success of deep neural networks in the past decade has largely relied on supervised learning. However, as models grow, their need for labeled data also grows. The labeling process is expensive, time-consuming, and often introduces ambiguity and privacy concerns. Companies such as ScaleAI and Appen have built large businesses around data labeling. For instance, ScaleAI reached a valuation of $29 billion in July 2025. Reducing dependence on labeled data by enabling models to learn useful representations from unlabeled data can therefore have a major impact. In this direction, chapter 2 introduces **CMSF**, a method that learns strong and transferable features from unlabeled data while keeping computation low.

The second direction of my work is **efficiency of parameters**. As models increase in size, the cost of GPU memory and communication grows nonlinearly, creating a major bottleneck for both training and deployment. This is especially problematic for real-time applications such as virtual reality (VR) systems. For example, Apple Vision Pro or Meta Quest, where 3D scenes must be rendered quickly. These systems often rely on cloud servers to render 3D scenes and send them back to the device, which adds latency and cost. By compressing 3D scene representations, such systems could perform local rendering and remove the need for expensive cloud computation and data transfer.

To address this, chapter 3 and 4 introduce two methods. The first, **PRANC**, reparameterizes neural networks so that only a small number of parameters need to be trained. This approach reduces the memory and compute requirements for both model training and data that has neural representation. The second, **CompGS**, extends our work on data compression by applying vector quantization to compress the 3D Gaussian Splatting representation.

The final direction is efficiency of model adaptation and reuse. Chapter 5 introduces a **merging-aware LoRA training** method in which all task experts share the same frozen low-rank basis and learn only task-specific coefficients. This makes the experts more compatible before merging and

reduces the need for post-hoc alignment.

The main contributions of this dissertation are:

- A constrained nearest-neighbour learning framework that improves self-supervised representation learning and extends naturally to supervised and semi-supervised settings.
- A neural network reparameterization method that represents a large model using a pseudo-random seed and a small set of learned coefficients.
- A compact Gaussian Splatting representation that combines vector quantization with opacity-based Gaussian removal to reduce both storage and rendering time.
- A merging-aware LoRA training strategy that places task experts in a shared low-rank basis, improving their compatibility before merging.

Together, these chapters study efficiency across supervision, parameterization, 3D scene representation, and model reuse, with the broader goal of making advanced machine learning systems easier to train, store, deploy, and combine under limited resources.

Chapter 2

CMSF: Constrained Mean Shift Using Distant Yet Related Neighbors

The success of deep neural networks over the past decade has largely depended on supervised learning. However, as models become larger and more complex, their need for labeled data increases dramatically. Creating high-quality labeled datasets is expensive, time-consuming, and often limited by human effort. Reducing this dependence on labeled data by enabling models to learn meaningful representations directly from unlabeled data can therefore have a major impact. Motivated by this challenge, we developed CMSF, a method that learns strong and transferable features from unlabeled data while keeping computational requirements low.

Although self-supervised learning (SSL) methods vary in their loss designs, a shared driver of performance is how diverse the positive and negative pairs are. Without human labels, these pairs provide the sole supervisory signal, so many approaches focus on constructing them more effectively by enriching augmentations to diversify positives [110, 152, 177], by mining harder examples [86, 90, 188], or by adding extra positives via nearest-neighbor (NN) search [6, 45, 98]. MSF [98], for example, improves over many SSL baselines by pulling a query not only toward a second view of itself but also toward the top-k NN views from other images.

However, top-k neighbors are near the query by design, which limits the diversity of positives. Since positive quality is the only supervision available in SSL, we aim to expand diversity by collecting

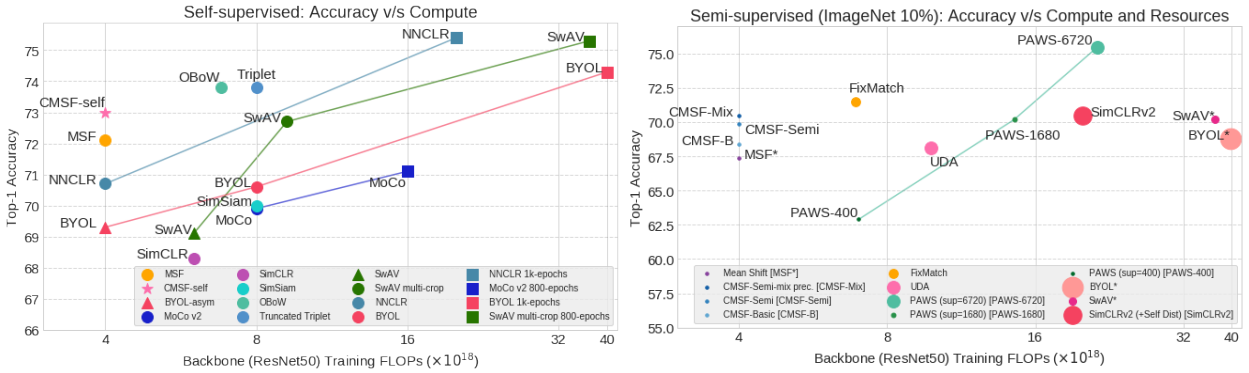


Figure 2.1: **Accuracy vs. training compute on ImageNet with ResNet50:** We report the total training FLOPs for forward and backward passes through the CNN backbone. **(Left) Self-supervised:** All methods are for 200 epochs. $\text{CMSF}_{\text{self}}$ achieves competitive accuracy with considerably lower compute. **(Right) Semi-supervised:** Circle radius is proportional to the number of GPUs/TPUs used. The results are on ImageNet with 10% labels. In addition to being compute efficient, CMSF is trained with an order of magnitude lower resources, making it more practical and accessible. * methods use self-supervised pre-training and finetuning on the labeled set.

positives that are farther in embedding space yet still semantically relevant.

A naïve way to reach farther positives is to increase k in NN search, but doing so lowers the "purity" of retrieved neighbors, the fraction that share the query's category, indicating more semantically irrelevant images. We address this by constraining the NN search to a set of images likely to be relevant, leading to Constrained Mean Shift (CMSF). In a fully supervised setting, we can train with an MSF-style loss but restrict neighbors to images in the same class, which also offers robustness to label noise compared with direct cross-entropy supervision. This idea extends naturally to semi-supervised learning by using pseudo-labels to define the constraint set.

For a purely self-supervised constraint, we limit the query's search space to the NNs of an additional augmentation of the same image. Because this extra view differs in appearance, its neighbors tend to be different and typically farther, while remaining semantically tied to the query. A straightforward implementation would generate and forward this extra view for every image, but we can make it efficient by reusing views computed in the previous epoch.

Empirically, CMSF surpasses strong baselines across supervised, semi-supervised, and self-supervised regimes, matching or reducing training compute. It improves over MSF [108] in SSL, outperforms cross-entropy supervision under clean and noisy labels, and exceeds PAWS [15] in semi-supervised

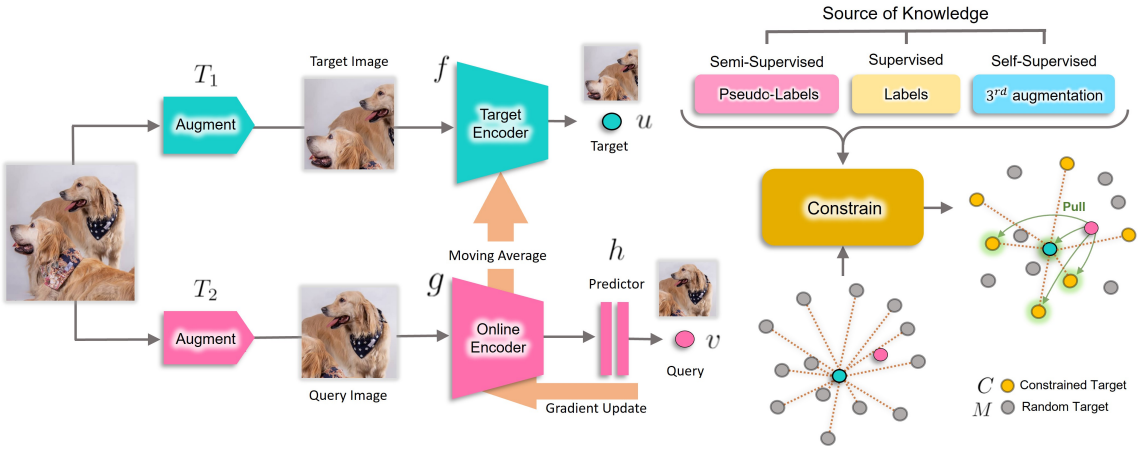


Figure 2.2: **Our method (CMSF):** We augment an image twice and pass them through online and target encoders followed by ℓ_2 normalization to get u and v. Mean-shift [98] encourages v to be close to both u and its nearest neighbors (NN). To make NNs diverse, we constrain the NN search space based on additional knowledge in the form of NNs of the previous augmentation in self-supervised setting or the labels or pseudo-labels in semi or fully supervised settings. These constraints encourages the query to be pulled towards semantically related NNs that are farther away from the target embedding. See Fig 2.3 for constructing the constrained set.

settings. Our core contribution is a simple, general mechanism for retrieving far yet relevant positives and applying it consistently from self to semi and fully supervised training. To summarize,

1. CMSF is a generalization of MSF that restricts NN search to relevance-aware constraint sets.
2. We instantiate constraint-set selection for self, semi, and fully supervised settings, yielding positives that are empirically farther in embedding space while remaining semantically aligned, thus providing a stronger signal than MSF.
3. With comparable compute, CMSF achieves non-trivial gains over self-supervised MSF and its semi-supervised extension, and sets state-of-the-art results in self and semi-supervised settings.

Author contributions. This work was conducted in collaboration with K. L. Navaneet, S. A. Koohpayegani, A. Tejankar, A. Subramanya, and H. Pirsiavash. My contributions focused on the experimental evaluation of the proposed method, including training and evaluating the models on the CIFAR-10 and CIFAR-100 benchmarks, conducting ablation studies, analyzing the results, and preparing the figures for the manuscript. Although these experiments were not included in the published paper, they contributed to the development and validation of the proposed method.

2.1 Related Work

Self-supervised learning (SSL): Earlier works on SSL focused on solving a pretext task that does not require additional labeling. Examples of pretext tasks include colorization [221], jigsaw puzzle [139], counting [140], and rotation prediction [60]. Another class of SSL methods is based on instance discrimination [43]. The idea is to classify each image as its own class. Some methods adopt the idea of contrastive learning for instance discrimination [75, 27, 21, 22, 23]. BYOL [65] proposes a non-contrastive approach by removing the negative set and simply regressing one view of an image from another.

Several recent works aim to find a larger positive sample set to improve learning. In LA [227], samples are clustered using k-means and samples within a cluster are brought closer together compared to cross-cluster samples. MSF [98] and MYOW [6] generalize BYOL by regressing target view and its NNs. NNCLR [45] extends SimCLR to use NNs as positives. CLD [195] integrates grouping using instance-group discrimination. Affinity diffusion [85] uses strongly connected nodes in a graph constructed using embeddings to find positive samples. Unlike these methods, we focus on grouping together far away neighbors that are semantically similar. We show quantitatively and qualitatively the diversity and purity of retrieved neighbors and improved performance over MSF. We generalize the idea in MSF [98] to use an additional source of knowledge to constrain the NN search space for the target view. CoCLR [72] and Cl-InfoNCE [181] also use additional information sources in the form of additional modality and auxiliary labels respectively to improve performance. However, we focus on self- and semi-supervised classification settings and design methods to obtain and use the additional information as a constraint in NN search space.

Supervised learning: A drawback of Cross-entropy is its lack of robustness to noisy labels [222, 168]. [170, 134, 180, 210] address the issue of hard labeling (one-hot labels) with label smoothing, [81, 9, 54] replace hard labels with prediction of pre-trained teacher, and [218, 216] propose an augmentation strategy to train on combination of instances and their labels. Another line of work [62, 156] is to learn representations with good kNN performance. SupCon [92] and [206] improve upon [62] by changing the distance to inner product on ℓ_2 normalized embeddings. We include the supervised setting to better understand the effect of using constrained NNs, particularly

in the noisy label setting.

Semi-supervised learning: Several methods combine self-supervised and supervised learning to form semi-supervised methods. S4L [217] uses rotation prediction based loss on the unlabeled set along with cross-entropy loss on the labeled set. Similarly, SuNCEt [4] combines SimCLR [27] and SwAV [22] methods with supervised contrastive loss. Pseudo-labeling is frequently used in semi-supervised learning. In Pseudo-Label [104], the network is trained with cross-entropy loss using supervised data on the labeled examples and pseudo-labels on the unlabeled ones. In SimCLR-v2 [28], a teacher network is pre-trained using SimCLR [27] and fine-tuned with supervised labels. The teacher is then distilled to a student network using pseudo-labels on the unlabeled set. FixMatch [164] uses pseudo-labels obtained using a weakly augmented image to train a strongly augmented version of the same image. UDA [208] leverages strong data augmentation techniques in enforcing this consistency in pseudo-labels across augmentations. MPL [145] optimizes a student network using pseudo-labels from a teacher network, while the teacher is optimized to maximize the student's performance on the labeled set. PAWS [5] uses consistency based loss on soft pseudo-labels obtained in a non-parametric manner. Our method too uses pseudo-labels to train the unlabeled samples. However, we use the labels as a constraint in MSF [98] and do not directly optimize samples using cross-entropy loss.

Metric learning: The goal of metric learning is to train a representation that puts two instances close in the embedding space if they are semantically close. Two important methods in metric learning are: triplet loss [35, 199, 157] and contrastive loss [163, 20]. Metric learning methods perform well on tasks like image retrieval [204] and few-shot learning [185, 162]. Prototypical networks [162] is similar to a contrastive version of our method with top-*all*.

2.2 Method

Similar to MSF [98], given a query image, we are interested in pulling its embedding closer to the mean of the embeddings of its nearest neighbors (NNs). However, since top NNs are close to the target itself, they may not provide a strong supervision signal. On the other hand, far away (non-top) NNs may not be semantically similar to the target image. Hence, we constrain

the NN search space to include mostly far away points with high purity. The purity is defined as the percentage of the selected NNs being from the same ground truth category as the query image. We use different constraint selection techniques to analyze our method in supervised, self- and semi-supervised settings.

Following MSF and BYOL, we use two embedding networks: a target encoder $f(.)$ with parameters θ_f and an online encoder $g(.)$ with parameters θ_g. The online encoder is directly updated using backpropagation while the target encoder is updated as a slowly moving average of the online encoder: $\theta_f \leftarrow m\theta_f + (1-m)\theta_g$ where m is close to 1. We add a predictor head $h(.)$ [65] to the end of the online encoder so that pulling the embeddings together encourages one embedding to be predictable by the other one and not necessarily encouraging the two embeddings to be equal. In the experiments, we use a two-layer MLP for $h(.)$.

Given a query image x_i, we augment it twice with transformations $T_1(.)$ and $T_2(.)$, feed them to the two encoders, and normalize them with their ℓ_2 norm to get $u_i = \frac{f(T_1(x_i))}{||f(T_1(x_i))||_2}$ and $v_i = \frac{h(g(T_2(x_i)))}{||h(g(T_2(x_i)))||_2}$. We add u_i to the memory bank M and remove the oldest entries to maintain a fixed size M. We select the constraint set C_i as a subset of M. Constraint set selection is explained in detail in Sections 2.2.1, 2.2.2, and 2.2.3. We then find the set S_i of top-k nearest neighbors of u_i in C_i including u_i itself. Finally, we update $g(.)$ by minimizing:

$$L = \sum_{i=1}^{n} \frac{1}{|S_i|} \sum_{z \in S_i} v_i^T z$$

where n is the size of mini-batch and $|S_i|$ is the size of set S_i, e.g., k in top-k. Finally, we update $f(.)$ with the momentum update. In the top-*all* variation of our method, number of neighbors k is set equal to the size of C_i, i.e., $S_i = C_i$. Note that since u_i itself is included in the nearest neighbor search, the method will be identical to BYOL [65] when $k = 1$ and to self-supervised mean-shift [98] when the constraint is fully relaxed ($C_i = M$). Our method covers a larger spectrum of algorithms by defining the constrained set. Below we discuss the selection of constrained set in various settings.

2.2.1 Self-Supervised Setting

In addition to M, we maintain a second memory bank M' that is exactly the same as M but contains features from a different (3^{rd}) augmentation of the image x_i fed through target encoder $f(.)$. We

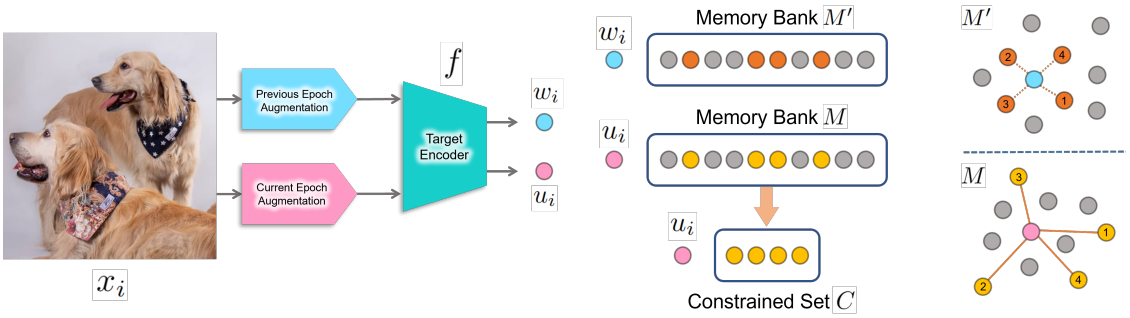


Figure 2.3: **CMSF$_{\text{self}}$:** The indices of the NNs of the previous epoch's memory bank M' are used to construct the constrained set C from the current memory bank M.

assume $w_i \in M'$ and $u_i \in M$ are two embeddings corresponding to the same image x_i. Then, we find NNs of w_i in M' and use their indices to construct the search space C_i from M (See Fig. 2.3). Note that although the NNs of w_i in M' are already close to each other, their corresponding elements in M may not be close to each other since M contains different augmentations u_i of the same images. As a result, C_i will maintain good purity while containing distant NNs (refer to Fig. 2.5).

Since it is expensive to embed a 3rd augmentation of each image, we embed only two augmentations as in MSF and BYOL and cache the embeddings from the previous epoch, keeping the most recent embedding for each image. The cached embedding will be still valid after one epoch since the target encoder is updated slowly using the momentum update rule (similar to MoCo). Since cache size is equal to the dataset size, we store it in the CPU memory and maintain the auxiliary memory bank M' by loading the corresponding part of it to the GPU memory for each mini-batch. Caching of features is not essential for CMSF to work and is only used to reduce computational cost. We performed experiments with an actual 3rd augmentation instead and found the results to be similar to our method except that it was nearly 30% slower due to forwarding an additional augmentation.

In the intermediate stages of learning, the top elements of C_i are spread apart in M with higher median ranks, and get closer to the top elements of M as the learning progresses. Note that we use w_i instead of u_i in finding the NNs in M' since both w_i and M' use an older target model, so are more comparable.

Since CMSF adds farther NNs only for stronger supervision, we additionally employ MSF loss calculated on the unconstrained M. Then, in the self-supervised setting, the total loss is an equally

weighted sum of MSF and CMSF losses.

2.2.2 Supervised Setting

While supervised setting is not our primary novelty or motivation, we study it to provide more insights into our constrained mean-shift framework. With access to the labels of each image, we can simply construct C_i as the subset of M that shares the same label as the query x_i. This guarantees 100% purity for NNs.

Note that most supervised methods, including cross-entropy loss, try to group all examples of a category together on the same side of a hyper-plane while remaining categories are on the other side. However, our method pulls the target to be close to only those examples of the same category that are already close to the target. This results in a supervised algorithm that may keep the latent structure of each category which can be useful for pre-training on coarse-grained labels. Moreover, as shown in the experiments (Fig. 2.6), our method is more robust to label noise since most mis-labeled images will be far from the target embedding, thus ignored in learning. This motivates applying our method to semi-supervised setting where the limited supervision provides noisy labels.

2.2.3 Semi-Supervised Setting

In this setting, we assume access to a dataset with a small labeled and a large unlabeled subset. We train a simple classifier using the current embeddings of the labeled data and use the classifier to pseudo-label the unlabeled data. Then, similar to the supervised setting, we construct C_i to be the elements of M that share the pseudo-label with the target embedding. Again, this method increases the diversity of C_i while maintaining high purity. To keep the purity high, we enforce the constraint only when the pseudo-label is very confident (the probability is above a threshold.) For the samples with non-confident pseudo-label, we relax the constraint resulting in regular MSF loss (*i.e.,* $C_i = M$.) Moreover to reduce the computational overhead of pseudo-labeling, we cache the embeddings of labeled examples throughout the epoch and train a 2-layer MLP classifier using the frozen cached features and their groundtruth labels in the middle and end of each epoch.

2.3 Experiments

Implementation details: We use PyTorch for all our experiments. Unless specified, we use the same hyper-parameter values in self-, semi- and fully supervised settings. All models are trained on ImageNet-1k (IN-1k) for 200 epochs with ResNet-50 [76] backbone and SGD optimizer (learning rate=0.05, batch size=256, momentum=0.9, and weight decay=1e-4) with cosine scheduling for learning rate. The momentum value of CMSF for the moving average key encoder is 0.99. The 2-layer MLP architecture for $\text{CMSF}_{\text{semi}}$ is as follows: (linear (2048×4096), batch norm, ReLU, linear (4096×512)). The default memory bank size is 128k. Top-k is set to 10 in the semi- and fully supervised settings and 5 in the self-supervised setting.

Our main CMSF experiment with 200 epochs takes nearly 6 days on four NVIDIA-2080TI GPUs. The overhead in training time due to NN search is negligible compared to the forward and backward passes through the network (that is also done in BYOL): the increase in time is 0.7% for MSF [98] and 2.1% for $\text{CMSF}_{\text{self}}$.

Recent SSL methods are usually computationally expensive leading to worse environmental impact and exclusion of smaller research labs. While our experiments are more efficient and accessible than most SOTA methods, e.g., PAWS, we limit our training length to 200 epochs due to resource constraints. We do not empirically verify whether the improvements observed over SOTA approaches at lower epochs (200) are persistent with longer training (e.g., 800 or 1000 epochs).

Evaluation: We evaluate the pre-trained models using linear evaluation (*Linear IN-1k*) in both ImageNet classification and transfer settings. The model backbone parameters are fixed and a single linear layer is trained atop them following the setting in CompRess [3]. Additionally, we report k-nearest neighbor ($k = 1, 20$) evaluation for the SSL setting as in [3]. The transfer performance is evaluated on the following datasets: Food101 [18], SUN397 [207], CIFAR10 [101], CIFAR100 [101], Cars196 [99], Aircraft [122], Flowers (Flwrs102) [137], Pets [142], Caltech-101 (Calt101) [49], and DTD [36].

Table 2.1: **Evaluation on full ImageNet:** We compare our model with other SOTA methods in Linear (Top-1 Linear) and Nearest Neighbor (1-NN,20-NN) evaluation. We use a memory bank of size 128K for CMSF and provide comparison with both 256K and 1M memory bank versions of MSF. Since $\text{CMSF}_{\text{self}}$ uses NNs from two memory banks, it is comparable to MSF (256K) in memory and computation overhead. Our method outperforms other SOTA methods with similar compute including MSF. "Multi-Crop" refers to use of more than 2 augmentations per image during training (e.g., OBoW uses $2 \times 160 + 5 \times 96$ resolution images in both forward and backward passes compared to a single 224 in $\text{CMSF}_{\text{self}}$). Multi-crops significantly increases compute while symmetric loss doubles the computation. Thus methods employing these strategies are not directly comparable with $\text{CMSF}_{\text{self}}$.

Method	Ref.	Batch Size	Epochs	Sym. Loss 2x FLOPS	Multi-Crop Training	Top-1 Linear	NN	20-NN
Supervised	[2]	256	100	-	-	76.2	71.4	74.8
Random-init	-	-	-	-	-	5.1	1.5	2.0
SeLa-v2 [213]	[22]	4096	400	✓	✗	67.2	-	-
SimCLR[27]	[27]	4096	1000	✓	✗	69.3	-	-
SwAV [22]	[22]	4096	400	✓	✗	70.1	-	-
DeepCluster-v2 [21]	[22]	4096	400	✓	✗	70.2	-	-
SimSiam [31]	[31]	256	400	✓	✗	70.8	-	-
MoCo v2 [75]	[30]	256	800	✗	✗	71.1	57.3	61.0
CompRess [3]	[3]	256	1K+130	✗	✗	71.9	63.3	66.8
InvP	[187]	256	800	✗	✗	71.3	-	-
BYOL [65]	[65]	4096	1000	✓	✗	74.3	62.8	66.9
SwAV [22]	[22]	4096	800	✓	✓	75.3	-	-
NNCLR	[45]	4096	1000	✗	✗	**75.4**	-	-
SimCLR[27]	[31]	4096	200	✓	✗	68.3	-	-
SwAV [22]	[31]	4096	200	✓	✗	69.1	-	-
MoCo v2 [75]	[31]	256	200	✓	✗	69.9	-	-
SimSiam [31]	[31]	256	200	✓	✗	70.0	-	-
NNCLR[45]	[45]	4096	200	✗	✗	70.7	-	-
BYOL [65]	[31]	4096	200	✓	✗	70.6	-	-
SwAV [22]	[31]	256	200	✓	✓	72.7	-	-
Truncated Triplet [188]	[188]	832	200	✓	✗	**73.8**	-	-
OBoW [59]	[59]	256	200	✗	✓	**73.8**	-	-
MoCo v2 [75]	[30]	256	200	✗	✗	67.5	50.9	54.3
CO2 [198]	[198]	256	200	✗	✗	68.0	-	-
BYOL-asym [65]	[98]	256	200	✗	✗	69.3	55.0	59.2
ISD [174]	[174]	256	200	✗	✗	69.8	59.2	62.0
MSF (1M) [98]	[98]	256	200	✗	✗	72.4	62.0	64.9
MSF (256K)[98]	[98]	256	200	✗	✗	72.2	62.1	65.1
$\text{CMSF}_{\text{self}}$ (128K)	-	256	200	✗	✗	**73.0**	**63.2**	**66.4**

2.3.1 Self-Supervised Learning ($\text{CMSF}_{\text{self}}$)

To reduce the GPU memory footprint, we cache the previous augmentation embedding of each sample in the dataset in the CPU. The cached features corresponding to the current mini-batch are retrieved from CPU memory to maintain memory bank M' with previous augmentations. This cache is updated using the oldest features in M that we remove from M after each iteration.

Results on ImageNet: Results of $\text{CMSF}_{\text{self}}$ are shown in Table 2.1. $\text{CMSF}_{\text{self}}$ outperforms MSF



Figure 2.4: **Nearest neighbor selection on constrained memory bank:** First row shows top-5 NNs of target in constrained set C and their corresponding rank in the unconstrained memory bank M obtained using an intermediate checkpoint (epoch 100). While they are not the closest samples to the target (higher rank index), they are semantically similar to the target. This shows that the constraint can capture far away samples with similar semantic as the target. The second row depicts images from memory bank with one rank lower than the corresponding image in the first row. These images contain incorrect category retrievals. Distant neighbors cannot be trivially obtained by increasing the number of NNs. Examples are chosen randomly.

baseline with a larger memory bank, which we believe is due to pulling together far yet semantically similar samples (Fig. 2.4). We use MSF with $2x$ larger memory bank for fair comparison. $\text{CMSF}_{\text{self}}$ also achieves state-of-the-art performance on both NN and Linear metrics when compared with approaches with similar computational budget. We compare our method to other state-of-the-art approaches with 200 epochs of training in Fig. 2.1. We observe a good trade-off in terms of accuracy and compute for $\text{CMSF}_{\text{self}}$. Note that we train without symmetrical loss and multi-crop strategy which are known to generally improve performance [22] at the cost of significant increase in compute.

Evaluation on ImageNet subsets: Following [79, 27], we evaluate the pre-trained models on the ImageNet classification task with limited labels. We report results with 1% and 10% labeled subsets of ImageNet (Table 2.4). $\text{CMSF}_{\text{self}}$ outperforms MSF on top-1 accuracy in both 1% and 10% settings and is comparable to existing approaches that require significantly higher training time.

Transfer learning: We follow the procedure in [65, 27] for transfer evaluation (refer to Table 2.2). Hyperparameters for each dataset are tuned independently based on the validation set accuracy and final accuracy is reported on the held-out test set. $\text{CMSF}_{\text{self}}$ achieves SOTA average performance among methods trained for 200 epochs.

Purity of constrained samples: In $\text{CMSF}_{\text{self}}$, we depend on information from previous augmentations to constrain NN search in the current memory bank. Our goal is to improve learning by using distant samples with a good purity. As shown in Fig. 2.5, we observe that the top-k samples from constrained memory bank C have higher rank in M, so are far neighbors of the target. Also, those samples maintain almost the same purity as the top-k samples from unconstrained

Table 2.2: **Transfer learning evaluation:** Our supervised CMSF model at just 200 epochs outperforms all supervised baselines on transfer learning evaluation. Our SSL model outperforms MSF, the comparable state-of-the-art approach, by 1.2 points on average over 10 datasets. We get the results for MoCo v2, MSF, and BYOL-asym from [98], SimCLR and Xent (1000 epoch) from [27], and BYOL from [65].

Method	Epoch	Food 101	CIFAR 10	CIFAR 100	SUN 397	Cars 196	Air-craft	DTD	Pets	Calt. 101	Flwr 102	**Mean Trans**	**Linear IN-1k**
Supervised Models													
Xent	200	67.7	89.8	72.5	57.5	43.7	39.8	67.9	91.8	91.1	88.0	71.0	77.2
Xent	90	72.8	91.0	74.0	59.5	56.8	48.4	70.7	92.0	90.8	93.0	74.9	76.2
ProtoNW	200	73.3	93.2	78.3	61.5	65.0	57.6	73.7	92.2	94.3	93.7	78.3	76.0
SupCon	200	72.5	93.8	77.7	61.5	64.8	58.6	74.6	**92.5**	93.6	94.1	78.4	**77.5**
Xent	1000	72.3	93.6	78.3	61.9	66.7	61.0	**74.9**	91.5	94.5	94.7	78.9	76.3
CMSF_{sup} top-*all*	200	73.7	94.2	**78.7**	62.1	**71.7**	**64.1**	73.4	**92.5**	94.5	**95.8**	**80.1**	75.7
CMSF_{sup} top-10	200	**74.9**	**94.4**	**78.7**	**62.7**	70.8	63.4	73.8	92.2	**94.9**	95.6	**80.1**	76.4
Self-Supervised Models													
SimCLR	1000	72.8	90.5	74.4	60.6	49.3	49.8	**75.7**	84.6	89.3	92.6	74.0	69.3
MoCo v2	800	72.5	**92.2**	74.6	59.6	50.5	53.2	74.4	84.6	90.0	90.5	74.2	71.1
BYOL	1000	**75.3**	91.3	**78.4**	**62.2**	**67.8**	**60.6**	75.5	**90.4**	**94.2**	**96.1**	**79.2**	**74.3**
MoCo v2	200	70.4	91.0	73.5	57.5	47.7	51.2	73.9	81.3	88.7	91.1	72.6	67.5
BYOL-asym	200	70.2	91.5	74.2	59.0	54.0	52.1	73.4	86.2	90.4	92.1	74.3	69.3
MSF	200	72.3	**92.7**	76.3	60.2	59.4	56.3	71.7	89.8	90.9	93.7	76.3	72.1
$\text{CMSF}_{\text{self}}$	200	**73.0**	92.2	**77.2**	**61.0**	**60.6**	**58.4**	**74.1**	**91.1**	**92.0**	**94.5**	**77.4**	**73.0**

memory bank M. As a result, C maintains good purity while being diverse.

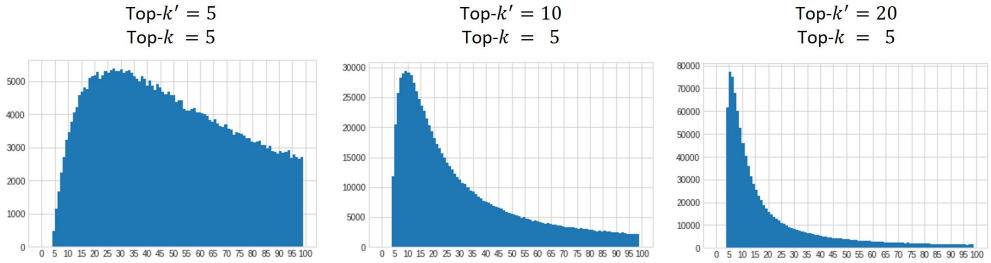
Effect of k': In $\text{CMSF}_{\text{self}}$, we first calculate top-k' samples (the first k' NNs of the target) from the secondary memory bank M'. We then use those indices to constrain NN search space in the primary memory bank M and select top-k for optimization. We varied the value of k' in $\text{CMSF}_{\text{self}}$ to explore its effect, keeping k fixed to 5. We observe that increasing k' (relaxing the constraint) will decrease the accuracy of the model. As observed in Table 2.3-right, the overlap between constrained and unconstrained NN set increases with increasing value of k'. Note that in a case where $k' = \infty$, $\text{CMSF}_{\text{self}}$ will be identical to the MSF baseline.

2.3.2 Supervised Learning

Evaluation: Unlike cross-entropy (Xent [15, 113, 155]) baseline, SupCon [92], ProtoNW [162] and CMSF do not train a linear classifier during the pre-training stage. Thus, we use the pre-training dataset ImageNet-1k (IN-1k) for linear evaluation of the frozen features as done in SSL. For Xent, we use the linear classifier trained during pre-training. We use the same settings and datasets as self-supervised for transfer learning evaluation.

Table 2.3: **Effect of k' in sampling NN from M':** In $\text{CMSF}_{\text{self}}$, we constrain top-$k$ NN search space in M with top-k' samples from M'. (**Top**) Increasing k' results in a drop in accuracy. The k in top-k is set to 5 for all values of k'. (**Bottom**) Histogram of the constrained sample ranks at epoch 50. The histogram shifts left, i.e., overlap between constrained and unconstrained NN sets increases with increasing value of k'.

k'	5	10	20	40	80
NN	63.2	62.9	62.7	62.3	61.7
20-NN	66.4	66.1	65.9	65.6	65.0



Results: Results on IN-1k dataset are shown in Table 2.2. In top-*all* variation of our method, k is equal to the total size of C. SSL inspired methods like CMSF and SupCon significantly outperform Xent when trained for similar number of epochs. We observe that improvements in ImageNet performance do not always translate to transfer performance. Interestingly, CMSF performs the best on transfer evaluation, particularly on fine-grained datasets like Cars196 and Aircraft. We believe that the absence of explicit cross-entropy based optimization using the supervised labels preserves the multi-modal distribution of categories improving fine-grained performance. Supervised CMSF uses class labels only as a constraint for MSF during pre-training and does not explicitly optimize on the classification task. Superior performance of CMSF_{sup} top-10 demonstrates the importance of using distant yet semantically related neighbors as positives.

Noisy Labels: In the noisy setting, we use random i.i.d. noise to corrupt the labels (change the label randomly) of a percentage of images. We consider, 5%, 10%, 25%, and 50% label corruption (noise) rates. For faster experiments, we report results on the ImageNet-100 dataset [176] (Fig. 2.6). We observe a significantly higher degradation in performance of Xent baseline and CMSF_{sup} top-*all* compared to CMSF_{sup} top-10 at high noise levels. The gap between the approaches is larger on transfer learning. These observations indicate that NN based methods like CMSF are better suited for noisy constraint settings compared to approaches utilizing all samples of a class as positives.

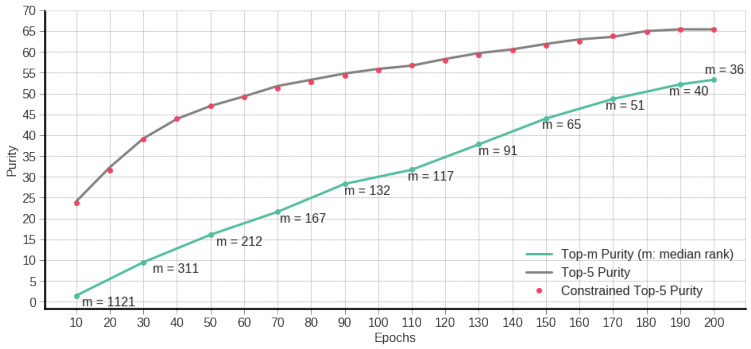


Figure 2.5: **Purity of constrained samples:** During training of $\text{CMSF}_{\text{self}}$, we plot purity of the top-5 samples in unconstrained set M (in black) and that of the top-5 samples in constrained set C (in red). The red curve is not significantly below the black one suggesting that the purity is not dropped by increasing the distance of the NNs. To show that elements in C may be far from the target u, we choose the 5^{th} element in C and find its rank in the set M. We calculate the median of this rank as m. The purity of the top-m elements of set M (green curve) is consistently lower than that of top-5 elements of the constrained set C (red curve). This suggests that one cannot maintain high purity by simply considering more NNs using a larger k.

This robustness to label noise motivates our application of CMSF to self- and semi-supervised settings where pseudo-labels or the NNs of previous augmentations may be noisy.

Coarse-grained ImageNet: CMSF groups together only top-k neighbors and thus can help in preserving the latent structure of the data compared to top-*all*. To verify this, we consider a dataset with coarse-grained labels where this difference is pronounced. Based on the WordNet hierarchy, we merge each category in the ImageNet dataset to its parent class. We further ensure that no two classes are in the same path in the graph by merging the descendant into the ancestor class. The total number of classes is thus reduced from 1000 in ImageNet-1k to 93 in our ImageNet-coarse. We train CMSF and the baseline approaches in a supervised manner using the coarse labels and then evaluate on the fine-grained / original labels on ImageNet-1k validation set. In Table 2.5 we compare the top-*all* and top-k variants on the coarse grained version of ImageNet. CMSF_{sup} top-k sees a minor drop in performance compared to training on ImageNet-1k. However, methods in which all samples in a class are explicitly brought closer - CMSF_{sup} top-*all*, cross-entropy and supervised contrastive - see a huge drop in accuracy. More details on coarse-grained ImageNet are in the supplementary.

Table 2.4: **Evaluation on small labeled ImageNet** : We compare our model to MSF and other baselines on ImageNet 1% and 10% linear evaluation benchmarks. "Fine-tuned" refers to fine-tuning the entire backbone network instead of a single linear layer. $\text{CMSF}_{\text{self}}$ outperforms MSF on top-1 metric in both 1% and 10% settings.

Method	Fine-tuned	Epochs	Top-1		Top-5	
			1%	10%	1%	10%
Supervised	✓		25.4	56.4	48.4	80.4
PIRL [131]	✓	800	-	-	57.2	83.8
CO2 [198]	✓	200	-	-	71.0	85.7
SimCLR [27]	✓	1000	48.3	65.6	75.5	87.8
InvP [187]	✓	800	-	-	78.2	88.7
BYOL [65]	✓	1000	53.2	68.8	78.4	89.0
SwAV [22]	✓	800	**53.9**	**70.2**	**78.5**	**89.9**
MoCo v2 [30]	✗	800	51.5	63.6	77.6	86.1
BYOL [65]	✗	1000	55.7	**68.6**	80.0	**88.6**
CompRess [3]	✗	1K+130	**59.7**	67.0	**82.3**	87.5
MoCo v2 [30]	✗	200	43.6	58.4	71.2	82.9
BYOL-asym	✗	200	47.9	61.3	74.6	84.7
ISD [174]	✗	200	53.4	63.0	78.8	85.9
MSF [98]	✗	200	55.5	66.5	**79.9**	87.6
$\text{CMSF}_{\text{self}}$	✗	200	**56.4**	**67.5**	79.8	**87.7**

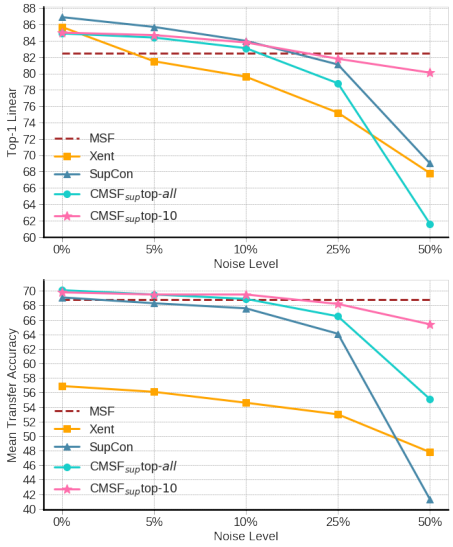


Figure 2.6: **Noisy supervised setting on ImageNet-100:** Our method is more robust to noisy annotation compared to Xent and SupCon. Also, using top-*all* degrades the results since all images from a single category are not guaranteed to be semantically related due to noisy labels. Mean Transfer Accuracy is the average over 10 transfer datasets.

Table 2.5: **Supervised learning on coarse grained ImageNet:** We train on the coarse grained version of ImageNet (93 super categories) and perform linear evaluation on the original ImageNet-1k validation set with fine-grained labels (1000 categories).

Train Dataset	ImageNet-1k Validation Set			
	Xent	SupCon	CMSF_{sup} top-*all*	CMSF_{sup} top-10
ImageNet-1k	77.2	**77.5**	75.7	76.4
ImageNet-coarse	61.4	58.7	67.0	**74.2**

2.3.3 Semi-Supervised Learning

Implementation Details: We train a 2-layer MLP atop the cached target features of supervised set for pseudo-labeling. The pseudo-label training is performed twice per epoch (takes 40 seconds per training) and the label assignment is done in an online fashion for each mini-batch. The confidence threshold for pseudo-labeling is set to 0.85. We use the same optimizer settings as in self-supervised CMSF for the pre-training stage. Similar to S4L [217], we perform two stages of fine-tuning with supervised and pseudo-labels. We fine-tune the backbone network with two MLPs (as in PAWS [5]) on the 10% labeled set for 20 epochs and pseudo-label the train set. Samples above confidence

Table 2.6: **Semi-supervised learning on ImageNet dataset with 10% labels:** FLOPs denotes the total number of FLOPS for forward and backward passes through ResNet-50 backbone while batch size denotes the sum of labeled and unlabeled samples in a batch. $\text{CMSF}_{\text{semi}}$-mix precision is compute and resource efficient, achieving SOTA performance at comparable compute. PAWS requires large number of GPUs to be compute efficient and its performance drastically drops with 4/8 GPUs. † Trained with stronger augmentations like RandAugment [38]. * TPUs are used.

Method	Epochs	Batch Size	GPUs	FLOPs ($\text{x}10^{18}$)	Top-1
Self-supervised Pre-training					
Mean Shift [98]	200	256	4	4	67.4
BYOL [65]	1000	4096	512*	40	68.8
SwAV [22]	800	4096	64	37	70.2
SimCLRv2 [28]	800	4096	128*	16	68.4
Semi-supervised Pre-training					
SimCLRv2 (+Self Dist) [28]	1200	4096	128*	20	70.5
UDA† [208]	800	15872	64*	10	68.1
FixMatch† [164]	300	6144	32*	7	71.5
MPL† [145]	800	2048	-	30	73.9
PAWS (support=6720) [5]	300	4096	64	21	75.5
PAWS (support=1680) [5]	100	256	8	15	70.2
PAWS (support=400) [5]	100	256	4	7	62.9
$\text{CMSF}_{\text{semi}}$-basic	200	256	4	4	68.6
$\text{CMSF}_{\text{semi}}$	200	256	4	4	69.9
$\text{CMSF}_{\text{semi}}$-mix precision	200	768	4	4	70.5

threshold (nearly 30% of dataset) are combined with supervised set to fine-tune again for 20 epochs (more details in suppl.). The second fine-tuning is equivalent to 5 epochs with full data and is a small increase in our total compute. This is needed since we do not directly optimize cross-entropy loss in pre-training as in [164, 208, 145].

Evaluation: The final epoch parameters are used to perform evaluation. We report top-1 accuracy on the ImageNet validation set. We additionally report the total number of FLOPs for forward and backward passes (backward is 2× forward) through ResNet-50 backbone and the number of GPUs/TPUs used by each method in the pre-training stage (more details in suppl.).

Baselines: We compare the proposed approach ($CMSF_{semi}$) with self- and semi-supervised approaches. $CMSF_{semi}$-*basic* minimizes unconstrained MSF loss on the unlabeled examples (no pseudo-labeling) and CMSF loss on the labeled examples only. We provide comparison of PAWS method with different support set sizes. We train PAWS on 4x 16GB GPUs with maximum possible support set size (200 classes, 2 images/class) using code provided by the authors. We also report results using mixed precision training ($CMSF_{semi}$-*mix precision*) as in PAWS [5] with a higher batch

size of 768 since it has lower memory requirement.

Results: $\text{CMSF}_{\text{semi}}$-mix precision achieves comparable performance to most methods with less training and without the use of stronger augmentation schemes like Rand-Augment [38] (Table 2.6, Fig. 2.1). PAWS with a support set size of 6720 outperforms other approaches. However, this requires higher compute (4.8× FLOPs) and resources (64 GPUs) compared to $\text{CMSF}_{\text{semi}}$-mix precision (4 GPUs). Since PAWS requires a large support set, it does not scale well to lower resource (4/8 GPUs) settings even if the total compute remains the same. When trained on only 4 GPUs, CMSF outperforms PAWS by **7.6%** points.

2.4 Conclusion

MSF is a recent SSL method that pulls an image towards its nearest neighbors. We argue that the model can benefit from more diverse yet pure neighbors. Hence, we generalize MSF method by constraining the NN search. This opens the door to using the mean-shift idea to various settings of self-supervised, supervised, and semi-supervised. To construct the constraint, our SSL method uses cached augmentations from the previous epoch while the supervised and semi-supervised settings use labels or pseudo-labels. We show that our method outperforms SOTA approaches like MSF in SSL, PAWS in semi-supervised, and supervised contrastive in transfer-learning evaluation of supervised settings.

Chapter 3

PRANC: Pseudo RAndom Networks for Compacting deep models

In the previous chapter, we discussed efficiency of supervision and showed that with CMSF, it is possible to reduce dependence on labeled data by enabling models to learn useful and transferable representations directly from unlabeled data while keeping computation low. In this chapter, we focus on the efficiency of parameters and address a major bottleneck in both training and deployment, the growing cost of GPU memory and communication as model sizes increase. To tackle this challenge, we propose PRANC, a method that compresses neural networks by reducing the number of trainable parameters, thereby lowering memory usage and communication overhead without sacrificing performance.

Many artificial intelligence and machine learning applications would benefit from the efficient communication and storage of deep models. However, modern deep neural networks often have millions to billions of parameters, making model communication costly or even infeasible. The problem worsens when the agents need to communicate in environments with low bit-rate constraints. Communication constraints could emerge from the physics of the environment, for instance, underwater or underground communications, or caused by an adversary as in communications-denied environments. The long-range communication bandwidth for these applications could be as low as 100 bits per second. In such a low bit-rate environment, transferring a relatively small network

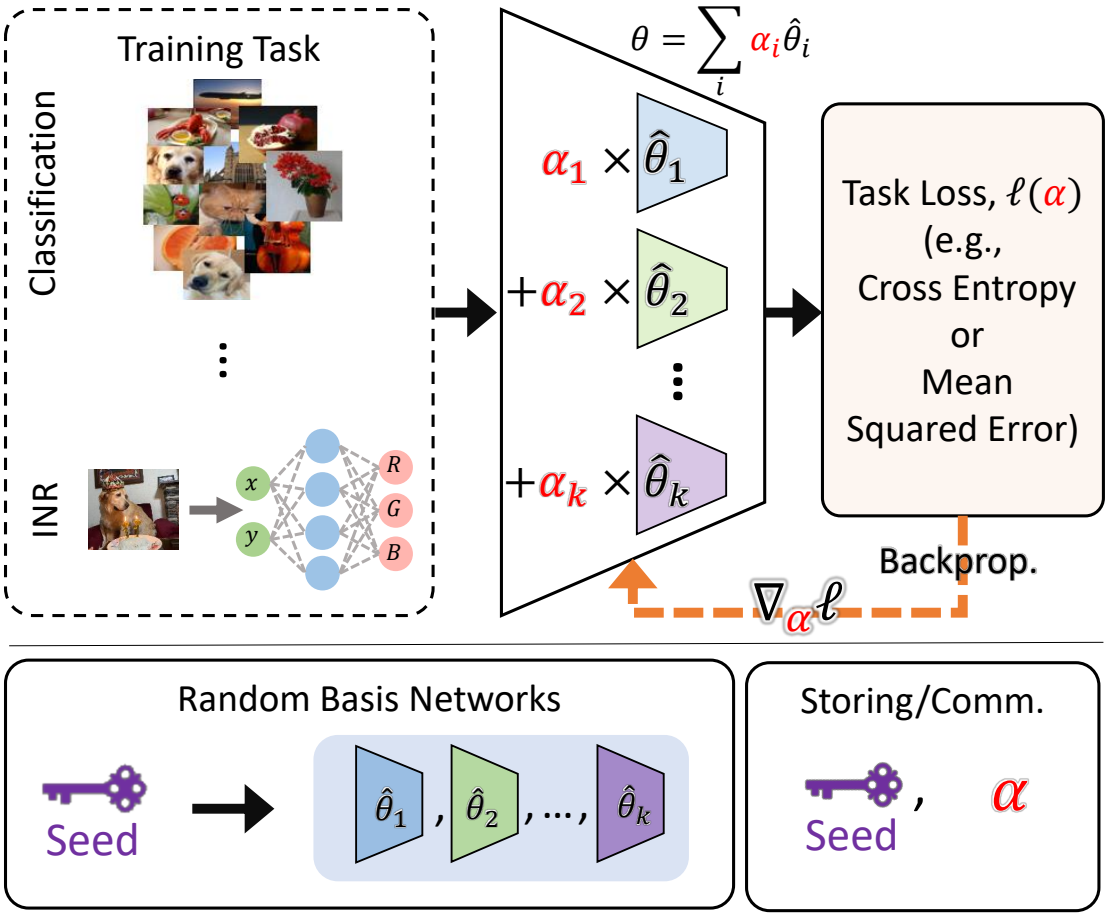


Figure 3.1: We restrict the deep model to be a linear combination of k randomly initialized models. Since the number of models is much less than the size of the model, it is much less expensive to communicate or store the coefficients compared to the model or data itself. We tune α to minimize the loss of the task using standard backpropagation.

like the ResNet18 model with 11M parameters takes more than 40 days. Moreover, in distributed learning applications with a large number of agents, even in high-bandwidth WiFi networks, many peer-to-peer communications may be limited by the congestion in the network (packets from different senders collide with one another).

Going beyond communications, storing these large models on edge devices poses another significant challenge. Edge devices often come with small memories unsuitable for storing large neural networks. Hence, such applications may benefit from compacting a deep model to fewer parameters to construct the model on-demand to run inference.

To solve this problem, one may compact the model by distilling it into a smaller model [81], pruning the model parameters [119], quantizing the parameters [106], or share the weights as much as

possible [166, 29]. Dataset distillation is another alternative method for reducing the dataset size to be used for training the model [194]. However, most of these methods are limited to small reduction factors, e.g., less than 30×. Also, knowledge distillation methods reduce the model architecture to a smaller one with fewer layers, which may limit the future application of that model, e.g., for future fine-tuning or lifelong learning.

We are interested in compacting a deep model by a considerable factor (e.g., 100×) without changing its architecture. The core idea behind our approach is simple. We constrain our model to be a linear combination of a finite set of randomly initialized models, called 'basis' models. Hence, the problem boils down to finding the optimal linear mixture coefficients that result in a network that can solve the task effectively. The model can then be succinctly represented by the seed (a single scalar) to generate the pseudo-random basis models and the linear mixture coefficients (See Figure 3.1). Our method can be seen as a novel reparameter-ization of a deep model without changing its architecture. This enables us to study the effect of increasing the size of the architecture (both depth and width) without changing the number of optimized parameters (see Figure 3.3).

In addition to efficiency, our proposed method provides secure communication and storage, which is of significant interest in applications concerning cybersecurity and privacy. Briefly, our 'basis' models are generated with pseudo-random generators with a specific 'seed.' Theis seed could be privately shared between authenticated entities. Given the minimal self-correlation of pseudo-random sequences, a slight seed change produces a drastically different set of 'basis' models, making the publicly shared linear mixture coefficients useless to unauthorized parties. This design choice facilitates secure communication and storage, especially in cybersecurity or privacy-sensitive applications.

We show that our proposed method is not only suitable for compacting image recognition models, but also it can be used to compress images by compacting their corresponding implicit neural representations (INRs). To do this, we train an INR to reconstruct a single image by overfitting to it while reparameterizing it using our PRANC framework to be compact. To summarize, our simple but effective network reparameterization, denoted as PRANC, is efficient in computation and memory at the learning and reconstruction time. We perform experiments using various datasets and model architectures and show higher accuracy with a much smaller number of parameters

compared to the baselines.

Author contributions. This work was conducted in collaboration with P. Nooralinejad, A. Abbasi, S. A. Koohpayegani, R. M. S. Khan, S. Kolouri, and H. Pirsiavash. My contribution was applying PRANC to image compression by compacting the implicit neural representations associated with images from the NIH Chest X-ray dataset. I conducted the corresponding experiments and evaluated the quality of the compressed reconstructions.

3.1 Related work

Some prior works [147, 124, 32, 55] have shown that randomly initialized networks have a subnetwork that competes with the original network's accuracy. Some recent papers like [203] introduced an application for using this fact in continual learning. Using random networks is appealing from multiple perspectives, for instance, for model communication or for rapid adaptation. In this work, as opposed to finding subnetworks in a randomly generated network (i.e., masking), we seek a linear combination of a relatively small set of randomly generated networks, denoted as *basis* models, that can reconstruct an accurate model. This approach can be used to transmit knowledge with very limited communication (which is one of the important bottlenecks in federated learning [69]), can reduce the storage required for a model (that makes the algorithm useful for embedded devices or continual learners [50]), and finally, can enable on-the-fly weight generation that reduces the memory queries and accelerate the running time of a model. However, our main goal in this paper is to minimize the communication between agents when exchanging knowledge. For that purpose, current methods can be divided into three main categories described below.

Model Compression: Model Compression is not a new topic in the field. HashedNet [29] uses weight grouping with a hash function to reduce the number of learnable parameters. HashedNet can be seen as a specific case of our method where the random models are binary, each random model has an equal number of ones, and each weight of the original model is one only in one of the random models. The original paper experiments with MLP on small datasets. We reproduce it for our setting and show that our method outperforms it. Similar to HashedNet, Weight Fixed Network (WFN) [166] compresses the model by minimizing the entropy and number of unique parameters in

a network. WFN preserves the accuracy of the model with 10× reduction in storage size. Instead of hard-sharing the weights in HasedNet, [182] uses soft sharing. Although all these methods reduce the number of parameters, they all need to keep the location of each element to reconstruct the network. Han etal [71] use pruning, quantization, and Huffman encoding to achieve compression rates generally less than 50×. More recent approaches like MIRACLE [73] and weightless [151] have shown promising results with much higher compression rates (e.g., +400×). However, they use large architectures, e.g., VGG which has 150M parameters, so even after 400×, there are still more than 300k parameters (more than a ResNet-20). In this work, we show that we can reduce the number of required parameters regardless of the network architecture.

Model Pruning and Quantization: Compressing a model can be defined as reducing the number of bytes required to store a deep model. Several papers like XNOR-NET [148] and EWGS [106] use weight/activation (W/A) quantization for reducing the size of a network. Although W/A Quantization has proven to be an effective approach for reducing network size while maintaining accuracy, it is mainly designed for optimizing the computation for network inference. Another approach that is used for compressing a model is pruning the set of less important weights to zero, which reduces the number of floating point operations (FLOPS) and can also reduce the amount of data required to store and communicate a network. These methods include: Neuron Merging [94], Dynamic pruning [119, 159], ChipNet [178], Pruning at initializing [74], Wang et.al. [189], and Collaborative Compression (CC) [118]. Once again, most of these methods use sparsity factors of 20× or less, which is lower than our goal in this paper. We compare our method, PRANC, with existing works that provide extreme compression rates (+99% pruning rate), e.g., DPF[119], STR[102], and SuRP[88]. Lastly, there are some prior work that decomposes model filters as a linear combination of some basis filters [70, 10]. The goal of such methods is to reduce the computation and not necessarily the number of parameters. We focus on extremely small number of parameters that cannot be achieved by such methods.

Data Compression - Core Set: Another approach to recreating an accurate network is to store or communicate its training dataset, and train a network in the target agent. Since most of the datasets are large, methods are proposed to synthesize metadata in the shape of images or obtain a core set of the dataset. These methods include: Dataset Distillation (DD) [194], which regresses

images and learning rate, Flexible Dataset Distillation (FDD) [17], which regresses pseudo-labels for real images, soft labeling dataset distillation (SLDD) [167], that generates pseudo-label and images. All these methods require the seed that initializes the network. Other methods, including Dataset Condensation with distribution matching (DM) [224], with differentiable Siamese augmentation (DSA) [223], and Dataset distillation by matching training trajectories (DDMT) [24] took a step further and devised seed-independent approaches. These methods often rely on a second-order optimization, which is computationally expensive and limits their application.Moreover, the size of data required for storage in these methods is proportional to the size of input images. We show that PRANC provides better accuracy with a much fewer number of regressed parameters on the same architectures compared to the mentioned approaches.

Image Compression: Some popular codecs like JPEG are based on hand-crafted modules to compress an image. Another line of image compression methods is learning-based approa-ches. These approaches usually train an auto-encoder on a large population of images [11, 130, 105, 42, 67] and store the code. Our method of using INR is also learning-based, but is different from the above methods since the model is learned on a single image (to overfit) rather than on a population of images. Hence, it may not suffer from the biases of the training data. COIN [44] is probably the closest to our method which overfits an INR and stores all the parameters. We are different since we compact the INR itself by learning a linear combination of random networks and store the coefficients.

3.2 Method

We are interested in training a deep model with a very small number of parameters so that it is less expensive to transfer the model from one agent to another or store it on an agent with small memory. This is in contrast to the goal of most prior works (e.g., model compression, pruning, or quantization) that aim to reduce the inference computation or improve the generalization while preserving the accuracy. Hence, we introduce a compact representation assuming no change in the model size, number of non-zero parameters, or the precision of computation.

We assume that the deep model can be written as a linear combination of a set of randomly initialized

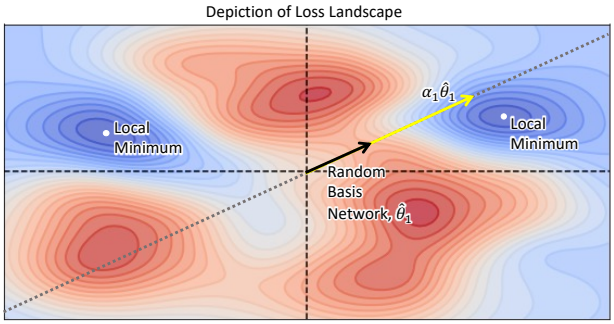


Figure 3.2: A simple illustration of the loss landscape of a model with two parameters and one basis model. None of the two local minimas may be in the span of the basis models, so we search for α to find a local minima in the span of the basis models.

models, called **basis**. Since we can use a pseudo-random generator to generate the random models, one can communicate or store all random models by simply communicating or storing a single scalar that is the seed of the pseudo-random generator. Although basis models are not necessarily orthogonal to each other, their pairwise dot product is close to zero since the number of samples (models) is much smaller than the dimensionality of each model. Then we optimize the weights of each base model so that their linear combination can solve the task (e.g., image classification).

More formally, given a set of training images $\{x_i\}_{i=1}^N$ and their corresponding labels $\{y_i\}_{i=1}^N$, we want to train a deep model $f(.;\theta)$ with parameters $\theta \in \mathbb{R}^d$ so that $f(x_i;\theta)$ predicts y_i. The standard practice is to optimize θ by minimizing the empirical risk:

$$R(\theta) = \frac{1}{N}\sum_{i=1}^{N} L(f(x_i;\theta), y_i)$$

where $L(\cdot,\cdot)$ is a discrepancy-measure, e.g., cross-entropy. In communicating such a model, we need to send a high-dimensional vector θ that contains d scalars.

To reduce the cost of communication, we assume a set of randomly initialized basis models with parameters $\{\hat{\theta}_j\}_{j=1}^k$. These basis models are generated using a known seed and are frozen throughout the learning process. Then we define: $\theta := \sum_{j=1}^k \alpha_j \hat{\theta}_j$, where α_j is a scalar weight for the j'th basis model. Assuming that $k \ll d$, it will be much less expensive to communicate or store α instead of θ.

To optimize α, one may first optimize for θ to find θ^* and then regress it by minimizing:

$$_\alpha||\theta^* - \sum_{j=1}^k \alpha_j \hat{\theta}_j||^2$$

However, since $k \ll d$, the optimum solution θ^* may be far from the span of the basis models,

resulting in an inferior solution (also shown empirically in our experiments). We argue that there are infinite number of solutions for θ that are as good as θ^*, so we may search for one of those that has smaller residual error when projected to the span of the basis models. Hence, we search for a solution that minimizes the task loss in the basis models' span by optimizing:

$$_{\alpha} \sum_i L\Big(f(x_i; \sum_{j=1}^{k} \alpha_j \hat{\theta}_j), y_i\Big)$$

Optimization efficiency: Note that the optimization is very simple and efficient since $\frac{dL}{d\alpha} = \frac{dL}{d\theta}\frac{d\theta}{d\alpha}$ and $\frac{d\theta}{d\alpha_j} = \hat{\theta}_j$. Hence, we use standard backpropagation to calculate $\frac{dL}{d\theta}$ and then multiply that with the basis models' matrix to get:

$$\frac{dL}{d\alpha} = \frac{dL}{d\theta} \times \hat{\theta}$$

Model reconstruction efficiency: Since basis models are generated using a pseudo-random generator, we can reconstruct the model using a simple running average of the basis models: generate each entry in $\hat{\theta}_j$, multiply it with α_j, add it to the running average, discard the entry and go to the next entry of $\hat{\theta}_j$. This way, the memory footprint of the reconstruction becomes negligible (i.e., $d + 1$).

On-demand model reconstruction: In some applications, the agent may need to run the inference rarely, but does not have enough memory to hold the model. The device can store α, reconstruct each convolutional filter using the corresponding entries of the basis models, apply it to the input, and then discard the filter and go to the next filter. This process has a very small memory footprint as it needs to store α and just one filter at a time.

Distributed learning: In order to train the model on multiple GPUs, we use a simple distributed learning algorithm to increase m, the number of basis models. We divide m basis models between g GPUs so that each GPU works on m/g basis models only. Then, we distribute α among GPUs. Each GPU calculates the partial weighted average over its basis models and distributes it to all GPUs. Then, all GPUs will have access to the complete weighted average and will use it to do backpropagation in standard distributed learning form and update their own set of α.

Batchnorm layer: We minimize the loss of the task by tuning the α instead of the model weights as done in standard learning. However, the parameters of the Batchnorm layer are not tuned by

PRANC. For the simplicity of this work, we assume that we can communicate those parameters and include them in the budget. This makes sense since the number of batchnorm parameters is relatively small compared to the number of weight parameters.

3.3 Application

We test our framework on two different applications.

Image classification networks: In this setting, we parameterize an image recognition model, e.g., ResNet-20 for CIFAR-10, by our PRANC framework and optimize αs instead of the model weights. This results in a compact model that can be stored and communicated very efficiently.

Image compression using implicit neural networks: We also test our framework on compressing an implicit neural network (INR) that is over-fitted to a single image [172]. Such an INR inputs the coordinate of the pixel and returns the color value. Hence, one can store or communicate the INR model instead of the original image. We parameterize a standard INR model [172] using PRANC framework so that we optimize the αs instead of the weights of the INR model. Our method outperforms JPEG compression on two standard datasets and two evaluation metrics.

3.4 Experiments on image classification

We report extensive results of PRANC on various datasets, architectures, and number of basis models.

3.4.1 Comparison with model pruning methods:

For communicating a sparse model with a sparsity rate of more than 50%, it is required to transmit 2 numbers per parameter: the value of a weight and its index in the network. Therefore, even if a model pruning method uses a pruning factor of 99% (100× reduction in size) since it should transmit the indices alongside the values, the actual reduction size will be smaller than 100×. DPF [119], STR[102], LAMP[107], RiGL[47], and SuRP[88] are the SOTA methods that use a large sparsity (+50×) and maintain a reasonable accuracy. We used their code on CIFAR-10 and CIFAR-100 along with ResNet-20 and ResNet-56 architectures and compared them with our method in Table

3.1. PRANC achieves consistently higher accuracy with fewer number of parameters. Please note that all these methods excluded batchnorm layers from their pruning process. Therefore we also excluded them from the parameter count. For ResNet-20, the number of BatchNorm parameters is 2,752 and for ResNet-56 it is 8,128.

Table 3.1: Comparison of our model with SOTA pruning methods, DPF [119], STR[102], LAMP[107], RiGL[47], and SuRP[88]. Pr. denotes the pruning rate. Also, when network is pruned, we have to keep two numbers for each weight: the weight itself and its position in the model. Note that we excluded the number of BatchNorm parameters in this table since that is constant for all the models. This number is 2,752 for Resnet-20 and 8,128 for ResNet-56.

Method	**Data**	**Arch.**	**# Params ex. BatchNorm**	**Acc.**
DPF(Pr. 98.2%)	C10	R20	4,920×2	41.86
RiGL(Pr. 99.62%)	C10	R20	1026×2	50.9
LAMP(Pr. 99.62%)	C10	R20	1026×2	51.24
SuRP (Pr. 99.62%)	C10	R20	1026×2	54.22
STR (Pr. 95.5%)	C10	R20	12,238×2	75.99
Ours	**C10**	**R20**	**1,000**	**64.59**
Ours	**C10**	**R20**	**10,000**	**81.48**
DPF (Pr. 98.43%)	C10	R56	13,414×2	47.66
SuRP (Pr. 98.73%)	C10	R56	10,834×2	66.65
STR (Pr. 98.4%)	C10	R56	13,312×2	67.77
Ours	**C10**	**R56**	**5,000**	**76.87**
DPF (Pr. 96.13%)	C100	R20	10,770×2	12.25
SuRP (Pr. 97.48%)	C100	R20	6,797×2	14.46
STR (Pr. 96.12%)	C100	R20	10,673×2	13.18
Ours	**C100**	**R20**	**5,000**	**32.33**
DPF (Pr. 97.8%)	C100	R56	19,264×2	19.11
SuRP (Pr. 98.72%)	C100	R56	10,919×2	14.59
STR (Pr. 97.8%)	C100	R56	18,881×2	25.98
Ours	**C100**	**R56**	**5,000**	**32.97**

3.4.2 Comparison with model distillation methods:

One of the critical baselines for our work is model distillation. However, the number of parameters we use is very small compared to any existing CNN architecture. Even the smallest invented CNN architecture, LeNet[103], has more than 60,000 parameters. To compare PRANC with model distillation, we trained a ResNet18 on CIFAR-10 and distilled its knowledge to a LeNet model. On the other hand, we compressed a ResNet20 model using PRANC with 10,000 αs and a ResNet56 model with merely 5,000 αs and compared their accuracies. The results are presented in Table 3.2. As we can see, PRANC-compressed architectures require almost 5× fewer parameters while achieving higher accuracies with a significant gap.

Table 3.2: Comparison with model distillation. PRANC outperforms a LeNet distilled from ResNet-18 on CIFAR-10. 2,752 and 8,128 are the number of BatchNorm parameters which we exclude from the coefficients but need to consider them as parameters.

Method	Arch.	# Params	Acc.
Distilled from R18	LeNet	62,006	74.1%
Ours	**R56**	**5,000 + (8,128)**	**76.87%**
Ours	**R20**	**10,000 + (2,752)**	**81.48%**

3.4.3 Comparison to Dataset Distillation methods:

In Table 3.3, we report the accuracy and the number of parameters for PRANC in comparison with various dataset distillation methods. Most of these methods are based on meta-learning approaches that involve a high computational cost and memory footprint at the training time, so they are limited in the depth of the model. Moreover, they need to do a few gradient descend steps in constructing the model. Nonetheless, the number of required parameters in dataset distillation methods is proportional to the size of the input image. For instance, in distilling CIFAR-10 t0 10 images only, we need to store at least $10 \times 32 \times 32 \times 3$ parameters. In order to be comparable with the SOTA Dataset Distillation methods, we use AlexNet (which is a modified version that is described in [194]) on CIFAR-10. For CIFAR-100 and tinyImageNet, we use depth-3 and depth-4 128-width ConvNet architectures described in [24], respectively. Note that some dataset distillation methods do not require a seed, so they solve a more challenging task since the distilled data should be able to tune any randomly initialized model. However, since we are focusing on reducing the cost of communication and storage, using a fixed seed as the central part of our idea is not prohibitive.

3.4.4 Large-Scale dataset and models

So far, we have provided evidence that our method can outperform recent works in dataset distillation, model pruning, and model compression in terms of the number of parameters vs. accuracy. Since our method is reasonably efficient in learning, particularly compared to meta-learning approaches that depend on second order derivatives of the network, we can evaluate it on larger scale models. We evaluate our method on ImageNet100 with ResNet-18 architecture. Table 3.4 shows the results. Our method achieves 61.08% accuracy with less than 1% parameters, while the standard ResNet-18 model achieves 82.1% accuracy with more than 11M parameters. Since ResNet-18 is a huge model, we use one of the unique capabilities of PRANC, which is creating the network on the fly. We only

Table 3.3: Comparison of PRANC with dataset distillation methods on various datasets and architectures. 3-128-Conv and 4-128-Conv represents 3-depth 128-width ConvNet and 4-depth 128-width ConvNet, respectively. “Trained model" is the upper-bound of our method since one can optimize all weights and transmit/store them. Our method outperforms the baselines with a large margin and much fewer number of parameters.

Method	**Task**	**Arch**	**# Params**	**Acc.**
Trained model	C10	AlexNet	1,756,426	84.8
FDD [17]	C10	AlexNet	397,000	43.2
SLDD [167]	C10	AlexNet	308,200	60.0
DD [194]	C10	AlexNet	307,200	54.0
DM [224]	C10	AlexNet	30,720	26.0
DSA [223]	C10	AlexNet	30,720	28.8
DC [225]	C10	AlexNet	30,720	28.3
CAFE [191]	C10	AlexNet	30,720	30.3
CAFE+DSA [191]	C10	AlexNet	30,720	31.6
DDMT [24]	C10	AlexNet	30,720	46.3
Ours	C10	AlexNet	**17,000**	**76.69**
Trained model	C100	3-128-Conv	504,420	56.2
FDD [17]	C100	3-128-Conv	317,200	11.5
DM [224]	C100	3-128-Conv	307,200	11.4
DSA [223]	C100	3-128-Conv	307,200	13.9
DC [225]	C100	3-128-Conv	307,200	12.8
CAFE+DSA [191]	C100	3-128-Conv	307,200	14.0
DDMT [24]	C100	3-128-Conv	307,200	24.3
Ours	C100	3-128-Conv	**15,000**	**25.57**
Trained model	tinyIN	4-128-Conv	857,160	37.6
DM [224]	tinyIN	4-128-Conv	2,457,600	3.9
DDMT [24]	tinyIN	4-128-Conv	2,457,600	8.8
Ours	tinyIN	4-128-Conv	**15,000**	**12.02**

used a single NVIDIA 3090 GPU and trained our model for 200 epochs, using Adam optimizer and step scheduler with $\gamma = 0.5$ for every 50 epochs and an initial learning rate of 0.001. Also, we distributed our budget of α vector across the layers, *i.e.,* we used 4,000 coefficients for each layer of the convolutional encoder and 20,000 coefficients for our classifier (giving us a total of 100,000 coefficients).

3.4.5 Ablation study:

Our method adds a hyperparameter to the training process which is $K = |\alpha|$. One question is: “How will K affect the performance?” Moreover, it is arguable that “why do we try to find a linear combination that is accurate on the task? why not try to regress an entire trained network?” Also, “Can K be more important than the architecture? *i.e.,* can we use large K with a small network and

Table 3.4: Result of our method on ImageNet-100 dataset and ResNet-18. 19,200 is the total number of parameters of all BatchNorm layers in our model

Method	**# Params**	**Acc.**
trained model	11,227,812	**82.1%**
HashedNet [29]	110,000 + (19,200)	52.96%
Ours	**100,000 + (19,200)**	**61.08%**
Ours	**200,000 + (19,200)**	**67.28%**

still get high accuracy?" In this section, we answer these questions.

Sensitivity to seed: Since one of the main applications of PRANC is federated learning, it is worth discussing its pseudo-encryption ability. We experimented with changing the seed at the reconstruction time. On CIFAR-10 with AlexNet, with a minor change in seed, the accuracy of the reconstructed model dropped from **74.0%** to **9.4%**, which is close to chance. This is expected as the new basis models are not correlated with the ones that are used in training. This fact means that the seed can act as a mutual key between the agents and even if α is intercepted, reconstructing the model is nearly impossible. Therefore, in federated learning applications which deal with safe communication of deep learning models, PRANC can be used as both **compression** and an **encryption** method.

Regressing θ^* directly: We can first train a model to get θ^* and then optimize for α by regressing that solution using MSE loss in the parameter space. As shown in Fig. 3.2, this may not succeed since the optimal model may not be in the span of the basis models, and also the MSE loss in the parameter space is not necessarily correlated with the task loss. Table 3.5 shows that the accuracy of this baseline using 10,000 parameters is close to chance.

Table 3.5: Results of regressing a pretrained model using 10,000 basis models. C10 and C100 denote CIFAR-10 and CIFAR-100 and tinyIN denotes tiny ImageNet datasets

Dataset	**Architecture**	**Full Acc.**	**Regress. Acc.**
C10	AlexNet	84.8 %	10.0%
C10	LeNet [103]	73.5%	12.74%
C100	3-128-Conv	56.2 %	1.14%
C100	AlexNet	50.7%	1.0%
tinyIN	4-128-Conv	37.6 %	0.5%

Effect of varying k vs Architecture: We perform an ablation study to understand the effect of the number of basis models, k vs. the architecture of the model. One can argue that sometimes it's

better to design a better architecture rather than increase the number of alphas. Here, we change k from 1,000 to 20,000 for LeNet, AlexNet, ResNet-20, and ResNe-56 on CIFAR-10 and plot the accuracy in Figure 3.3. All experiments have been done in the same setup with 400 epochs and Adam optimizer. As expected, the accuracy increases as we increase the number of basis functions. However, as we can see, the architecture has more effect on the accuracy rather than the k.

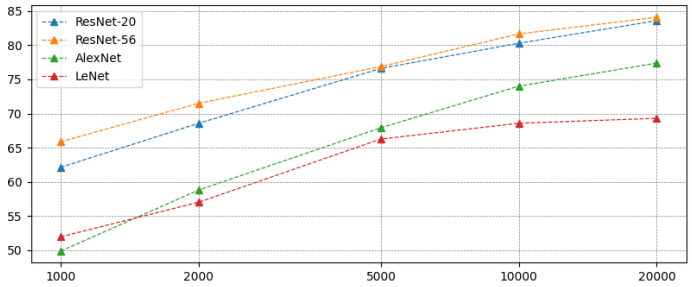


Figure 3.3: Illustration of impact of k in accuracy of different architectures on CIFAR-10. The accuracy improves by increasing the number of basis models. However, the architecture plays more critical role in the accuracy compared to the number of basis.

3.5 Experiments on image compression

As another application of PRANC, here we show that we can compress an image by compacting the implicit neural network that is overfitted to the image. The INR model inputs the coordinates of a pixel and outputs the color of that pixel. We will store or communicate the seed of the pseudo-random generator and alpha values only. We use [172] as our INR model. The INR is a 4 layer MLP (3 hidden layers) model with 512 neurons at each layer and a Layer-Norm [7] after each layer. We encode the pixel coordinate in the input to 512 dimensions using Fourier mapping and use three neurons in the output for RGB images (the color values) and one neuron for X-ray images.

We use half-precision floating point for αs to further reduce the storage cost. We use a different set of αs for each layer of the network.

To reduce the memory footprint, we divide the base network matrix $\hat{\theta}$ into smaller chunks and generate and then discard each chunk at the GPU for each iteration of the optimization. Similar to stochastic gradient descent, we randomly sample a subset of pixels to be optimized at each iteration, leading to faster convergence due to more frequent parameter updates.

We evaluate our model on three different datasets: **Kodak** dataset [97] that contains 24 non-compressed images of size 512×768, **CLIC-mobile** test set [179] that contains 178 high resolution images (e.g., 1512×2016), and 64 randomly sampled images from NIH Chest X-ray dataset [196] that consists of $100,000$ de-identified chest x-rays images of size 1024×1024.

Baselines. We compare our method with the following hand-crafted image codecs: JEPG, JPEG2000, and WebP. Note that our goal is to show that PRANC is a general framework that performs well when simply applied to INR compression out-of-the-box. Hence, we do not compare it with more advanced codecs like BPG and VTM since they are highly engineered and include components like entropy coding. Those results are presented in the supp. We also compare with learning-based image compression methods in the supplementary (BMS, MBT, and CST). Note that unlike our method, these methods require a large dataset of images to pre-train their auto-encoder. This limits the applications and also may degrade the results for out-of-distribution data points.

COIN [44] is probably the closest to our method which trains an INR using SIREN [160] and stores all the parameters. Since COIN does not use Fourier mapping, for a fair comparison, we produce a similar baseline, called 'trained INR,' using our MLP network described above without the PRANC framework. For the trained INR, we reduce the width of the network to match the compression rate of our method and use half-precision floating points.

Results. We evaluate our model with two metrics: PSNR and MS-SSIM. Note that we fix the network architecture and vary the number of alpha values to get different bit-per-pixel (bpp) values for each image. We report the results for Kodak dataset on Figure B.6 and CLIC-mobile dataset on Table B.3. Our method outperforms JPEG and INR. We report the results of chest x-ray in Table 3.7.

Ablation. Since we reconstruct the model weights with alpha values, one can vary the size of the architecture without changing the number of parameters (alpha values). Note that we can increase both width and depth without changing the bit-rate. We keep the number of parameters $k = 102$K and vary both network width and depth of the MLP model. Results on Kodak dataset are shown in Table 3.8. Interestingly, we can improve the performance by increasing the model depth while keeping the number of learnable parameters constant.

Table 3.6: **CLIC-mobile Dataset Image Compression:** PRANC outperforms JPEG and JPEG2000 with smaller bpp on this dataset.

Model	bpp	PSNR	MS-SSIM
WebP	0.185	30.07	0.940
JPEG2000	0.126	29.40	0.918
JPEG	0.195	24.82	0.836
Trained INR	0.125	26.93	0.864
PRANC (ours)	0.119	29.71	0.920

Table 3.7: **Chest x-ray Dataset Image compression:** We compare PRANC with JPEG for X-ray images. Our method is better than JPEG with lower bpp. Since unlike auto-encoders, PRANC does not use any population based training, it may be better suited for medical images since it may preserve out-of-distribution artifacts which are important for diagnosis purposes. However, we leave studying this for the future work.

Model	PRANC	JPEG
bpp	0.152	0.168
PSNR	36.28	34.25
MS-SSIM	0.972	0.921

Implementation Details. For each image, we train alpha values with $10k$ iterations on Kodak and $5k$ iterations on CLIC-mobile dataset. Each iteration processes 25% of the pixels of the image sampled randomly. Note that increasing the number of iterations cannot damage the model since the goal is to overfit to the image and generalization is not an issue. We use PyTorch Adam [95] optimizer with initial learning rate of $1e-3$ and Cosine scheduler. More details about number of alpha values per layer for each bpp is in the supplementary material.

3.6 Conclusion

We introduced a simple yet effective method that can learn a model as a linear combination of a set of frozen randomly initialized models. The final model can be compactly stored or communicated using the seed of the pseudo-random generator and the coefficients. Moreover, our

Table 3.8: **Effect of increasing width/depth of the model:** We can increase both depth and width of the MLP model without changing the number of parameters. When changing the depth, we redistribute the number of alpha values uniformly among layers to keep the total number constant. More details are in the supp.

Width	128	256	512	1024
PSNR	30.00	32.05	31.5	31.32
MS-SSIM	0.937	0.963	0.959	0.961

Depth	3	4	5
PSNR	30.78	31.5	32.38
MS-SSIM	0.978	0.959	0.965

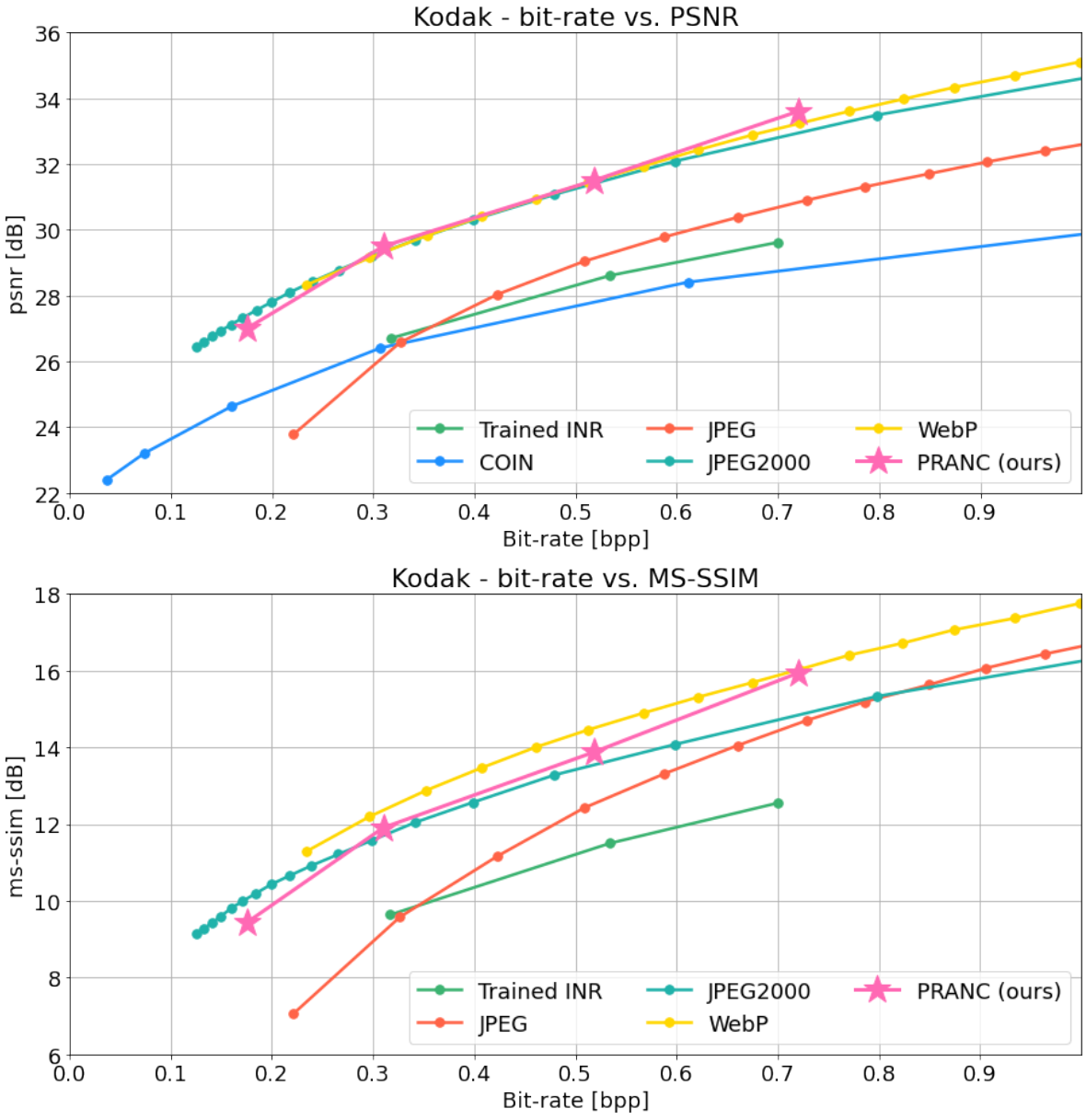


Figure 3.4: **Kodak Dataset Image Compression:** Our method outperforms JPEG and 'trained INR' on both PSNR and MS-SSIM evaluations at various bit-rates. Note that unlike the other baselines, our method is learned on a single image and is not hand-crafted, except the architecture of the INR model, which is a simple MLP.

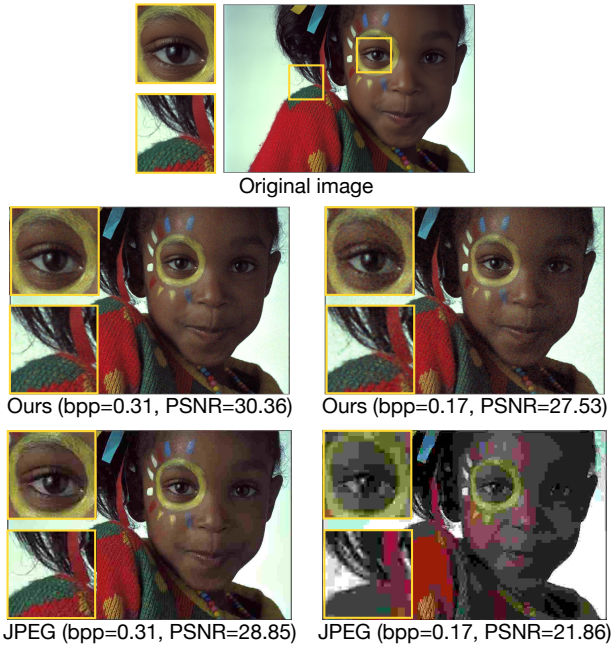


Figure 3.5: **Kodak visualization** We compare PRANC with JPEG on image 15 of Kodak. More visualization are in the supp.

method has a small memory footprint at the learning or reconstruction stages. We perform extensive experiments on multiple image classification datasets with multiple archite-ctures and also on image compression settings and show that our method achieves better accuracy with fewer number of parameters compared to SOTA baselines. We believe many applications including lifelong learning and distributed learning can benefit from our ideas. Hence, we hope this paper opens the door to studying more advanced compacting methods based on frozen random networks.

Chapter 4

CompGS: Smaller and Faster Gaussian Splatting

In the previous chapter, we explored PRANC, which advanced the efficiency of parameters by compressing neural networks through a reduction in the number of trainable parameters. This approach effectively lowered the memory and computational requirements for both the model and any data represented through neural networks. In this chapter, we continue our work on parameter efficiency, but focus on data compression. Similar to Neural Radiance Fields (NeRF), 3D Gaussian Splatting (3DGS) can also be viewed as a neural representation that overfits to a specific 3D scene. Therefore, by compressing 3DGS, we are effectively compressing the data itself. This time, instead of using PRANC, we use vector quantization.

Recently, we have seen great progress in radiance field methods to reconstruct a 3D scene using multiple images captured from multiple viewpoints. NeRF [129] is probably the most well-known method that employs an implicit neural representation to learn the radiance field using a deep model. Although very successful, NeRF methods are very slow to train and render. Several methods have been proposed to solve this problem; however, they usually come with some cost in the quality of the rendered images.

The Gaussian Splatting method (3DGS) [91] is a new paradigm in learning radiance fields. The idea is to model the scene using a set of Gaussians. Each Gaussian has several parameters including

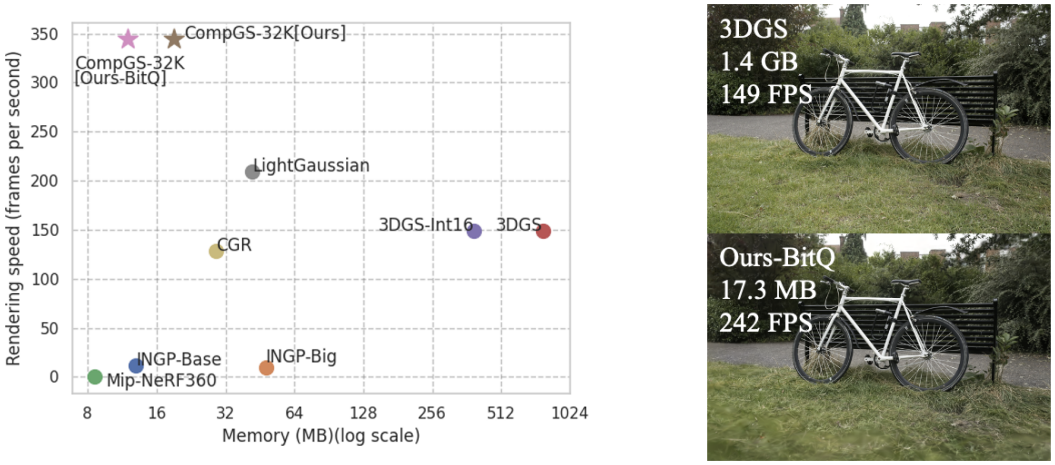


Figure 4.1: **Inference speed vs. memory comparison.** All methods except INGP achieve comparable PSNR that are reported in Table 4.1. CompGS, our compressed version of 3DGS, maintains the speed and performance of 3DGS while reducing its size to the levels of NeRF based approaches. We achieve around 45× compression and 2.5× inference speed up with little drop in performance (CompGS-32K). A bit quantized version of this (Ours-BitQ) compresses it further to a total compression of 65× with hardly noticeable difference in quality.

its position in 3D space, covariance matrix, opacity, color, and spherical harmonics of the color that need to be learned from multiple-view images. Thanks to the simplicity of projecting 3D Gaussians to the 2D image space and rasterizing them, 3DGS is significantly faster to both train and render compared to NeRF methods. This results in real-time rendering of the scenes on a single GPU (ref. Fig. 4.1). Additionally, unlike the implicit representations in NeRF, the 3D structure of the scene is explicitly stored in the parameter space of the Gaussians. This enables many operations including editing the 3D scene directly in the parameter space.

One of the main drawbacks of the 3DGS method compared to NeRF variants is that 3DGS needs at least an order of magnitude more parameters compared to NeRF. This increases the storage and communication requirements of the model and its memory at the inference time, which can be very limiting in many real-world applications involving smaller devices. For instance, the large memory consumption may be prohibitive in storing, communicating, and rendering several radiance field models on AR/VR headsets.

We are interested in compacting 3DGS representations without sacrificing their rendering speed to enable their usage in various applications including low-storage or low-memory devices and AR/VR headsets. Our main intuition is that several Gaussians may share some of their parameters (e.g.

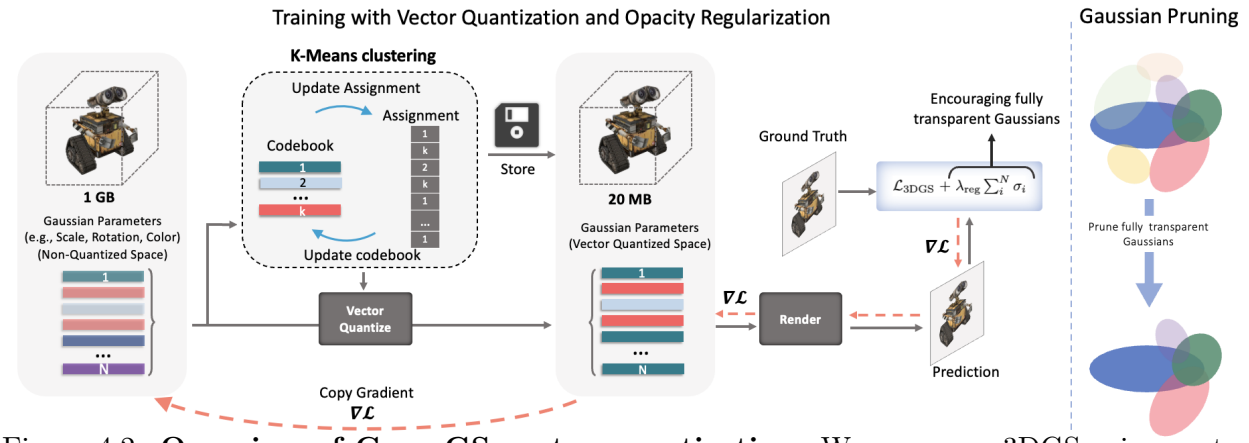


Figure 4.2: **Overview of CompGS vector quantization:** We compress 3DGS using vector quantization of the parameters of the Gaussians. The quantization is performed along with the training of the Gaussian parameters. Considering each Gaussian as a vector, we perform K-means clustering to represent the N Gaussians in the model with k cluster centers (codes). Each Gaussian is then replaced by its corresponding code for rendering and loss calculation. The gradients wrt centers are copied to all the elements in the corresponding cluster and the non-quantized versions of the parameters are updated. Only the codebook and code assignments for each Gaussian are stored and used for inference. To further reduce the storage and inference time, we regularize opacity in the loss to encourage fully transparent Gaussians. CompGS maintains the real-time rendering property of 3DGS while compressing it by an order of magnitude.

covariance matrix). Hence, we simply vector-quantize parameters while learning and store the codebook along with the index for each Gaussian. This can result in a huge reduction in the storage. Also, it can reduce the memory footprint at the rendering time since the index can act as a pointer to the correct code freeing the memory needed to replicate those parameters for all Gaussians.

To this end, we use simple K-means algorithm to vector quantize the parameters at the learning time. Inspired by various quantization-aware learning methods in deep learning [149], we use the quantized model in the forward pass while updating the non-quantized model in the backward pass. To reduce the computation overhead of running K-means, we update the centroids in each iteration, but update the assignments less frequently (e.g., once every 100 iterations) since it is costly. Moreover, since the Gaussians are a set of non-ordered elements, we compress the representation further by sorting the Gaussians based on one of the quantized indices and storing them using the Run-Length-Encoding (RLE) method. Furthermore, we employ a simple regularizer to promote zero opacity (essentially invisible Gaussians), resulting in a significant reduction in storage and rendering time by reducing the number of Gaussians. Our final model is 40× to 50× smaller and 2× to 3× faster during rendering compared to 3DGS.

Author contributions. This work was conducted in collaboration with K. L. Navaneet, S. A. Koohpayegani, and H. Pirsiavash. I proposed the main idea and made substantial contributions to the implementation of the method, the design and execution of the experiments, and the writing of the paper.

4.1 Related Work

Novel-view synthesis methods: Early deep learning techniques for novel-view synthesis used CNNs to estimate blending weights or texture-space solutions [226, 51, 77, 175, 154]. However, the use of CNNs faced challenges with MVS-based geometry and caused temporal flickering. Volumetric representations began with Soft3D [144], and subsequent techniques used deep learning with volumetric ray-marching [80, 161]. Mildenhall et al. introduced Neural Radiance Fields (NeRFs) [129] to improve the quality of synthesized novel views, but the use of a large Multi-Layer Perceptron (MLP) as the backbone and dense sampling slowed down the process a lot. Successive methods sought to balance quality and speed, with Mip-NeRF360 achieving top image quality [12]. Recent advances prioritize faster training and rendering via spatial data structures, encodings, and MLP adjustments [53, 214, 78, 33, 56, 153, 205, 171, 133]. Notable methods, like InstantNGP [133], use hash grids and occupancy grids for accelerated computation with a smaller MLP, while Plenoxels [53] entirely forgo neural networks, relying on Spherical Harmonics for directional effects. Despite impressive results, challenges in representing empty space, limitations in image quality, and rendering speed persist in NeRF methods. In contrast, 3DGS [91] achieves superior quality and faster rendering without implicit learning [12]. However, the main drawback of 3DGS is its increased storage compared to NeRF methods which may limit its usage in many applications such as edge devices. We are able to keep the quality and fast rendering speed of 3DGS method while providing reduced model storage by applying a vector quantization scheme to Gaussian parameters.

Bit quantization: Reducing the number of bits to represent each parameter in a deep neural network is a commonly used method to quantize models [89, 71, 100] that result in smaller memory footprints. Representing weights in 64 or 32-bit formats may not be crucial for a given task, and a lower-precision quantization can lead to memory and speed improvements. Dettmers et al. [40] show 8-bit quantization is sufficient for large language models. In the extreme case, weights of

neural networks can be quantized using binary values. XNOR [148] examines this extreme case by quantization-aware training of a full-precision network that is robust to quantization transformations.

Vector quantization: Vector quantization (VQ) [64, 58, 46, 63] is a lossy compression technique that converts a large set of vectors into a smaller codebook and represents each vector by one of the codes in the codebook. As a result, one needs to store only the code assignments and the codebook instead of storing all vectors. VQ is used in many applications including image compression [37], video and audio codec [111, 123], compressing deep networks [34, 63], and generative models [183, 66, 150]. We apply a similar method to compressing 3DGS models.

Deep model compression. Model compression tries to reduce the storage size without changing the accuracy of original models. Model compression techniques can be divided to 1) model pruning [68, 71, 184, 200] that aims to remove redundant layers of neural networks; 2) weight quantization [138, 89, 100], and 3) knowledge distillation [8, 81, 26, 146, 3], in which a compact student model is trained to mimic the original teacher model. Some works have applied these techniques to volumetric radiance fields [39, 116, 214]. For instance, TensoRF [25] decompose volumetric representations via low-rank approximation.

Compression for 3D scene representation methods. Since NeRF relies on dense sampling of color values and opacity, the computational costs are significant. To increase efficiency, methods adopt different data structures such as trees [192, 214], point clouds [209, 143], and grids [25, 53, 133, 158, 169, 171]. With grid structures training iterations can be completed in a matter of minutes. However, dense 3D grid structures may require substantial amounts of memory. Several methods have worked on reducing the size of such volumetric grids [173, 25, 171, 133]. Instant-NGP [133] uses hash-based multi-resolution grids. VQAD [171] replaces the hash function with codebooks and vector quantization. Another line of work decomposes 3D grids into lower dimensional components, such as planes and vectors, to reduce the memory requirements [25, 173, 84]. Despite reducing the time and space complexity of the 3D scenes, their sizes are still larger than MLP-based methods. VQRF [115] compresses volumetric grid-based radiance fields by adopting the VQ strategy to encode color features into a compact codebook.

While we also employ vector quantization, we differ from the above approaches in the method employed for novel view synthesis. Unlike the NeRF based approaches described above, we aim to

compress 3DGS which uses a collection of 3D Gaussians to represent the 3D scene and does not contain grid like structures or neural networks. We also achieve a significant amount of compression by regularizing and pruning the Gaussians based on their opacity.

Concurrent works: Some very recent works developed concurrently to ours [135, 48, 108, 132, 61] also propose vector quantization and pruning based methods to compress 3D Gaussian splat models. LightGaussian [48] uses importance based Gaussian pruning and distillation and vector quantization of spherical harmonics parameters. Similarly, CGR [108] masks Gaussians based on their volume and transparency to reduce the number of Gaussians and uses residual vector quantization for scale and rotation parameters. In CGS [135], highly sensitive parameters are left non-quantized while the less sensitive ones are vector quantized.

4.2 Method

Here, we briefly describe the 3DGS [91] method for learning and rendering 3D scenes and explain our vector quantization approach for compressing it.

Overview of 3DGS: 3DGS models a scene using a collection of 3D Gaussians. A 3D Gaussian is parameterized by its position and covariance matrices in the 3D space. $G(x) = e^{-\frac{1}{2}(x-\mu)^T\Sigma^{-1}(x-\mu)}$ where $x-\mu$ is the position vector, μ is the position, and Σ is the 3D covariance matrix of the Gaussian. Since the covariance matrix needs to be positive definite, it is factored into its scale (S) and rotation (R) matrices as $\Sigma = RSS^TR^T$ for easier optimization. In addition, each Gaussian has an opacity parameter σ. Since the color of the Gaussians may depend on the viewing angle, the color of each Gaussian is modeled by a Spherical Harmonics (SH) of order 3 in addition to a DC component.

Given a view-point, the collection of 3D Gaussians is efficiently rendered in a differentiable manner to get a 2D image by α-blending of anisotropic splats, sorting, and using a tile-based rasterizer. Color of a pixel is given by $C = \sum_i c_i\alpha_i\prod_{j=1}^{i-1}(1-\alpha_j)$ where c_i is the color of the i^{th} Gaussian and α_i is the product of the value of the Gaussian at that point and its learned opacity σ_i. At the training time, 3DGS minimizes the loss between the groundtruth and rendered images in the pixel space. The loss is ℓ_1 loss plus an SSIM loss in the pixel space. 3DGS initializes the optimization by

a point cloud achieved by a standard SfM method and iteratively prunes the ones with small opacity and adds new ones when the gradient is large. 3DGS paper shows that it is extremely fast to train and is capable of real-time rendering while matching or outperforming SOTA NeRF methods in terms of rendered image quality.

Compression of 3DGS:

We compress the parameters of 3DGS using vector quantization aware training and reduce the number of Gaussians by regularizing the opacity parameter.

Vector quantization: 3DGS requires a few million Gaussians to model a typical scene. With 59 parameters per Gaussian, the storage size of the trained model is an order of magnitude larger than most NeRF approaches (e.g., Mip-NeRF360 [12]). This makes it inefficient for some applications including edge devices. We are interested in reducing the number of parameters. Our main intuition is that many Gaussians may have similar parameter values (e.g., covariance). Hence, we use simple vector quantization using K-means algorithm to compress the parameters. Fig. 5.1 provides an overview of our approach.

Consider a 3DGS model with N Gaussians, each with a d dimensional parameter vector. We run K-means algorithm to cluster the vectors into K clusters. Then, one can store the model using K vectors of size d and N integer indices (one for each Gaussian). Since $N >> K$, this method can result in a large compression ratios. In a typical scene, N is a few millions while K is a few thousands.

However, clustering the model parameters after training results in performance degradation, hence, we perform quantization aware training to ensure that the parameters are amenable to quantization. In learning 3DGS, we store the non-quantized parameters. In the forward pass of learning 3DGS, we quantize the parameters and replace them with the quantized version (centroids) to do the rendering and calculate the loss. Then, we do the backward pass to get the gradients for the quantized parameters and copy the gradients to the non-quantized parameters to update them. We use straight-through estimator proposed in STE [16]. After learning, we discard the non-quantized parameters and keep only the codebook and indices of the codes for Gaussians.

Since the number of Gaussians N is typically in millions, cost of performing K-means at every

iteration of training can be prohibitively high. K-means has two steps: updating centroids given assignments, and updating assignments given centroids. We note that the latter is more expensive while the former is a simple averaging. Hence, we update the centroids after each iteration and update the assignments once every t iterations. We observe that the modified approach works well even for values of t as high as 500. This is crucial in limiting the training time of the method.

Performing a single K-means for the whole d dimensional parameters requires a huge codebook since the different parameters of the Gaussian are not necessarily correlated. Hence, we group similar types of parameters, e.g., all rotation matrices, together and cluster them independently to learn a separate codebook for each. This requires storing multiple indices for each Gaussian. In our main method, we quantize DC component of color, spherical harmonics, scale, and rotation parameters separately, resulting in 4 codebooks. We do not quantize opacity parameter since it is a single scalar and do not quantize the position of the Gaussians since sharing them results in overlapping Gaussians.

Since the indices are integer values, we use fewer number of bits compared to the original parameters to store each. Moreover, 3DGS models the scene as a set of order-less Gaussians. Hence, we sort the Gaussians based on one of the indices, e.g., rotation, so that Gaussians using the same code appear together in the list. Then, for that index, instead of storing n integers, we store how many times each code appears in the list, reducong the storage from n integers to k integers. This is similar to run-length-encoding for data compression.

Opacity Regularization: Some parameters like position of the Gaussians cannot be quantized easily, so as shown in Table 4.6 after quantization, they dominate the memory(more than 80% of memory). This means quantization cannot improve the compression any further. One way to compress 3DGS more is to reduce the number of Gaussians. Interestingly, this reduction comes with a bi-product that is increase in inference speed. We know that very small values of opacity (σ) correspond to transparent or nearly invisible Gaussians. Hence, inspired by training sparse models, we add ℓ_1 norm of the opacity to the loss as a regularizer to encourage zero values for opacity. Therefore, the final loss becomes: $\mathcal{L} = \mathcal{L}_{\text{3DGS}} + \lambda_{\text{reg}} \sum_i^N \sigma_i$, where $\mathcal{L}_{\text{3DGS}}$ is the original loss of 3DGS with or without quantization and λ_{reg} controls the sparsity of opacity. Finally, similar to the original 3DGS, we remove the Guassians with opacity smaller than a threshold, resulting in

significant reduction in storage and inference time.

4.3 Experiments

Implementation details: For all our experiments, we use the publicly available official code repository [1] of 3DGS [91] provided by its authors. There are no changes in the hyperparameters used for training compared to 3DGS. The Gaussian parameters are trained without any vector quantization till $20K$ iterations and K-means quantization is used for the remaining $10K$ iterations. A standard K-means iteration involves distance calculation between all elements (Gaussian parameters) and all cluster centers followed by assignment to the closest center. The centers are then updated using new cluster assignments and the loop is repeated. We use just 1 such K-means iteration in our experiments once every 100 iterations till iteration $25K$ and keep the assignments constant thereafter till the last iteration, $30K$. The K-means cluster centers are updated using the non-quantized Gaussian parameters after each iteration of training. The covariance (scale and rotation) and color (DC and harmonics) components of each Gaussian is vector quantized while position (mean) and opacity parameters are not quantized. Additional results with different parameters being quantized are provided in Table 4.9. Unless mentioned differently, we use a codebook of size 4096 for the color and 16384 (CompGS $16K$) for the covariance parameters. The scale parameters of covariance are quantized before applying the exponential activation on them. Similarly, quaternion based rotation parameters are quantized before normalization. For opacity regularization, we use $\lambda_{\text{reg}} = 10^{-7}$ from iterations $15K$ to $20K$ along with opacity based pruning every 1000 iterations and remove regularization thereafter. All experiments were run on a single RTX-6000 GPU.

Datasets: We primarily show results on three challenging real world datasets; Tanks&-Temples[96], Deep Blending [77] and Mip-NeRF360 [12] containing two, two and nine scenes respectively. Also, we provide results on a subset of the recently released DL3DV-10K dataset [120] which contains 140 scenes. DL3DV-10K [120] is an annotated dataset with $10,510$ real-world scene-level videos. Out of these, 140 scenes have been used to create a novel-view synthesis (NVS) benchmark, making it an order of magnitude larger than the typical NVS benchmarks. We use this NVS benchmark in our experiments. Additionally, we provide results on a subset of the large scale ARKit [14] dataset, called ARKit-200, which contains 200 scenes. Details of this dataset is presented in the Appendix.

Table 4.1: **Comparison with SOTA methods for novel view synthesis.** 3DGS [91] performs comparably or outperforms the best of the NeRF based approaches while maintaining a high rendering speed during inference. Trained NeRF models are significantly smaller than 3DGS since NeRFs are parameterized using neural networks while 3DGS requires storage of parameters of millions of 3D Gaussians. CompGS is a vector quantized version of 3DGS that maintains the speed and performance advantages of 3DGS while being **40×** to **50× smaller**. CompGS 32K BitQ is the post-training bit quantized version of CompGS 32K, in which position parameters are 16-bits, opacity is 8 bits, and the rest are 32 bits. *Reproduced using official code. † Reported from 3DGS [91]. Our timings for 3DGS and CompGS are reported using a RTX6000 GPU while those with † used A6000 GPU. We boldface entries for emphasis. Please see the Appendix for results on Deep Blending dataset.

	Mip-NeRF360						Tanks&Temples					
Method	SSIM↑	PSNR↑	LPIPS↓	FPS	Mem (MB)	Train Time(m)	SSIM↑	PSNR↑	LPIPS↓	FPS	Mem (MB)	Train Time(m)
Plenoxels† [53]	0.626	23.08	0.463	6.79	2,100	25.5	0.719	21.08	0.379	13.0	2300	25.5
INGP-Base† [133]	0.671	25.30	0.371	11.7	13	5.37	0.723	21.72	0.330	17.1	13	5.26
INGP-Big† [133]	0.699	25.59	0.331	9.43	48	7.30	0.745	21.92	0.305	14.4	48	6.59
M-NeRF360† [12]	0.792	27.69	0.237	0.06	8.6	48h	0.759	22.22	0.257	0.14	8.6	48h
3DGS † [91]	0.815	27.21	0.214	134	734	41.3	0.841	23.14	0.183	154	411	26.5
3DGS * [91]	0.813	27.42	0.217	**149**	**778**	21.6	0.844	23.68	0.178	**206**	**433**	12.2
LigthGaussian [48]	0.805	27.28	0.243	209	42	-	0.817	23.11	0.231	209	22	-
CGR [109]	0.797	27.03	0.247	128	29.1	-	0.831	23.32	0.202	185	20.9	-
CGS [136]	0.801	26.98	0.238	-	28.8	-	0.832	23.32	0.194	-	17.28	-
CompGS 16K	0.804	27.03	0.243	**346**	**18**	22.8	0.836	23.39	0.200	**479**	**12**	15.6
CompGS 32K	0.806	27.12	0.240	344	19	29.4	0.838	23.44	0.198	475	13	20.6
CompGS 32K BitQ	0.797	26.97	0.245	**344**	**12**	29.4	0.832	23.35	0.202	**475**	**8**	20.6

Baselines: As we propose a method (termed CompGS) for compacting 3DGS, we focus our comparisons with 3DGS and different baseline methods for compressing it. We consider bit quantization (denoted as Int-16/8/4 in results) and 3DGS without the harmonic components for color (denoted as 3DGS-No-SH) as alternative parameter compression methods. Bit-quantization is performed using the standard Absmax quantization [40] technique. Similarly, we consider several alternative approaches to reduce the number of Gaussians. Densification process in 3DGS increases the Gaussian count and is controlled by the gradient threshold (termed grad thresh) parameter and the frequency (freq) and iterations (iters) until densification is performed. The opacity threshold (min opacity) controls the pruning of transparent Gaussians. We modify these parameters in 3DGS to compress the model with as little drop in performance as possible. Additionally, Table 4.1 shows comparison with state-of-the-art NeRF approaches [12, 53, 133]. Mip-NeRF360 [12] achieves high performance comparable to 3DGS while Plenoxels [53] and InstantNGP[133] have high frame-rate for rendering and very low training time. InstantNGP and Mip-NeRF360 are also comparable in model size to our compressed model.

Evaluation: For a fair comparison, we use the same train-test split as Mip-NeRF360 [12] and

Table 4.2: **Comparison of parameter compression methods for 3DGS.** We evaluate different baseline approaches for compressing the parameters of 3DGS without any reduction in the number of Gaussians. All memory values are reported as a ratio of the method with our smallest model. Our K-Means based vector quantization performs favorably compared to all methods both in terms of novel view synthesis performance and compression. Not quantizing the position values (Int-x no-pos) is crucial in bit quantization. Since harmonics constitute 76% of each Gaussian, 3DGS-no-SH achieves a high level of compression. But CompGS with only quantized harmonics achieves similar compression with nearly no loss in performance compared to 3DGS .

	Mip-NeRF360			Tanks&Temples			Deep Blending			
Method	SSIM	PSNR	LPIPS	SSIM	PSNR	LPIPS	SSIM	PSNR	LPIPS	Mem
3DGS	0.813	27.42	0.217	0.844	23.68	0.178	0.899	29.49	0.246	20.0
3DGS-No-SH	0.802	26.80	0.229	0.833	23.16	0.190	0.900	29.50	0.247	4.8
Post-train K-means 4K	0.768	25.46	0.266	0.803	22.12	0.226	0.887	28.61	0.268	1.7
K-means 4K Only-SH	0.811	27.25	0.223	0.842	23.57	0.183	0.902	29.60	0.246	4.8
K-means 4K	0.804	26.97	0.234	0.836	23.31	0.194	0.904	29.76	0.248	1.7
K-means 32K	0.808	27.16	0.228	0.840	23.47	0.188	0.903	29.75	0.247	1.8
Int16	0.804	27.25	0.223	0.836	23.56	0.185	0.900	29.49	0.247	10.0
Int8 no-pos	0.812	27.38	0.219	0.843	23.67	0.180	0.900	29.47	0.247	5.8
Int8	0.357	14.41	0.629	0.386	12.37	0.625	0.709	21.58	0.457	5.0
Int4 no-pos	0.489	17.42	0.525	0.488	12.94	0.575	0.746	19.90	0.446	3.4
3DGS-No-SH Int16	0.789	26.59	0.237	0.826	23.04	0.198	0.900	29.50	0.248	2.4
K-means 4K, Int16	0.796	26.83	0.239	0.830	23.21	0.199	0.904	29.76	0.248	1.0

3DGS [91] and directly report the metrics for other methods from 3DGS [91]. We also report our reproduced metrics for 3DGS since we observe slightly better results compared to the ones in [91]. We report the standard evaluation metrics of SSIM, PSNR and LPIPS along with memory or compression ratio, rendering FPS and training time. The common practice is to report the average of PSNR across a set of images and scenes. However, this metric may be dominated by very accurate reconstructions (smaller errors) since it is based on the geometric average of the errors due to the log operation in PSNR calculation. Hence, for the larger ARKit dataset, we also report PSNR-AM for which we average the error across all images and scenes before calculating the PSNR. In comparing model sizes, we normalize all methods by dividing them by the size of our method to obtain compression ratio.

Results: Comparison of our results with SOTA novel view synthesis approaches is shown in Table 4.1. Our vector quantized method has a comparable performance to the non-quantized 3DGS with a small drop on MipNerf-360 and TandT datasets and a small improvement on the DB dataset. We additionally report results with post-training bit quantization of our model (CompGS BitQ)

where the position and opacity parameters are quantized to 16 bits and 8 bits respectively. The model memory footprint drastically reduces for CompGS compared to 3DGS , making it comparable to NeRF approaches. Our models are 65× and 54× **smaller** than 3DGS models on MipNerf-360 and TandT datasets respectively. This reduces a big disadvantage of 3DGS models and makes them more practical. The compression achieved by CompGS is impressive considering that more than two-thirds of its memory is due to the non-quantized position and opacity parameters (refer table 4.6). Additionally CompGS maintains the other advantages of 3DGS such as low inference memory usage and training time. while also **increasing its already impressive rendering FPS by** 2× **to** 3×. A limitation of CompGS compared to 3DGS is the overhead in compute and training time introduced by the K-means clustering algorithm. This is compensated in part by the reduced compute and time due to the decrease in Gaussian count. CompGS 16K variant requires marginally more time than 3DGS while CompGS 32K needs 1.4× to 1.7× more training time. However, this is still orders of magnitude smaller than the high-quality NeRF based approaches like MipNerf-360. Per-scene evaluation metrics are in Appendix. Note that there are large differences in reproduced results for 3DGS across various works in the literature. We observe a median standard deviation of 0.05dB for PSNR when the experiment is repeated 20 times with several scenes having differences more than 0.4dB across runs (refer Appendix). One must be careful when analyzing as these variations are often comparable to differences in performance between methods.

We decouple our compression method into parameter and Gaussian count compression components and perform ablations on each of them.

4.3.1 Comparison of parameter compression methods:

In Table 4.2, we compare the proposed vector quantization based compression against other baseline approaches for compressing 3DGS. Since the spherical harmonic components used for modeling color make up nearly three-fourths of all the parameters of each Gaussian, a trivial compression baseline is to use a variant of 3DGS with only the DC component for color and no harmonics. This baseline (3DGS-No-SH) achieves a high compression with just 23.7% of the original model size but has a drop in performance. Our K-Means approach outperforms 3DGS-No-SH while using less than half its memory. We also consider a variant of CompGS with a single codebook for both SH



Figure 4.3: **Qualitative comparison of novel view synthesis approaches**. We visualize images from different scenes across datasets for SOTA NeRF, 3DGS, our CompGS and the No-SH variant of 3DGS . All methods based on 3DGS have better reconstruction of finer details like spokes of the bicycle wheel compared to NeRF approaches. Both compressed versions CompGS and 3DGS-No-SH are similar in appearance to 3DGS with no additional visually apparent errors.

and DC parameters (termed SH+DC) with a larger codebook of size of 4096. This has a marginal decrease in both memory and performance compared to default CompGS suggesting that correlated parameters can be combined to reduce the number of indices to be stored.

Fig. 4.3 shows qualitative comparison of CompGS across multiple datasets with both SOTA approaches and compression methods for 3DGS . Both CompGS and 3DGS-No-SH are visually similar to 3DGS, preserving finer details such as the spokes of the bike and bars of dish-rack. Among NeRF approaches, Mip-NeRF360 is closest in terms of quality to 3DGS while InstantNGP trades-off quality for inference and training speed.

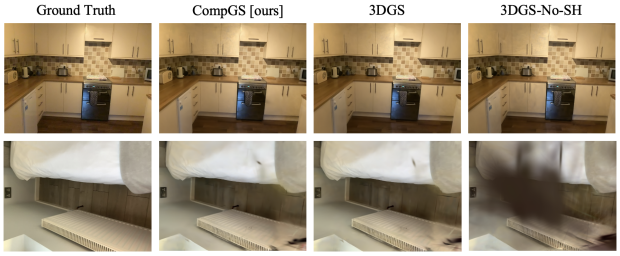
All the above approaches are trained using 32-bit precision for all Gaussian parameters. Post-training bit quantization of 3DGS to 16-bits reduces the memory by half with very little drop in performance. However, reducing the precision to 8-bits results in a huge degradation of the model. This drop is due to the quantization of the position parameters of the Gaussians. Excluding them from

Table 4.3: **Reducing number of Gaussians in 3DGS.** We evaluate different baseline approaches for compressing 3DGS by reducing the number of Gaussians. Gaussian count is proportional to model size. CompGS performs favorably compared to all methods both in terms of novel view synthesis performance and compression.

	Mip-NeRF360				Tanks&Temples				Deep Blending			
Method	SSIM	PSNR	LPIPS	#Gauss	SSIM	PSNR	LPIPS	#Gauss	SSIM	PSNR	LPIPS	#Gauss
3DGS	0.813	27.42	0.217	**3.30*M***	0.844	23.68	0.178	**1.83*M***	0.899	29.49	0.246	**2.80*M***
Min Opacity	0.802	27.12	0.244	1.46M	0.833	23.44	0.204	780K	0.902	29.50	0.255	1.01M
Densify Freq	0.794	26.98	0.255	1.07M	0.832	23.36	0.206	709K	0.902	29.76	0.258	844K
Densify Iters	0.780	27.02	0.267	1.12M	0.835	23.55	0.194	810K	0.896	29.42	0.264	795K
Grad Thresh	0.769	26.57	0.292	809K	0.825	23.31	0.217	578K	0.900	29.49	0.260	1.01M
Opacity Reg	0.813	27.42	0.227	**845*K***	0.844	23.71	0.188	**520*K***	0.905	29.73	0.249	**554*K***

Table 4.4: **Comparison on ARKit-200 dataset.** It contains 200 scenes from the ARKit [13] indoor scene understanding dataset (see the Appendix for details.). We report results for just the vector quantized version of CompGS. (left) CompGS achieves a high level of compression with nearly identical metrics for view synthesis. (right) 3DGS-No-SH fails to reconstruct well in several images while CompGS is nearly identical to 3DGS with a large reduction in model size.

Method	SSIM	PSNR	PSNR-AM	LPIPS	Mem
3DGS	0.909	25.76	20.73	0.226	20.0
3DGS-No-SH	0.905	25.31	20.11	0.234	4.8
CompGS	0.909	25.70	20.73	0.229	1.7



quantization (denoted as Int8 no-pos) results in a model comparable to the 32-bit variant. However, further reduction to 4-bits degrades the model even when the position parameters are not quantized. Note that bit quantization approaches offer significantly lower compression compared to CompGS and they are a subset of the possible solutions for our vector quantization method. Similar to 3DGS, CompGS has a small drop in performance when 16-bit quantization is used.

	DL3DV-140					
Method	SSIM$^{\uparrow}$	PSNR$^{\uparrow}$	PSNR-AM$^{\uparrow}$	LPIPS$^{\downarrow}$	FPS	Mem(MB)
3DGS *	0.905	29.06	27.37	0.134	282	291
CompGS 32K	0.895	28.42	26.97	0.149	566	10

Table 4.5: Results on the 140 scenes NVS benchmark of DL3DV-10K[120] dataset. Similar to the results on the smaller benchmarks, CompGS compresses 3DGS by nearly 30 times with a small drop in reconstruction quality. * is our reproduced results.

4.3.2 Comparison of Gaussian count compression methods:

In Table 4.3, we compare the proposed opacity regularization method for reducing the Gaussian count with baselines. In these baselines, we modify the 3DGS parameters to decrease densification and increase pruning and thus reduce the number of Gaussians. We report the best metrics for each baseline here (refer Appendix for ablation). Our opacity regularization results in 3.5× to 5× reduction in Gaussian count with nearly identical performance as the larger models. Similar level of compression is achieved only by the gradient threshold baseline that reduces densification. However, it results in a large drop in performance.

4.3.3 Results on ARKit-200 and DL3DV datasets:

Table 4.4 shows the quantitative and qualitative results on our large-scale ARKit-200 benchmark. Our compressed model achieves nearly the same performance as 3DGS with ten times smaller memory. Unlike CompGS , the 3DGS-No-SH method suffers a significant drop in quality. We also report PSNR-AM as the PSNR calculated using arithmetic mean of MSE over all the scenes in the dataset to prevent the domination of high-PSNR scenes. Similarly, Table 4.5 shows the performance of CompGS with both KMeans quantization and opacity regularization. CompGS achieves nearly 30× compression compared to 3DGS with a small drop in performance.

Table 4.6: **Breakdown of memory usage in CompGS.** We observe that just 4 non-quantized values of the total 59 values per Gaussian contribute to 68% and 81% of the total memory in our 16-bit and 32-bit variants respectively. For the quantized parameters, nearly the entire memory is used to store the indices.

	Non Quant	Quant
Num Params	4	55
Mem (16bit)	68%	32%
Mem (32bit)	81%	19%

k-Means Quantization	
Index	Codebook
99%	1%
98%	2%

Table 4.7: **Compression performance tradeoff.** Gaussian count decreases drastically with heavy regularization but also results in some drop in performance on Mip-NeRF360 dataset. We choose $\lambda_{\text{reg}} = 10^{-7}$ as default.

λ_{reg} ($\times 10^{-7}$)	SSIM	PSNR	LPIPS	#Gauss
0.5	**0.808**	**27.17**	**0.234**	1.21M
1.0	0.806	27.12	0.240	845K
2.0	0.801	26.98	0.253	536K
3.0	0.794	26.83	0.266	**390*K***

Table 4.8: **Performance and training time trade-off.** Depending on user's needs, it is possible to obtain models with fast training or high performance. The hyperparameters of vector quantization - number of K-Means iterations (iters), K-Means index assignment frequency (freq) and codebook size (#codes) can be varied to obtain the desired point on the curve. They offer a good trade-off, with huge decrease in training time with minor changes in performance. Results are shown on MipNerf-360.

Iters	Freq	#Codes	SSIM	PSNR	Time
1	100	8K	0.802	26.94	19.3
3	100	8K	0.802	26.94	20.9
5	100	8K	0.802	26.95	22.5
10	100	8K	0.802	26.95	26.5
5	50	8K	0.803	27.00	28.7
5	200	8K	0.799	26.76	19.4
5	500	8K	0.783	26.15	**18.1**
5	100	4K	0.800	26.84	19.6
5	100	16K	0.804	27.05	28.9
5	100	32K	**0.806**	**27.12**	42.4

Table 4.9: **Effect of quantization on different Gaussian parameters.** Each Gaussian in 3DGS is parameterized using position (pos), scale, rotation (rot) and color (DC and harmonics SH). We analyze the effect of quantizing combinations of these parameters on the view synthesis performance. SH+DC denotes that a single codebook is used for both SH and DC. Position values cannot be quantized without greatly affecting model performance. The rest of the parameters can be simultaneously combined to obtain a high degree of compression without much loss in quality.

Quantized Params	Train SSIM$^\uparrow$	Train PSNR$^\uparrow$	Truck SSIM$^\uparrow$	Truck PSNR$^\uparrow$	Mem
3DGS	0.811	21.99	0.878	25.38	20.0
3DGS-No-SH	0.798	21.40	0.871	24.92	4.8
Variants of CompGS					
Pos	0.673	19.81	0.730	21.65	19.0
SH	0.809	21.88	0.876	25.27	4.8
SH, DC	0.806	21.68	0.875	25.24	3.8
Rot(R)	0.808	21.83	0.876	25.32	18.7
Scale(Sc)	0.809	21.79	0.877	25.30	19.0
SH,R	0.805	21.67	0.874	25.20	3.5
SH,Sc	0.806	21.63	0.875	25.18	3.8
SH,Sc,R	0.801	21.64	0.872	25.02	2.6
SH+DC,Sc,R	0.797	21.41	0.868	24.89	1.6
SH,DC,Sc,R	0.801	21.64	0.871	24.97	1.7
SH,DC,Sc,R Int16	0.790	21.49	0.869	24.93	1.0

4.3.4 Ablations

We analyze our design choices and the effect of various hyperparameters on reconstruction performance and model size.

Memory break-down of CompGS: In Table 4.6, we show the contribution of various components to the final memory usage of CompGS . Out of 59 parameters of each Gaussian, we quantize 55 parameters of color and covariance while the 3 position and 1 opacity parameters are used as is. However, the bulk of the stored memory (68% and 81% for 16- and 32-bits) is due to the non-quantized parameters. For the quantized parameters, nearly all the memory is used to store the cluster assignment indices with less than 2% used for the codebook.

Trade-off between performance, compression, and training time: Compressing the Gaussian

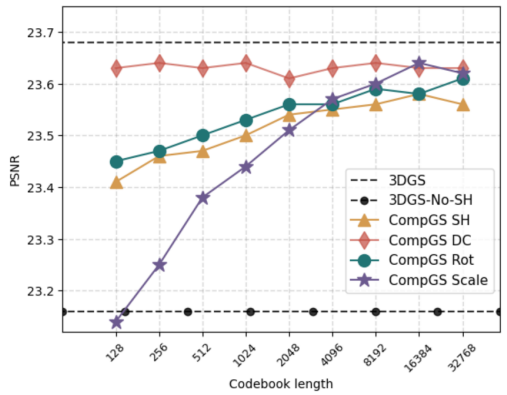


Figure 4.4: **Effect of codebook length.** We vary codebook size while quantizing one of the parameters (SH, DC, Rotation, Scale).

Dataset	Mip-NeRF360		
Method	SSIM↑	PSNR↑	LPIPS↓
3DGS	0.815	27.21	0.214
3DGS *	0.813	27.42	0.217
CompGS 4k	0.804	26.97	0.234
CompGS Shared Codebook	0.797	26.64	0.242

Table 4.10: **Effect of shared codebook.** A frozen codebook trained on one scene ('Counter' scene) generalizes well to all other scenes in MipNerf-360 dataset. Only code assignments are learned during training.

parameters comes with a trade-off, particularly between performance and training time. In our method, the size of codebook, frequency of code assignment and number of iterations in code computation control this trade-off. Similarly, regularization strength can be modified in Gaussian count reduction to obtain a trade-off between performance and compression. We show ablations on these hyperparameters in Tables 4.7 and 4.8. CompGS offers great flexibility, with different levels of compression and training time without sacrificing much on performance.

Parameter selection for quantization: Table 4.9 shows the effect of quantizing different subsets of the Gaussian parameters on the Tanks&Temples dataset. Quantizing the position parameters significantly reduces the performance on both the scenes. We thus do not quantize position in any of our other experiments. Quantizing only the harmonics (SH) of color parameter is nearly identical in size to the no-harmonics (3DGS-No-SH) of 3DGS . Our SH has very little drop in metrics compared to 3DGS while 3DGS-No-SH is much worse off without the harmonics. As more parameters are quantized, the performance of CompGS slowly reduces. The combination of all color and covariance parameters still results in a model with good qualitative and quantitative results.

Effect of codebook size: Fig. 4.4 shows the effect of codebook size for quantization of different Gaussian parameters on the Tanks&Temples dataset. The DC component of color has the smallest drop in performance upon quantization and achieves results similar to the non-quantized version with as few as 128 cluster centers. The harmonics (SH) components of color lead to a much bigger

drop at lower number of clusters and improve as more clusters are added. Note that CompGS with only SH components is nearly the same size as 3DGS-No-SH but has better performance (23.43 for ours vs. 23.14 for 3DGS-No-SH). The covariance parameters (rotation and scale) have a drop in performance at a codebook size of 1024 but improve as the codebook size is increased.

Generalization of codebook across scenes We train our train our method on a single scene ('Counter') of the Mip-NeRF360 dataset. We then freeze the codebook and calculate only assignments for the rest of the eight scenes in the dataset and report the averaged performance metrics over all scenes (Fig. C.3). Interestingly, we observe that the shared codebook generalizes well across all scenes with a small drop in performance compared to learning a codebook for each scene. Sharing learnt codebook can further reduce the memory requirement and can help speed up the training of CompGS. The quality of the codebook can be improved by learning it over multiple scenes.

4.4 Conclusion

3D Gaussian Splatting efficiently models 3D radiance fields, outperforming NeRF in learning and rendering efficiency at the cost of increased storage. To reduce storage demands, we apply opacity regularization and K-means-based vector quantization, compressing indices and employing a compact codebook. Our method cuts the storage cost of 3DGS by almost 45×, increases rendering FPS by 2.5× while maintaining image quality across benchmarks.

Chapter 5

LoRA-merging-aware Training

Parameter-efficient fine-tuning has become the default way to adapt large foundation models to downstream tasks [82, 83, 121, 220]. Among these methods, Low-Rank Adaptation (LoRA) [83] is especially attractive because it represents each task update as a product of two small matrices, enabling efficient storage and training. As public model repositories continue to grow, a natural question follows: can we combine multiple LoRA experts into a single model that inherits their abilities without retraining?

Model merging offers a promising answer. Classical approaches such as Task Arithmetic [87], TIES [211], and DARE [215] combine independently fine-tuned task updates into one multi-task model without access to the original training data. However, recent work [165] has shown that methods developed for fully fine-tuned models do not transfer cleanly to LoRA experts. The difficulty is not merely that task updates disagree at individual parameters. Rather, LoRA experts often learn updates in different low-rank subspaces, so their task vectors are poorly aligned before merging. KnOTS [165] directly diagnoses this issue and improves LoRA merging by aligning task updates post hoc through an SVD-based shared space. TSV [57] similarly shows that task interference can be understood through interactions between singular directions of task matrices, while more recent methods such as Core Space [141] and Essential Subspace Merging [114] further demonstrate that the choice of merging subspace is critical for both accuracy and efficiency.

These works point to a common conclusion: mergeability is fundamentally geometric. A merged

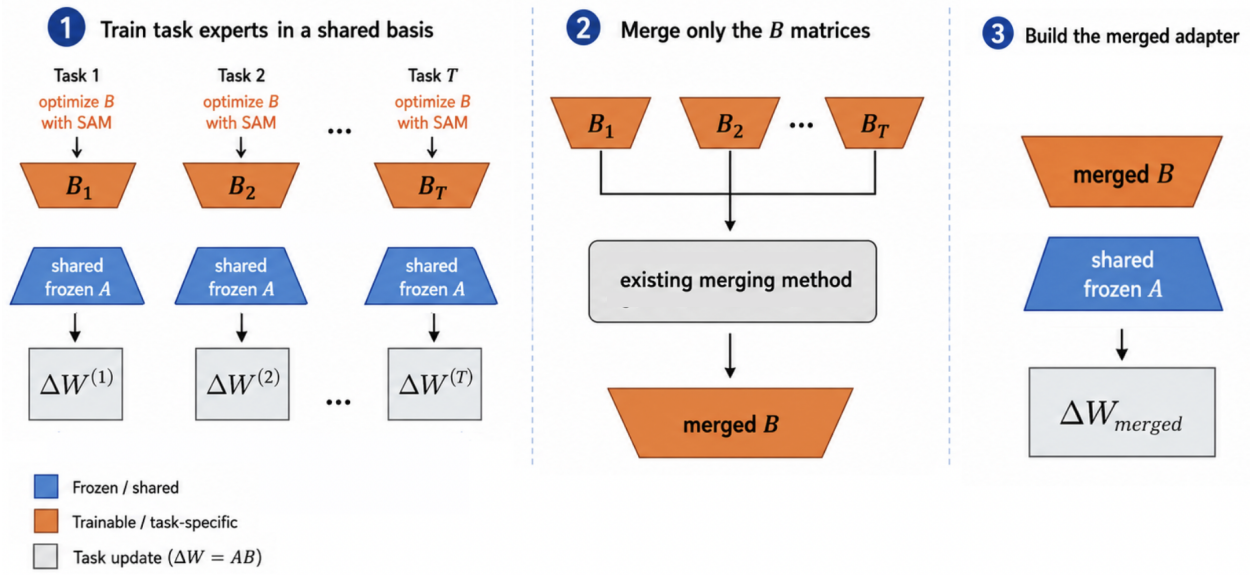


Figure 5.1: **Overview of LoRA-merging-aware Training:** Instead of repairing misalignment after training, we train LoRA experts to share a common subspace from the start. For each task, we fine-tune a LoRA expert while keeping the same randomly initialized low-rank matrix A fixed and shared across all tasks, and optimizing only the task-specific matrix B using SAM. Because every expert is learned in the same common basis, merging can be performed directly on the B matrices with existing merging methods, without needing to align A afterward. The merged adapter is then formed by combining the merged B with the shared frozen A, making the experts more compatible for merging by construction.

model succeeds when task updates are represented in compatible subspaces, and fails when independently trained experts encode their knowledge in conflicting directions. Existing methods mostly address this problem after the experts have already been trained. They first learn task-specific LoRA modules independently, then attempt to repair their geometry through alignment, whitening, orthogonalization, or projection into a shared space. This post-hoc strategy can be effective, but it treats mergeability as an inference-time or merging-time concern rather than a property of the training process itself.

In this work, we take a different view. Instead of asking how to align LoRA experts after training, we ask whether they can be trained to be mergeable from the beginning. Our key idea is simple: all task experts share the same frozen LoRA basis matrix, initialized randomly, while each task learns only its own task-specific coefficient matrix. Because the basis is fixed and shared across experts, all task updates are forced to use the same low-rank coordinate system during fine-tuning. This makes the resulting experts naturally more compatible for merging, since the task-specific components differ only in how they combine a common set of update directions.

Our method is motivated by LoRA-FA [219], which shows that freezing matrix A and optimizing only matrix B can preserve strong fine-tuning performance while significantly reducing activation memory. We reinterpret this asymmetry through the lens of model merging. The frozen factor can serve as a shared basis that constrains all experts to share the same input-side low-rank subspace by construction. To further improve the robustness of the learned task-specific components, we optimize them using Sharpness-Aware Minimization (SAM) [52]. SAM encourages flatter solutions, which we hypothesize are less brittle under merging. Fig 5.1 provides an overview of our approach.

This perspective shifts the goal from designing merging algorithms to learning experts whose updates are naturally compatible. Our approach reduces interference by constraining the geometry of the learned LoRA experts themselves. We evaluate three setups: (1) first multiplying the low-rank matrices and then applying existing merging techniques, (2) merging in the KnOTS space [165], and (3) training LoRA experts with our proposed merging-aware strategy before applying the same merging techniques. Our experiments focus on the language domain using Llama 3 8B. The results show that training experts with a shared frozen LoRA basis and SAM-based optimization improves their compatibility before merging, leading to stronger generalization and task performance.

Author contributions. This chapter is based on my independent work on an idea proposed by H. Pirsiavash. I developed and implemented the method, conducted the experiments, analyzed the results, and wrote the manuscript.

5.1 Related Work

Parameter-efficient fine-tuning (PEFT). These methods adapt large pretrained models by updating only a small number of additional parameters while keeping the backbone frozen [82, 117, 112]. Among them, LoRA [83] has become one of the most widely used approaches, representing each task update with a low-rank decomposition that is cheap to train, store, and share. Many follow-up works improve LoRA through quantization, adaptive rank allocation, weight decomposition, initialization, or gradient approximation [41, 220, 121, 128, 193, 197]. Most relevant to our work is LoRA-FA [219], which freezes one LoRA factor and trains only the other, showing that this asymmetric parameterization can preserve strong fine-tuning performance while reducing memory

cost. We build on this observation, but reinterpret the frozen factor as a shared basis for mergeability rather than only an efficiency mechanism.

Model merging. Model merging combines multiple task-specific models, usually fine-tuned from the same pretrained checkpoint, into a single multi-task model without additional training. Early approaches average model weights directly, as in Model Soups [202], while Task Arithmetic [87] merges task vectors formed by subtracting the pretrained weights from fine-tuned weights. Since direct merging often suffers from task interference, later methods such as TIES [211] and DARE [215] reduce conflicts by trimming, sign resolution, random dropping, and rescaling of task-vector entries. Other methods introduce adaptive coefficients, Fisher-weighted merging, isotropic transformations, or layer-wise scaling to better preserve task knowledge [127, 212, 125, 190]. Recent work further shows that merging quality depends strongly on the geometry of task updates. TSV [57] reduces interference by decorrelating task singular vectors, and Essential Subspace Merging [114] constructs activation-aware subspaces that better preserve task behavior. These works motivate our view that mergeability is not only about how updates are combined, but also about the space in which they are learned.

Merging LoRA models. Merging LoRA experts is harder than merging fully fine-tuned models because each adapter is represented through low-rank factors, and independent experts may learn updates in different subspaces. KnOTS [165] studies this issue directly and shows that LoRA task updates are less aligned than full fine-tuning updates; it improves merging by projecting LoRA updates into a shared SVD-based space before applying standard merging methods. Core Space [141] addresses the efficiency limitation of such full-update alignment by constructing a compact shared basis from the LoRA factors themselves and merging in a low-dimensional core representation. These methods show that shared-space alignment is crucial for LoRA merging. In our work, instead of building the shared space after training, we train all experts in the same randomly initialized frozen basis from the start, so existing merging methods can operate directly on the learned task-specific factors.

5.2 Method

Preliminaries. Consider a pretrained model with a frozen linear layer

$$W_0 \in \mathbb{R}^{d_{\text{out}} \times d_{\text{in}}}.$$

LoRA adapts this layer by adding a low-rank update to the pretrained weight:

$$W_t = W_0 + \Delta W_t, \qquad \Delta W_t = B_t A_t,$$

where

$$A_t \in \mathbb{R}^{r \times d_{\text{in}}}, \qquad B_t \in \mathbb{R}^{d_{\text{out}} \times r},$$

and $r \ll \min(d_{\text{in}}, d_{\text{out}})$. For a task t, standard LoRA optimizes both factors A_t and B_t, while keeping W_0 frozen.

Given T task-specific LoRA experts, model merging aims to construct a single adapter that preserves performance across all tasks. A common approach first reconstructs the full task updates $\Delta W_t = B_t A_t$, then applies an existing merging function $\mathcal{M}$:

$$\Delta W_{\text{merged}} = \mathcal{M}\left(\Delta W_1, \Delta W_2, \ldots, \Delta W_T\right).$$

However, when the LoRA factors are trained independently, each task may learn its own low-rank subspace. As a result, the updates ΔW_t are not necessarily aligned, making direct merging unreliable.

Training LoRA Experts in a Shared Basis Our method constrains all task experts to share the same low-rank basis matrix A. Instead of learning a separate A_t for every task, we randomly initialize one matrix

$$A \sim \mathcal{N}(0, \sigma^2), \qquad A \in \mathbb{R}^{r \times d_{\text{in}}},$$

and keep it frozen throughout training for all tasks. Each task t learns only its own matrix

$$B_t \in \mathbb{R}^{d_{\text{out}} \times r}.$$

The task update is therefore written as

$$\Delta W_t = B_t A.$$

Since every task update is expressed through the same frozen basis A, the task-specific information is isolated in the learned B_t matrices. This makes the resulting LoRA experts naturally more compatible for merging.

SAM Optimization of Task-Specific Coefficients. While the shared basis aligns the update subspaces across tasks, the learned B_t matrices may still be sensitive to small perturbations. This can make the merged adapter unstable. To improve robustness, we optimize each B_t using Sharpness-Aware Minimization (SAM) [52]. SAM encourages solutions with lower neighborhood sharpness that are expected to be less sensitive to the perturbations induced by merging.

For task t, let $\mathcal{L}_t(B_t; A, W_0)$ denote the task loss, where A and W_0 are fixed. SAM seeks parameters that minimize the worst-case loss in a small neighborhood around B_t:

$$\min_{B_t} \max_{\|\epsilon\|_2 \leq \rho} \mathcal{L}_t(B_t + \epsilon; A, W_0),$$

where ρ controls the perturbation radius.

Merging Shared-Basis LoRA Experts. After training, we obtain a collection of task-specific matrices

$$\{B_1, B_2, \ldots, B_T\},$$

all associated with the same frozen matrix A. Since A is shared, merging can be performed directly

on the B_t matrices:

$$B_{\text{merged}} = \mathcal{M}\left(B_1, B_2, \ldots, B_T\right),$$

where $\mathcal{M}$ can be any existing merging method.

The final merged LoRA update is then constructed as

$$\Delta W_{\text{merged}} = B_{\text{merged}} A.$$

The resulting model uses

$$W_{\text{merged}} = W_0 + B_{\text{merged}} A.$$

5.3 Experiments

We evaluate our method on multi-task natural language inference (NLI) using Llama 3-8B as the base model. Following the setup in KnOTS [165], we fine-tune and merge six PEFT Llama 3-8B models trained on SNLI [19], MNLI [201], SICK [126], QNLI [186], RTE [186], and SCITAIL [93]. Each task is formulated as sentence-pair classification, where the model predicts the relationship between a premise and a hypothesis. The full label space contains three classes: entailment, contradiction, and neutral. Since QNLI, RTE, and SCITAIL contain only two of these labels, we mask the missing class during both fine-tuning and evaluation. In all experiments, we only apply LoRA on the weight matrices in the key, query, and value attention layers and output layers. Unless otherwise specified, each LoRA has a rank $r = 16$.

In our experiments, we use TIES [211] as the base merging method and compare our method against three baselines: 1) TIES merging on standard LoRA experts, 2) KnOTS-TIES [165] merging on standard LoRA experts, and 3) TIES merging on standard LoRA experts trained with a weight decay of 1.0. In the first baseline, we reconstruct each LoRA update by multiplying its low-rank factors and then apply TIES to the resulting task updates. In the second, we use KnOTS to align the standard LoRA updates before applying TIES. In the third, we test whether stronger

regularization alone can improve mergeability, without enforcing a shared basis across tasks. Our method differs from all three settings by changing the training procedure: all experts share the same frozen randomly initialized matrix A, and only the task-specific B_t matrices are optimized with SAM before being merged with TIES.

5.3.1 Per-task Evaluation

We first evaluate the merged model on each dataset independently. In this setting, six task-specific LoRA experts are fine-tuned on different NLI datasets and then merged into a single adapter. The merged model is evaluated separately on each dataset using that dataset's own examples and label space. This evaluation measures how well the merged model preserves the individual abilities of the original task experts.

Table 5.1 reports per-task normalized accuracy relative to the corresponding fine-tuned expert, with the first row showing the absolute fine-tuned accuracies as an upper reference. Applying TIES to standard LoRA experts achieves 59.47% average accuracy, while KnOTS-TIES improves this to 64.12% by aligning the LoRA updates before merging. Strong weight decay does not improve mergeability, reaching only 57.07%, suggesting that regularization alone is not enough. In contrast, our shared-basis training already reaches 63.95% without post-hoc alignment, and adding SAM further improves the average performance, showing that training LoRA experts in a common basis makes them more compatible for merging.

5.3.2 Ablations

We conduct an additional analysis to better understand the behavior of our merging-aware training strategy. We study how TIES, KnOTS-TIES, and our method behave as we increase the number of merged experts.

Effect of the number of merged models. Following the analysis in KnOTS [165], we merge different subsets of the six NLI experts and evaluate the resulting model only on the tasks included in the merge. For each number of merged models $k \in \{2, 3, 4, 5, 6\}$, we report the average normalized accuracy across all task combinations. Figure 5.2 shows that direct TIES merging becomes less effective as more LoRA experts are merged, reflecting the increasing difficulty of combining

Table 5.1: **Per-task evaluation on six NLI datasets.** We merge six LLama3-8B models finetuned with LoRA on different NLI tasks. "Finetuned" refers to the accuracy of each finetuned model on the dataset it was trained on. We report the per-task (including the average) normalized-accuracies across other merging baselines. These describe how close they get to the "Finetuned" accuracy.

Method	Datasets						
	SICK	QNLI	RTE	SNLI	MNLI	SCITAIL	Avg
	Per-Task Absolute Accuracies (%)						
Finetuned	91.35	94.87	89.87	92.49	90.30	96.51	92.56
	Per-Task Accuracies of Merged Models Normalized Against Finetuned Models (%)						
TIES	64.43	57.47	59.68	67.60	39.67	67.98	59.47
KnOTS-TIES	66.97	56.6	67.74	77.52	50.91	65.01	64.12
WD-TIES	61.54	55.20	48.39	64.06	42.45	70.81	57.07
Shared A-TIES	62.30	69.41	55.65	54.80	46.02	95.52	63.95
Shared A+SAM-TIES	63.10	69.85	56.20	58.10	46.60	96.45	65.05

independently trained low-rank updates. KnOTS-TIES improves over TIES by aligning the updates before merging. Our method follows the same trend but remains consistently competitive.

Future Work. The goal of this work was to study whether LoRA experts can be made easier to merge by constraining them to a shared frozen basis during training. Several directions remain open for future investigation. Does the benefit extend to other merging methods such as RegMean, Task Arithmetic, DARE, and DARE-TIES? Does the same training strategy remain effective on vision benchmarks when merging LoRA-adapted ViT-B/L models trained on different image-classification tasks? How does LoRA rank affect the trade-off between individual-task performance and mergeability? It would also be useful to study larger collections of experts, different model families, alternative shared-basis initializations, and other optimization strategies beyond SAM to better understand when training-

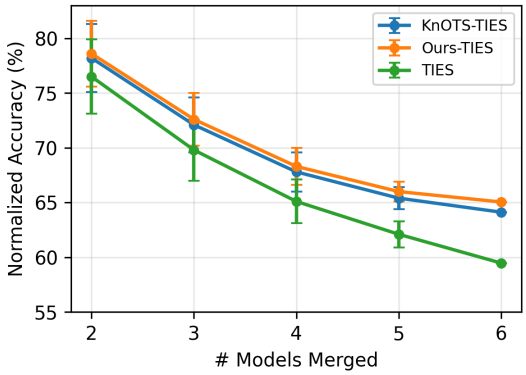


Figure 5.2: **Effect of the number of merged models.** Ours-TIES continues to outperform direct TIES and remains competitive with KnOTS-TIES as the number of merged NLI experts increases. Performance is reported as average normalized accuracy in the per-task evaluation setting, with error bars denoting the standard deviation over different task combinations.

time alignment is most effective.

5.4 Conclusion

We presented a simple way to make LoRA experts easier to merge: use the same randomly initialized frozen matrix A across tasks and train only the task-specific B_t matrices with SAM. This keeps the experts in a common low-rank subspace and avoids having to repair their basis mismatch after training. On six NLI tasks with Llama 3-8B, the method improves direct TIES merging and performs competitively with KnOTS, showing that mergeability can be addressed during fine-tuning rather than only at merge time.

Chapter 6

Discussion

In this dissertation, I studied several ways to improve the efficiency of machine learning systems. The chapters address different stages of the model lifecycle, from learning useful representations to storing, deploying, and combining trained models. Although the settings are different, the common goal is to reduce the resources required to build and use modern machine learning systems without giving up their performance or flexibility.

The first part of the dissertation focuses on supervision and parameter efficiency. CMSF improves representation learning by using distant but semantically related neighbours as training targets. The method works across self-supervised, semi-supervised, and supervised settings, and shows that better use of available supervision can improve performance without requiring much more computation. PRANC studies a different form of efficiency by reparameterizing a neural network as a learned combination of frozen random networks. This allows a large model to be represented using a random seed and a small set of coefficients. CompGS reduces the storage and rendering costs of 3D Gaussian Splatting by combining vector quantization with opacity-based pruning.

The final part of the dissertation considers efficient model adaptation and reuse. LoRA makes it possible to fine-tune large models by training only a small number of parameters, but independently trained LoRA experts can be difficult to merge because they may learn incompatible low-rank subspaces. The merging-aware training method addresses this problem during fine-tuning by placing all experts in a shared frozen basis. This makes the resulting adapters more compatible before

merging and reduces the need for post-hoc alignment.

Together, these works show that efficiency is not limited to reducing the number of floating-point operations. It also depends on how supervision is used, how models are parameterized, how learned representations are stored, and how task-specific knowledge is reused. Several directions remain open for future work.

Scaling compact reparameterization to larger models. PRANC shows that useful solutions can be found in low-dimensional random subspaces, but the cost of generating and combining many basis networks makes the method difficult to apply to very large models. Future work could study layer-wise basis allocation and better progressive reconstruction methods that reduce this cost.

Improving compression of Gaussian Splatting. CompGS leaves position and opacity mostly unquantized because these parameters are more sensitive to compression. As a result, they dominate the remaining model size after the other parameters are compressed. A future direction is to develop scene-dependent quantization. Another direction is extending the method to dynamic scenes, where both the number of Gaussians and their temporal variation create additional compression challenges.

Understanding the relationship between individual performance and mergeability. The LoRA experiments suggest that training experts in a common basis improves merging, but this constraint may also limit how freely each expert can adapt to its own task. Future work should study this trade-off more carefully across LoRA ranks, model families, and more diverse tasks. It would also be useful to test the method with merging approaches beyond TIES.

The broader goal of this dissertation is to make machine learning systems easier to train, store, deploy, and combine under limited resources. The results show that meaningful gains can come from changing the structure of learning itself, rather than only relying on larger models or more computation. Continued work in this direction can help make advanced machine learning more practical and accessible to a wider research community.

Appendix A

CMSF

Here, we provide additional results and analysis on self-supervised (section A.1), semi-supervised (section A.2) and supervised (section A.3) settings. Additional results include analysis of retrieved far neighbors (Figs. A.1) and ablations to justify various design choices (tables A.4, A.5, A.6, A.7, and A.9).

A.1 Results on Self-supervised Setting

A.1.1 Using far neighbors in unconstrained MSF:

In $\text{CMSF}_{\text{self}}$, we use augmented images from previous epoch to obtain distant neighbors. A trivial way to sample farther neighbors is to just increase the number of neighbors k in the original (unconstrained) MSF [98] method. Here we train MSF with $k = 500$ to compare with our $\text{CMSF}_{\text{self}}$. We train all models for 80 epochs. Settings are similar to our self-supervised settings in section 3.1. For fair comparison, we use memory-bank of size $256k$ for MSF while we use $128K$ for $\text{CMSF}_{\text{self}}$. Results are in Table A.1. While increasing k in MSF helps to sample far NNs, it degrades the accuracy. We hypothesize that this happens due to reducing purity of top-k in unconstrained MSF with increasing k (also shown in Fig. 5 of the main submission). This experiment shows that it is not trivial to to sample far NNs with good purity.

Table A.1: **Using far neighbors in unconstrained MSF [98]:** Using far NNs by trivially increasing k in MSF baseline degrades the accuracy. This is due to the low purity of far NNs in MSF when no constraint is utilized. However, $\text{CMSF}_{\text{self}}$ achieves high accuracy while using distant NNs.

	MSF [98] Top-$k = 500$	MSF [98] Top-$k = 5$	$\text{CMSF}_{\text{self}}$ Top-$k = k' = 5$
NN	35.8	49.7	51.4
20-NN	40.2	54.0	55.5

A.1.2 Effect of number of neighbors k

$\text{CMSF}_{\text{self}}$ uses top-k NNs as part of loss calculation. Here we study the effect of k in $\text{CMSF}_{\text{self}}$ performance. We set $k' = k$ in all models. Note that as described in Line 350 of the main submission, k' is the number of NNs retrieved from the second memory bank M'. We train all models for 80 epochs. All settings are similar to that of $\text{CMSF}_{\text{self}}$ in Section 3.1. Results are shown in Table A.2. Higher values of k and k' degrade model performance. We thus use $k' = k = 5$ in all our main experiments.

Table A.2: **Effect of k in top-k NNs sampling within M:** We set $k = k'$ in all models and varied k. We use memory bank of size $256k$. Increasing k degrades the accuracy of the model.

	k'=k=5	k'=k=10	k'=k=20	k'=k=50
NN	51.4	51.3	51.1	49.5
20-NN	55.5	55.3	55.3	53.8

we experimented with ResNet-18 and ResNet-101 networks. ResNet-101 is trained for 100 epochs. CMSF outperforms MSF by 1.9 points (51.7% vs 49.8%) with ResNet-18 and 0.8% points (71.9% vs 71.1%) with ResNet-101.

A.1.3 Results with Different Architectures

In addition to the ResNet-50 architecture, we experiment with a smaller and a larger backbone architecture. We consider ResNet-18 and ResNet-101 networks. The results are shown in table A.3. The proposed $\text{CMSF}_{\text{self}}$ improves over MSF across different architectures. Note that the networks are trained only for 100 epochs on ResNet-101.

Table A.3: **Results with different backbone architectures:** We compare performance of our method with that of CMSF with ResNet architectures of different sizes. We observe that CMSF consistently outperforms MSF. * models were trained for 100 epochs instead of 200.

Method	ResNet-18	ResNet-50	ResNet-101*
MSF [98]	49.8	72.2	71.1
$\text{CMSF}_{\text{self}}$	51.7	73.0	71.9

A.2 Results on Semi-supervised Setting

Unless specified, we use the ImageNet100 dataset for all the ablations on the semi-supervised setting for faster experimentation.

A.2.1 Role of Confidence Threshold in Pseudo-labeling

We use a MLP classification head to predict pseudo-labels for the unlabeled set. As shown in Fig. A.1, the accuracy of the classifier is low in the initial stages and improves as training progresses. Using constraints from incorrectly labeled samples might affect the learning process. Thus, we use confidence (class probabilities) based thresholding to select the samples to be used for pseudo-labeling. Only those samples with confidence higher than the threshold are assigned a pseudo-label. Fig. A.1 shows that the accuracy of the classifier on the confident samples remains high throughout training, limiting the number of incorrect pseudo-labels. Results for threshold value (t) selection are shown in table A.4. As expected, $t = 0$ (i.e, no thresholding) performs poorly compared to higher threshold values. Pseudo-labeling accuracy increases with increasing value of t and the best result is observed for $t = 0.9$. While further increase in t could result in higher pseudo-labeling accuracy, it would also mean that fewer samples are assigned pseudo-labels. Thus, we use $t = 0.9$ in all our experiments on ImageNet100. Since ImageNet-1k has ten times more classes, we reduce the value to 0.85 for all our experiments on ImageNet-1k.

A.2.2 Effect of Caching on Pseudo-label Training

In addition to optimizing the query encoder network using CMSF loss, we train the pseudo-label classifier head at the end of each epoch of query encoder training. Each round of pseudo-label

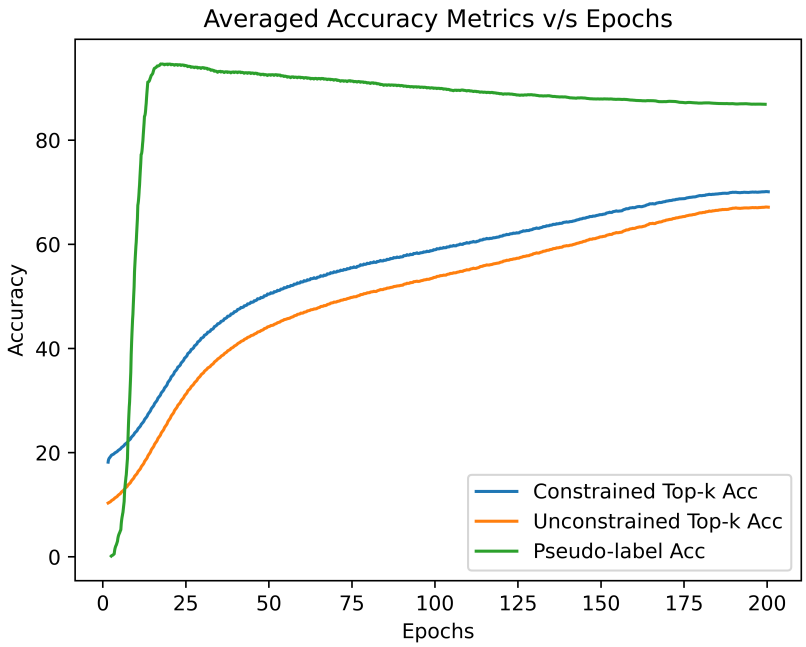


Figure A.1: **Unconstrained NN accuracy in semi-supervised:** For analysis, we track the pseudo-labeling accuracy and accuracy of top-k neighbors chosen with and w/o applying the pseudo-label based constraint during training. The more accurate constrained NNs provide a better training signal. Pseudo-label accuracy on confident samples remains high throughout training, decreasing slightly as more confident samples are added.

Table A.4: **Role of confidence threshold in pseudo-labeling (ImageNet100 results):** Using confidence based threshold to pseudo-label helps improve performance by eliminating noisy pseudo-labels. A higher threshold value results in higher pseudo-label accuracy but also limits the number of samples that participate in constraint selection. We set the value of t to 0.9 on the ImageNet100 dataset and to 0.85 on the more diverse (1000 classes) ImageNet-1k dataset.

Threshold (t)	1-NN	20-NN	Top-1
0	67.9	71.2	76.2
0.7	67.9	72.3	77.1
0.9	**69.0**	**72.7**	**77.5**

Table A.5: **Feature caching for pseudo-label classifier training (ImageNet100 results):** We experiment two different caching schemes for pseudo-label training - offline and online. In offline caching, the features are calculated once at the beginning each round of pseudo-label training while in the online setting, the features are cached for each mini-batch during query encoder training. Since both approaches have similar performance, we use the online version since it has minimal computational overhead.

Method	1-NN	20-NN	Top-1
Offline Caching	67.9	71.2	76.2
Online Caching	66.5	71.9	76.0

training entails 40 epochs of classifier head training on the supervised subset of the data (10%). While the time required for backward pass is minimal since only the MLP head is updated, forward pass through the encoder adds significant computational overhead. We thus employ encoder feature caching to overcome this issue. We experiment with two caching settings - offline caching and online caching. In offline caching, encoder features for all the supervised samples are calculated once at the beginning of pseudo-label training and kept fixed for the remaining 39 epochs. In online caching, encoder features for the supervised samples are cached for each mini-batch during the query network training. Similar to offline caching, these features are then fixed and used throughout the 40 epochs of pseudo-labeling network training. Offline technique requires one epoch of forward pass through the encoder, but has the advantage of using the most recent model parameters for feature calculation. Online caching results in features for different images being calculated using different encoder parameters. We observe that both these settings perform similarly on the ImageNet100 dataset (refer table A.5). We thus use the online version in all our experiments since it has almost no overhead. With this setting, pseudo-label training increases the training time of each epoch approximately by just 40 seconds.

A.2.3 Pseudo-label Classifier Selection

The classifier used to generate pseudo-labels plays a crucial role in obtaining effective constraint sets for $\text{CMSF}_{\text{semi}}$. We experiment with two classification techniques - k-NN classifier and MLP classifier trained with cross-entropy loss. Results on ImageNet100 dataset are shown in table A.6. k-NN classifier has lower pseudo-labeling accuracy and thus results in poorer performance. We

Table A.6: **Pseudo-label classifier selection (ImageNet100 results):** We experiment with different classifier methods and architectures for pseudo-label prediction. Linear layer or multi-layer perceptron (MLP) heads trained using cross-entropy loss on the supervised examples outperform a k-NN classifier. MLP classifiers achieve higher accuracy on the pseudo-labeling task on both the train and test sets. We use a two layer MLP head based classifier in all our experiments.

Pseudo-label Classifier	1-NN	20-NN	Top-1
k-NN Classifier	64.9	69.5	74.7
Linear Classifier	65.6	70.0	75.5
2 Layer MLP Head	67.9	71.2	76.2
3 Layer MLP Head	67.1	71.0	76.1

Table A.7: **Role of network fine-tuning on classification performance (ImageNet-1k results):** We evaluate the trained models using the linear evaluation technique commonly employed for evaluating self-supervised approaches and entire network fine-tuning as performed in semi-supervised methods. Both the methods use 10% of the dataset as supervision. We observe an increase in classification performance when both the encoder and MLP classifier are fine-tuned.

Fine-tune Method	Top-1
Linear layer training	76.5
Full network fine-tune	76.9

additionally experiment with linear, two and three layer architectures for the MLP classifier head. As shown in table A.6, multi-layer head significantly outperform the linear classifier. Since there is minimal difference in performance of two and three layer MLPs, we use a two layer MLP head in all our experiments.

A.2.4 Fine-tuning without Pseudo-labels

Since we do not explicitly optimizer our encoder networks on the label classification task in the pre-training stage, we perform two-stage fine-tuning. We initially fine-tune the pretrained model on only the supervised samples and use the fine-tuned model to obtain pseudo-labels for the unsupervised ones. The combined data is then used to fine-tune the network again. In table A.7, we present results with just a single round of fine-tuning with the 10% supervised samples on ImageNet-1k. Two rounds of fine-tuning provides a small improvement in performance over the single-stage version.

Table A.8: **Supervised learning on coarse grained ImageNet:** We train on the coarse grained version of ImageNet (93 super categories) and perform linear evaluation on the original ImageNet-1k validation set with fine-grained labels (1000 categories). CMSF_{sup} top-10 outperforms all other variants and baselines.

Train Dataset	ImageNet-1k Validation Set				
	Xent	SupCon	CMSF_{sup} top-*all*	CMSF_{sup} top-1000	CMSF_{sup} top-10
ImageNet-1k	77.2	**77.5**	75.7	-	76.4
ImageNet-coarse	61.4	58.7	67.0	71.0	**74.2**

A.3 Results on Supervised Setting

A.3.1 Coarse-grained ImageNet

CMSF_{sup} top-k groups together only top-k neighbors and thus can help in preserving the latent structure of the data compared to top-*all*. To verify this, we consider a dataset with coarse-grained labels where this difference is pronounced. ImageNet dataset was constructed using the WordNet hierarchy. Consider the subtree of WordNet that contains all the 1000 categories from ImageNet-1k as the leaf nodes. To obtain a coarse-grained version, we merge each category in the leaf node to its parent node. After merging, we further ensure that no two of the newly obtained super-classes are in the same path in the graph by merging the descendant into the ancestor class. The total number of classes is thus reduced from 1000 in ImageNet-1k to 93 in our ImageNet-coarse. We train $\text{CMSF}_{\text{self}}$and the baseline approaches in a supervised manner using the coarse labels and then evaluate on the fine-grained (i.e., original) labels on ImageNet-1k validation set. The training settings remain same as that of CMSF_{sup} in Sec.3.2 of main submission.

In Table A.8 we compare the top-*all*, top-1000 and top-k variants on the coarse grained version of ImageNet. We consider the top-1000 variant to limit the effect of dataset imbalance introduced due to the merging of classes. CMSF_{sup} top-k sees a minor drop in performance compared to training on ImageNet-1k. However, methods in which most or all samples in a class are explicitly brought closer - CMSF_{sup} top-*all* and top-1000, cross-entropy and supervised contrastive - see a huge drop in accuracy.

A.3.2 Ablations

We explore different design choices and parameters of our method and baselines. We add the techniques used for our methods to the baselines to isolate the effect of different losses. The results are reported in Table A.9.

Table A.9: **Ablations of baselines and CMSF$_{\text{sup}}$:** All experiments use 200 epochs if not mentioned and use ImageNet-1k dataset. **(a)** More epochs does not improve transfer accuracy for Xent. Thus, the model available from PyTorch [2] (last row) has the best transfer accuracy; **(b)** We add components of our method to improve SupCon baseline. The baseline implementation of SupCon uses std. aug and 16k memory size and it does not include the target embedding u in the positive set. **(c)** We find that our method is not very sensitive to the size of memory bank or top-k in supervised settings; **(d)** Interestingly, excluding the target embedding u from C does not hurts the results. Note that when we do not include the target, the nearest neighbors are still chosen based on the distance to the target, so they will be close to the target.

Method	Mean Trans	Linear IN-1k
(a) Xent		
lr=0.05, cos, epochs=200, strong aug.	71.5	77.2
lr=0.05, cos, epochs=200, std. aug.	71.0	77.3
lr=0.10, cos, epochs=200, strong aug.	72.3	77.1
lr=0.05, cos, epochs=90, std. aug.	72.4	76.8
lr=0.10, cos, epochs=90, std. aug.	74.0	76.7
lr=0.10, step, epochs=90, std. aug.	74.9	76.2
(b) SupCon		
Base SupCon	77.2	77.9
+ change to strong aug.	77.9	77.4
+ add target to positive set	77.8	77.4
+ change to weak/strong aug.	77.8	77.2
+ increase mem size to 128k	78.4	77.5
(c) CMSF$_{sup}$		
top-1 (BYOL-asym)	74.3	69.3
mem=128k, top-2	78.4	76.2
mem=128k, top-10	80.1	76.4
mem=128k, top-20	79.9	76.3
mem=128k, top-*all*	80.1	75.7
mem=512k, top-10	79.9	76.2
mem=512k, top-20	80.1	76.3
(d) CMSF$_{sup}$		
target in top-10	80.1	76.4
target not in top-10	80.3	76.4

Appendix B

PRANC

B.1 Orthogonality and norm of basis networks:

In the main submission, we mentioned that random basis networks are almost perpendicular to each other in high dimensional spaces. To show this, we generate 1000 random vectors at the d dimensional space (varying d from 10 to 1000), and plot the histogram of their ℓ_2 norm and pairwise cosine similarities in Figures B.1 and B.2 respectively. We also run this experiment 100 times, calculate the maximum cosine similarity for each run, and plot the histogram of maximum values in Figure B.3. As expected, at higher number of dimensions, the cosine similarity gets closer to 0 and the norm gets closer to 1. This empirically suggests that at higher dimensions, the random basis are close to orthonormal basis. Please note that our method does not require orthonormal basis to work.

B.2 Reconstruction using a subset of basis:

Figure B.5 shows the distribution of alpha values for a few images from Kodak dataset. As expected, the alpha values vary across the basis models. This motivated us to study "what if we use only a subset of basis models instead of all of them?".

For image classification, we do this experiment by selecting random $p\%$ of the basis with varying p. We repeat this 4 times. As another selection method, we sort alpha values based on their absolute

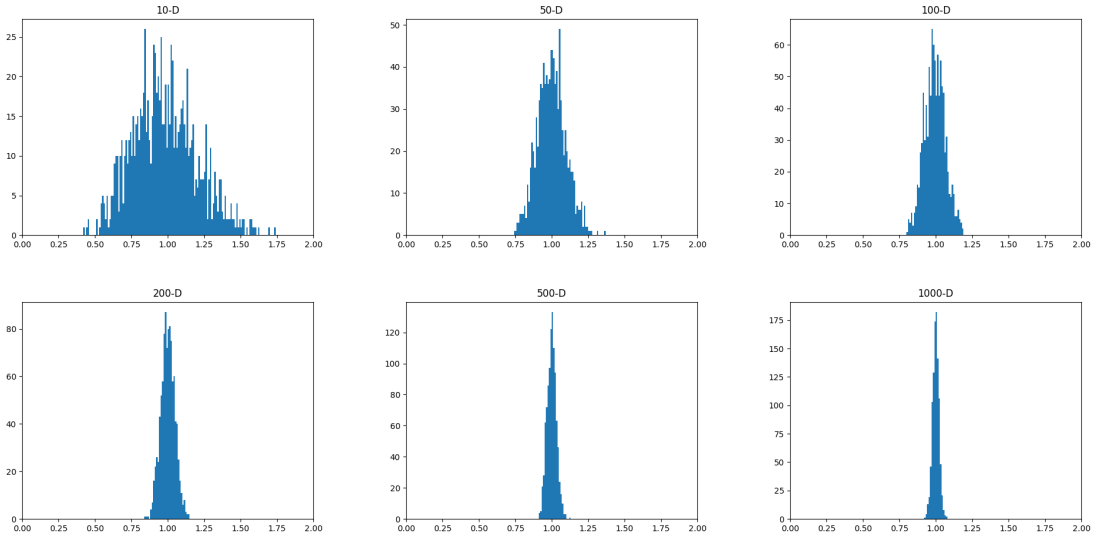


Figure B.1: Histogram of ℓ_2 norm of 1000 randomly generated d dimensional vectors. When increasing d the norm approaches 1.

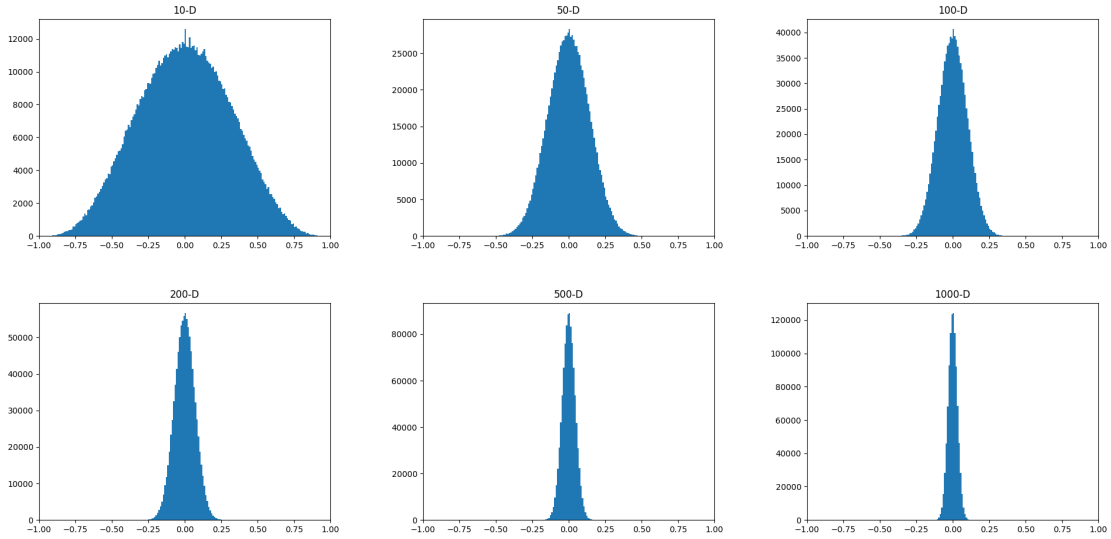


Figure B.2: Histogram of pairwise cosine similarity of 1000 randomly generated d dimensional vectors. When increasing d the cosine similarity approaches 0.

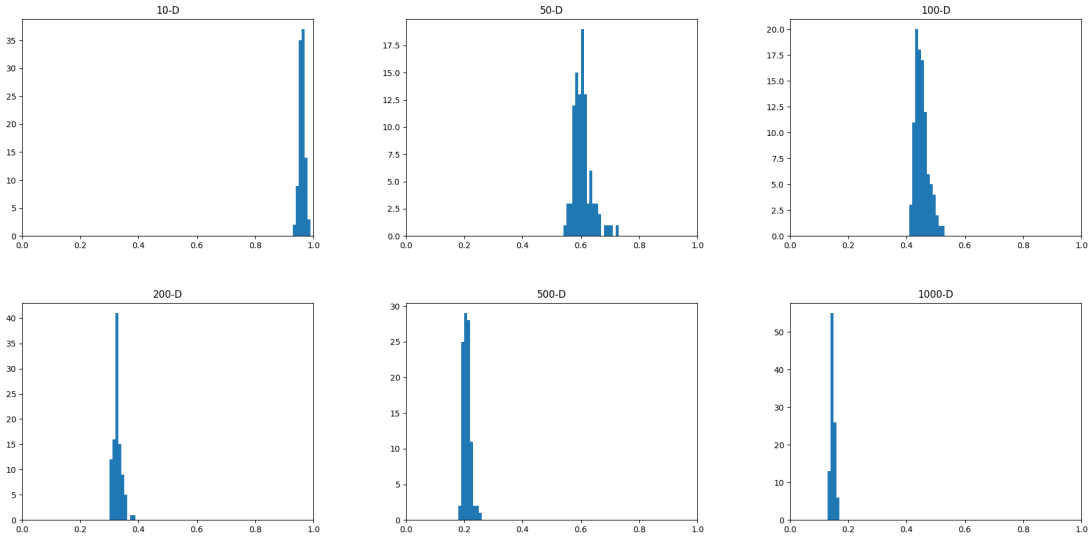


Figure B.3: Histogram of maximum pairwise cosine similarity of 1,000 random d-dimensional vectors over 100 trials. As d increases, the maximum approaches 0

values and use the top $p\%$ of them. As shown in Figure B.4, in the random selection, we need most of the basis to be able to retrieve a reasonable accuracy while for the sorted case, a small percentage of the basis models is sufficient to get a reasonable accuracy. This is an interesting observation that somehow loosely suggests that the loss landscape of the optimization for alpha values is reasonably smooth. This is intentionally a vague statement since it needs further investigation as future work.

For image compression, we perform the same experiment and show reasonable reconstructed images with a subset of alpha values with the largest absolute values. Since we have a different set of alphas for each layer of MLP in image compression, we sort absolute values and select top $p\%$ of each layer with higher values. We vary p and visualize the reconstructed images for each p value in Figure B.7, B.8 and B.9. Additionally, we report the effect of p in PSNR and MS-SSIM for Kodak dataset in Table B.1. Similar to our observation in mage classification, we observe that images with $p = 70\%$ has acceptable quality.

Aside from better understanding the method, this observation can enable communicating a deep model or an image progressively where the sender sends a subset of alpha values (most important ones) first and then gradually sends the rest to improve the model accuracy or the image quality. For image compression, this method is somehow similar to the application of Progressive JPEG where it is hand-crafted to send the low frequency components first. Please note that to use this

in practice, this progressive version of our method has some extra-overhead since we also need to communicate which alpha values are sent at each step (e.g., using a bit for each basis). Studying this in more detail is left for future work.

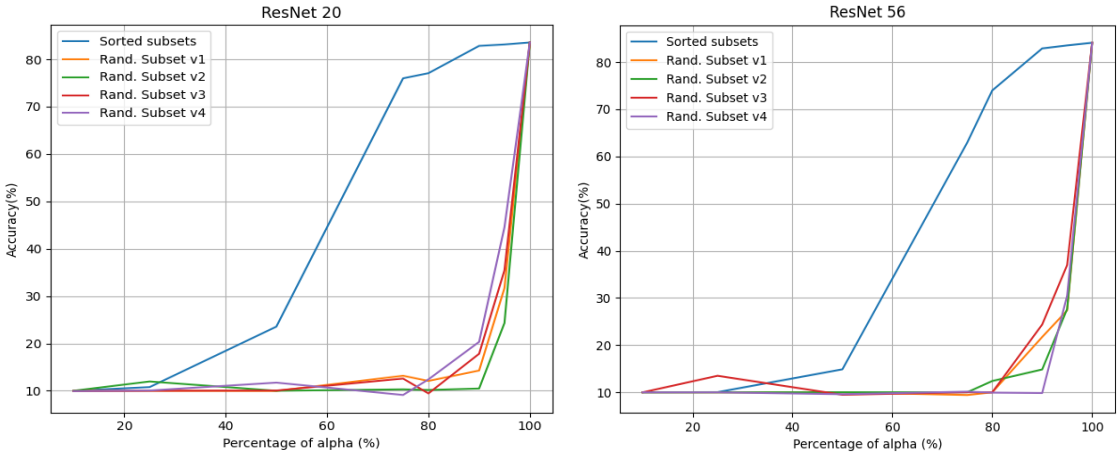


Figure B.4: Effect of using only $p\%$ of basis models selected randomly (4 times) or selected based on highest absolute values of alphas.

B.3 Details of Image Compression:

As described in the main submission, in image compression experiments, in order to change the bit-per-pixel (bpp), we fix the network architecture and vary the number of α values per layer. We report the number of α values per layer for each bpp in Table B.2.

B.4 Compression to Advanced Image Compression methods:

We compare our method with more advanced codecs (e.g., BPG, VTM) and learned-based image compression methods. Results for CLIC-Mobile are in Table B.3 and the results for Kodak are in Figure B.6. Note that advanced codecs are heavily hand-crafted by a large community. And, learned-based methods utilize a training dataset to learn a good code (similar to auto-encoder), hence, they may not be able to compress a single image without having access to a corpus of images. In contrast, our method can compress a single image without using a population of images. Moreover, for the same reason, our method cannot be biased towards the popular cases (head of distribution). Obviously, our method has other biases (e.g., what can be represented with INR)

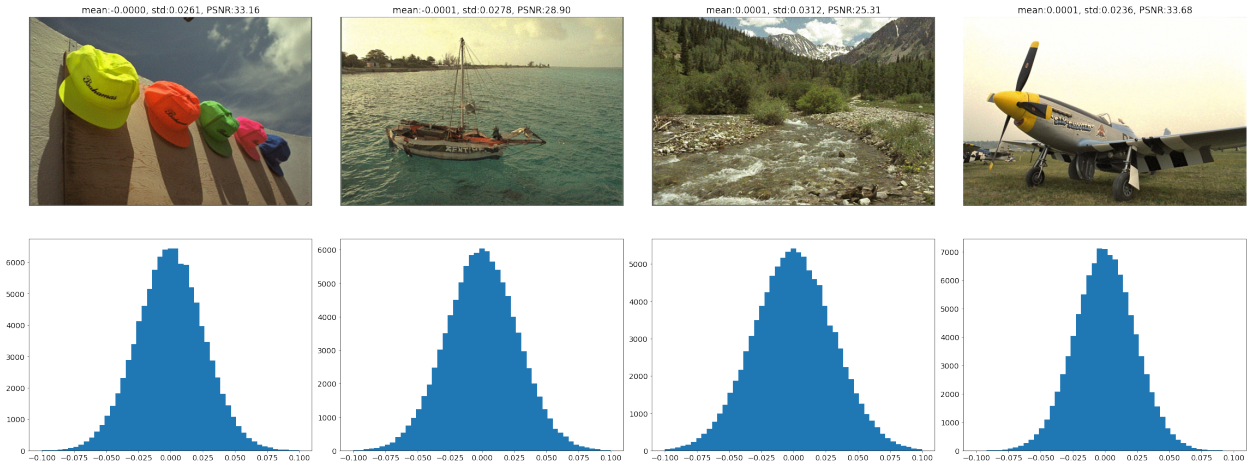


Figure B.5: **Distribution of alphas:** We plot the distribution of alpha values for a few Kodak images.

Table B.1: **Effect of keeping $p\%$ of α with highest absolute value:**

percentile ($p\%$)	10	20	30	40	50	60	70	80	90	100
bpp	0.07	0.14	0.22	0.29	0.36	0.43	0.50	0.58	0.65	0.72
PSNR	11.94	12.3	13.4	14.98	17.01	19.51	22.71	26.92	31.85	33.64
MS-SSIM	0.10	0.12	0.18	0.28	0.41	0.55	0.71	0.86	0.96	0.97

that needs to be studied as the future work.

B.5 Visual Comparison to JPEG:

Similar to Figure 5 of main submission, we include more visual comparison to JPEG in Figures B.10 through B.13 (for Kodak dataset with 512×768 resolution) and Figures B.14 and B.15 (for CLIC-Mobile dataset with 1512×2016 or 2016×1512 resolution). Moreover, in Figures B.16 through B.18, we include visual comparison in chest x-ray dataset with 1024×1024 resolution.

Table B.2: **Details of image compression models:** We report the number of α values per layer for each bpp. We use MLP with hidden dimension of 256 for Kodak dataset at bpp of 0.18 (first row), and hidden dimension of 512 for all other experiments. All settings use Fourier mapping of size 512. We use fewer number of alpha values for the last layer since the last layer has fewer number of weights as it goes from hidden layer to only 3 dimensions (RGB values). Also, for the first row, we use more number of alpha values for first layer since it has more number of weights (512×256).

Dataset	bpp	Layer-1	Layer-2	Layer-3	Layer-4	Layer-5	Total (αs)
Kodak	0.18	10k	7.5k	7.5k	7.5k	2k	34.5k
	0.31	15k	15k	15k	15k	2k	57k
	0.52	10k	30k	30k	30k	2k	102k
	0.72	20k	40k	40k	40k	2k	142k
CLIC-Mobile	0.12	45k	45k	45k	45k	2k	182k
Chest x-ray	0.15	20k	20k	20k	20k	2k	82k

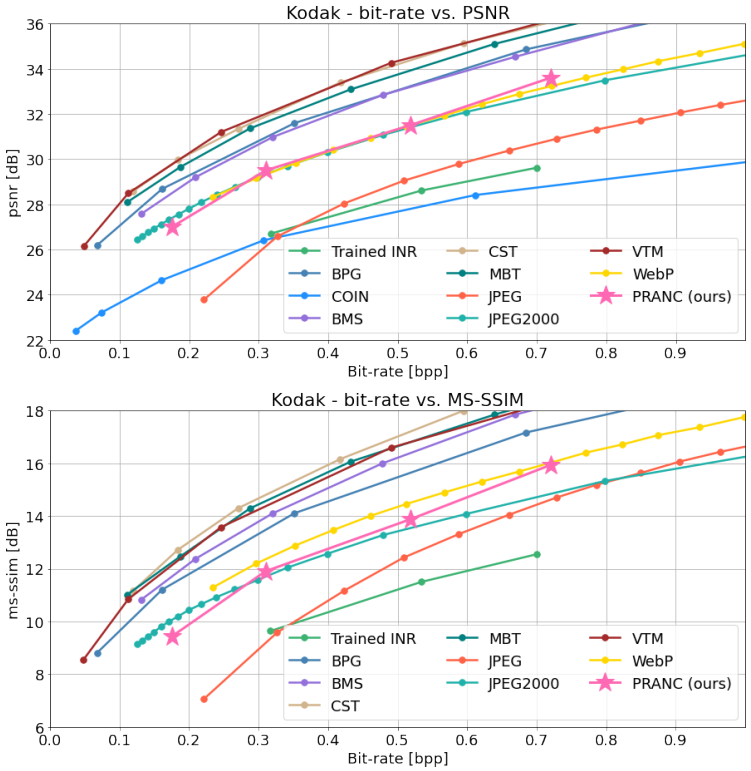


Figure B.6: **Kodak Dataset Image Compression:** Similar to Figure 4 of the main submission, we include comparison to advanced codecs like BPG and VTM. We also compared with learned-based image compression (e.g., MBT, CST, BMS).

Table B.3: **CLIC-mobile Dataset Image Compression:** Similar to Table 6 of the main submission, we include comparison to advanced codecs like BPG and VTM. We also compare with some learning-based image compression methods (e.g., MBT, CST, BMS).

Model	bpp	PSNR	MS-SSIM
VTM	0.183	33.07	0.964
CST	0.146	31.85	0.957
MBT	0.146	31.62	0.955
BPG	0.128	30.65	0.942
WebP	0.185	30.07	0.940
BMS	0.113	29.38	0.936
JPEG2000	0.126	29.40	0.918
JPEG	0.195	24.82	0.836
Trained INR	0.125	26.93	0.864
PRANC (ours)	0.119	29.71	0.920

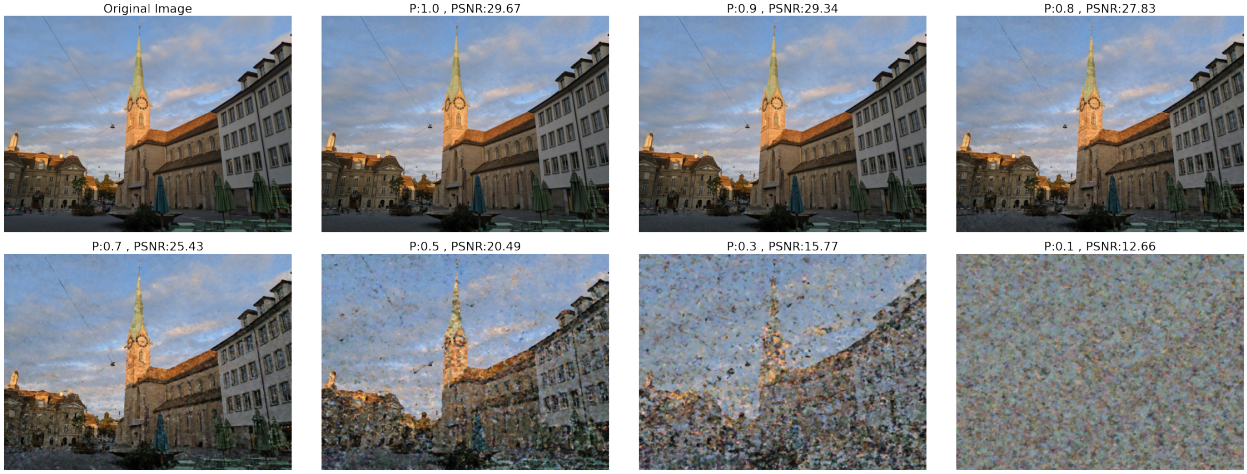


Figure B.7: **Effect of keeping $p\%$ of basis models with the highest absolute alpha values.** We get reasonable images with smaller subset of basis models.

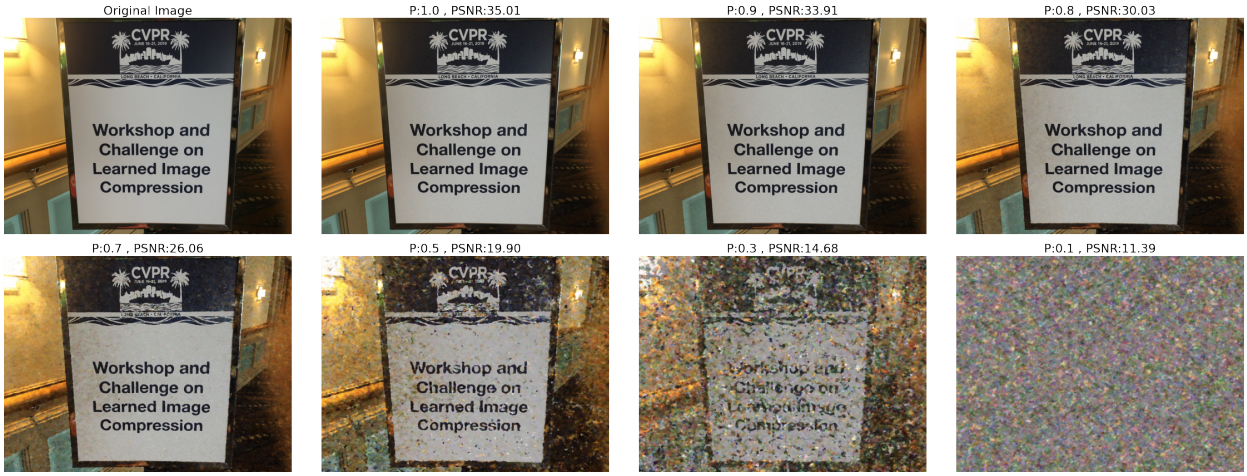


Figure B.8: See Figure B.7.

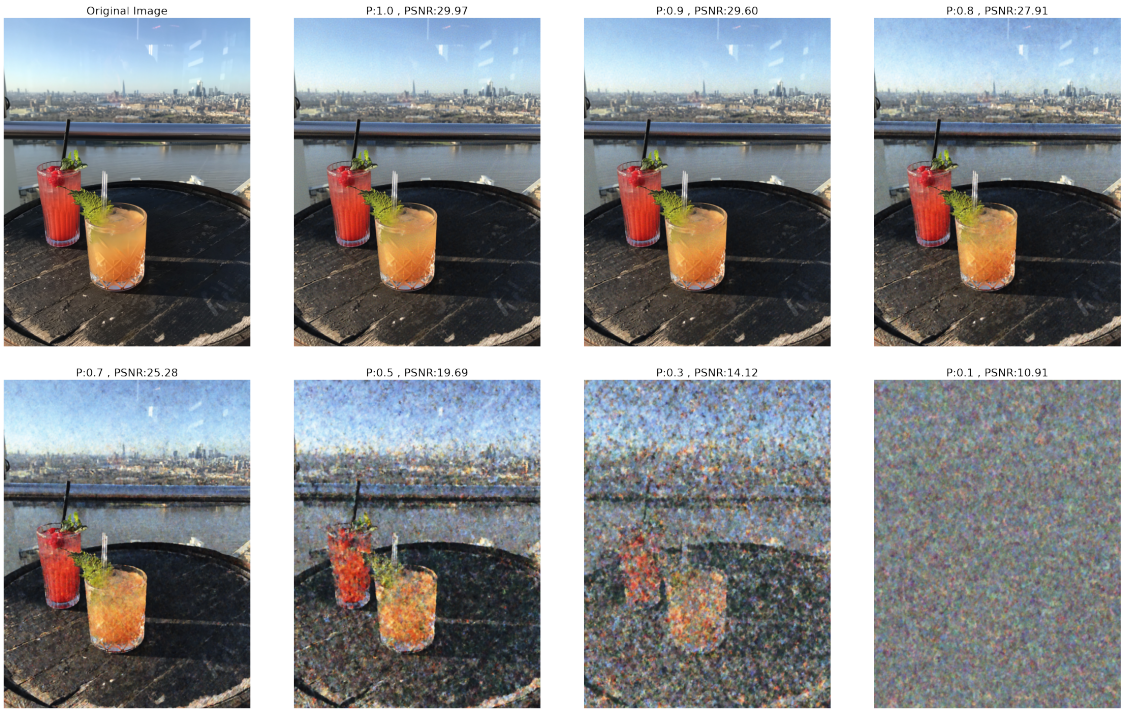


Figure B.9: See Figure B.7.

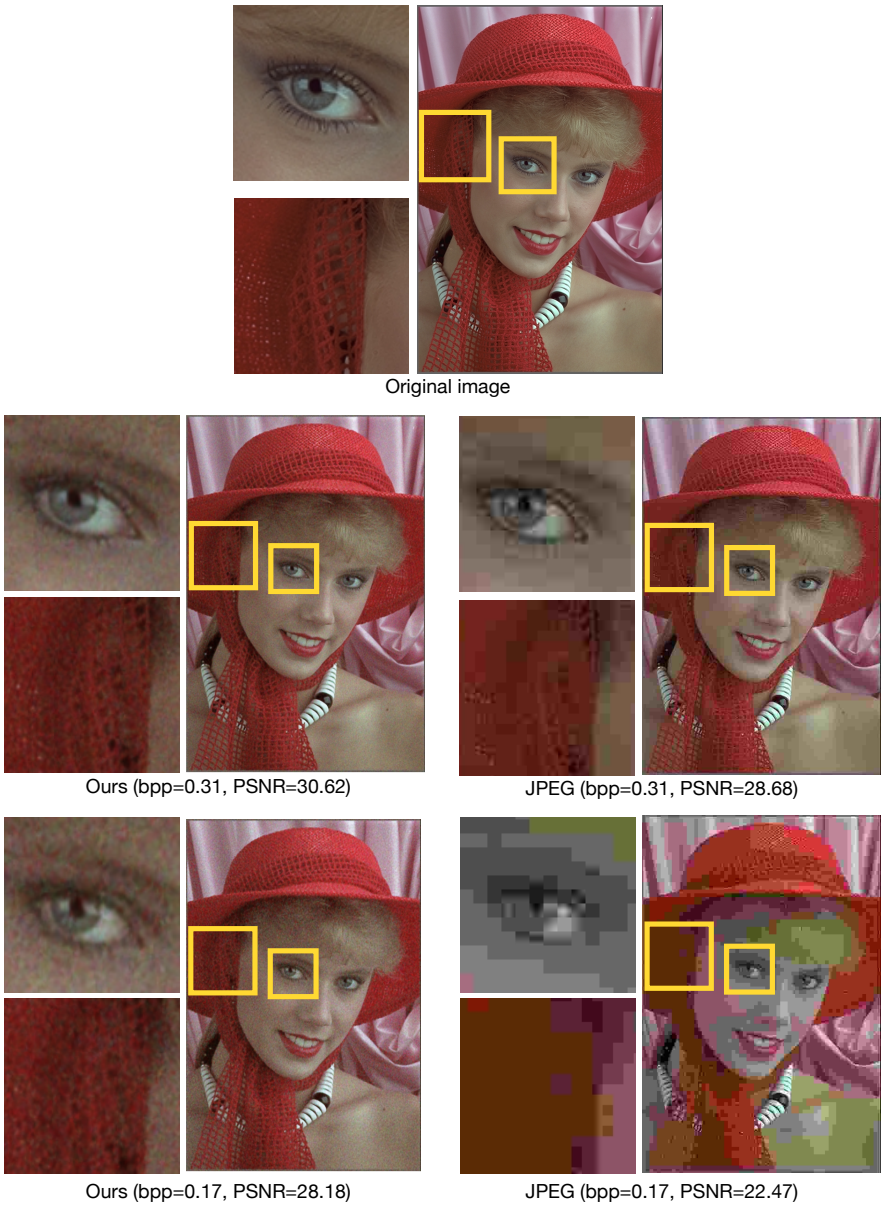


Figure B.10: **Kodak visualization**. We compare PRANC and JPEG on image 4 of Kodak dataset at bpp=0.31 and 0.17

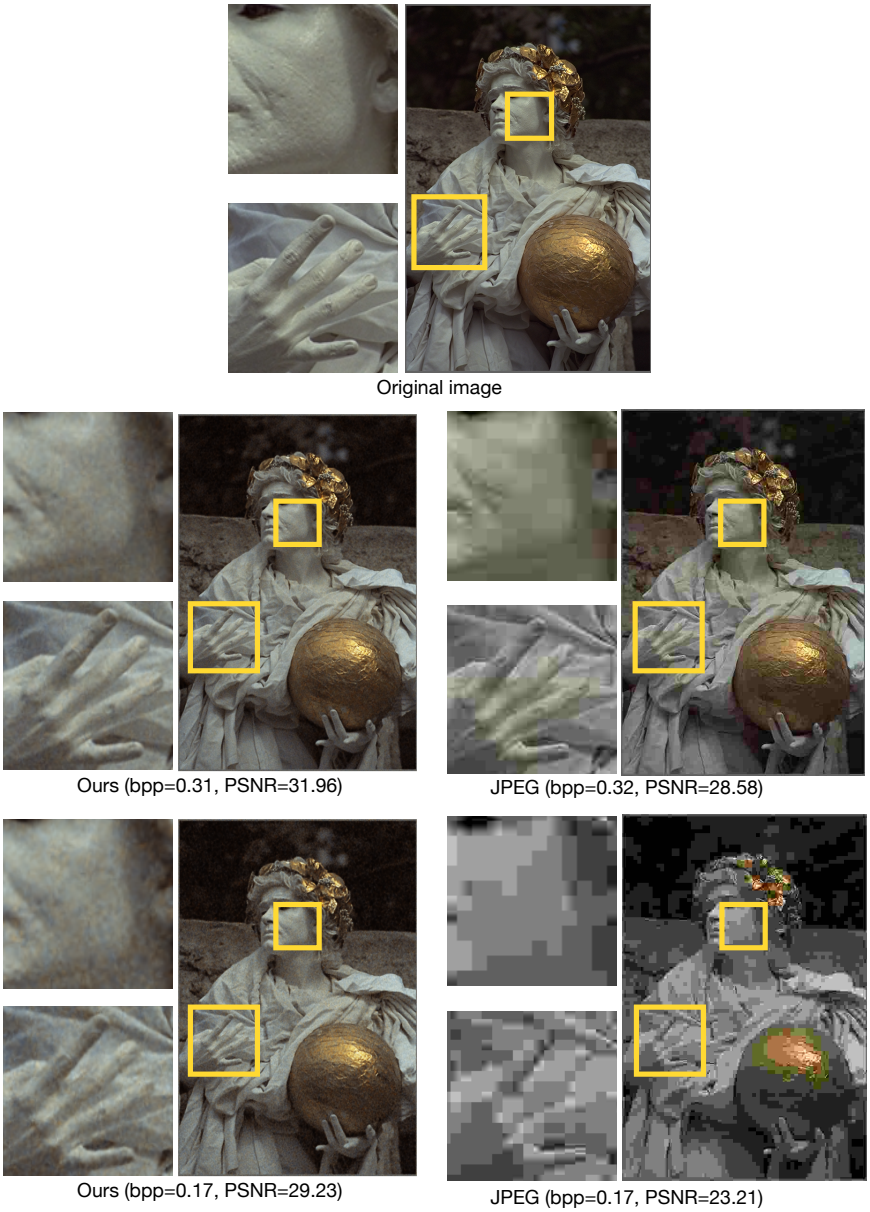


Figure B.11: **Kodak visualization**. We compare PRANC and JPEG on image 17 of Kodak dataset.



Original image



Ours (bpp=0.31, PSNR=25.57)



JPEG (bpp=0.31, PSNR=21.34)



Ours (bpp=0.17, PSNR=23.3)



JPEG (bpp=0.21, PSNR=19.42)

Figure B.12: **Kodak visualization**. We compare PRANC and JPEG on image 5 of Kodak dataset.



Original image



Ours (bpp=0.31, PSNR=31.41)



JPEG (bpp=0.32, PSNR=31.16)



Ours (bpp=0.17, PSNR=27.88)



JPEG (bpp=0.17, PSNR=23.64)

Figure B.13: **Kodak visualization**. We compare PRANC and JPEG on image 23 of Kodak dataset.

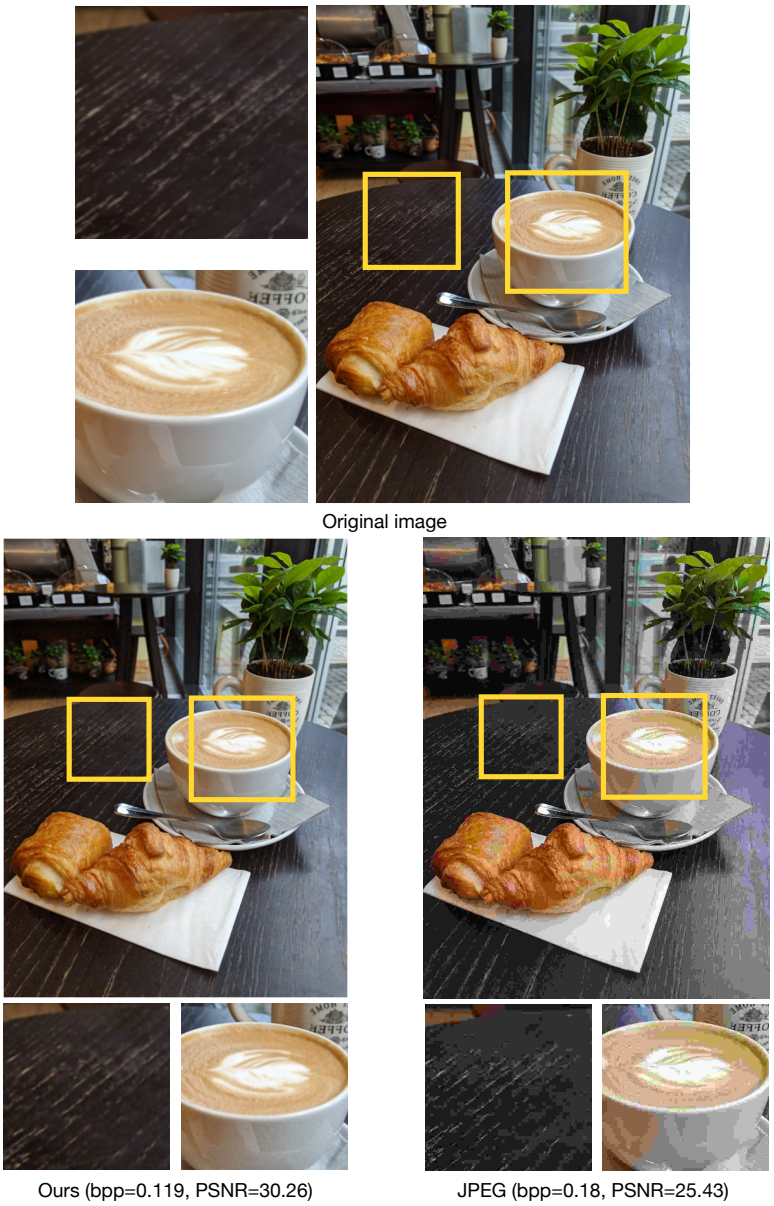


Figure B.14: **CLIC visualization**. We compare PRANC at bpp=0.119 with JPEG at bpp=0.18 on a CLIC image.

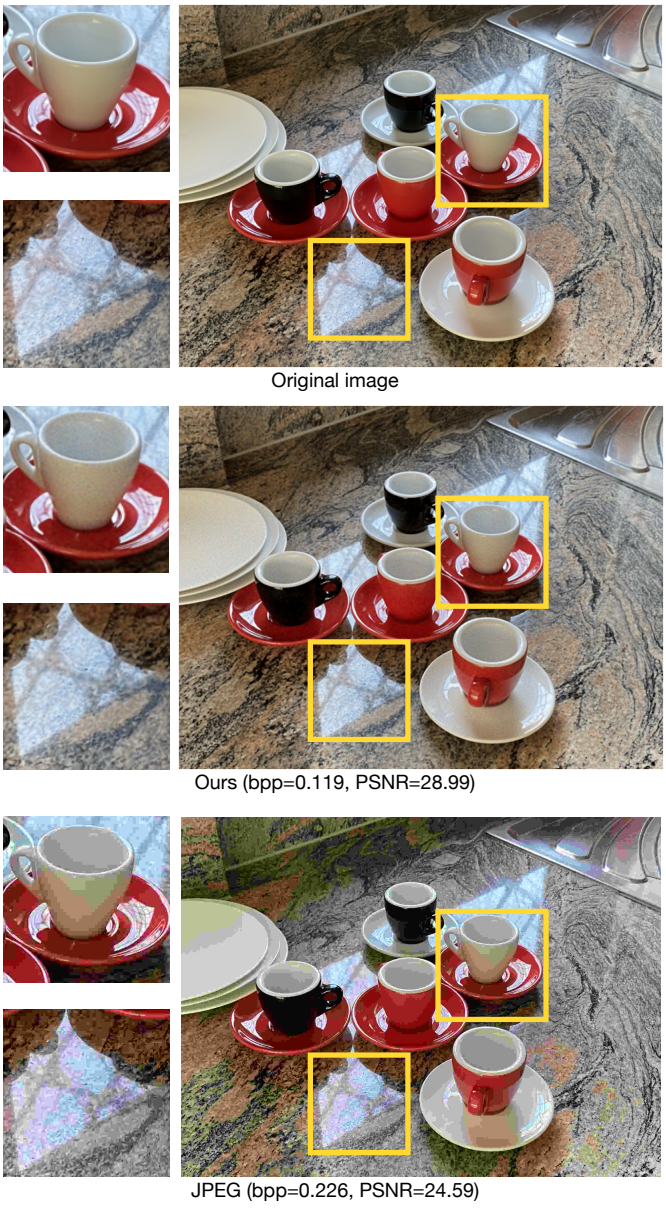
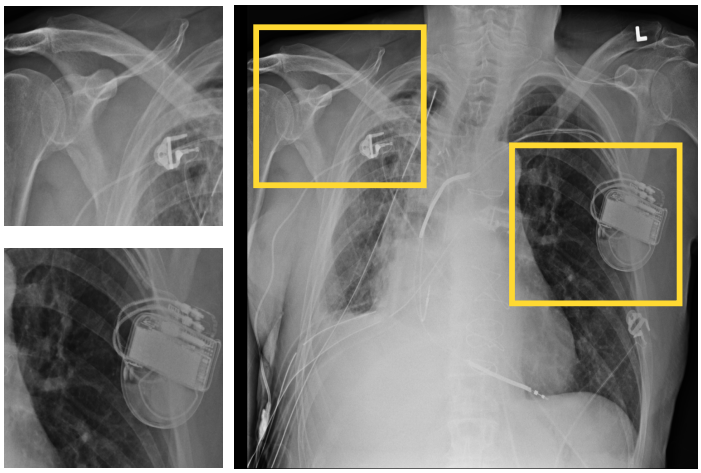
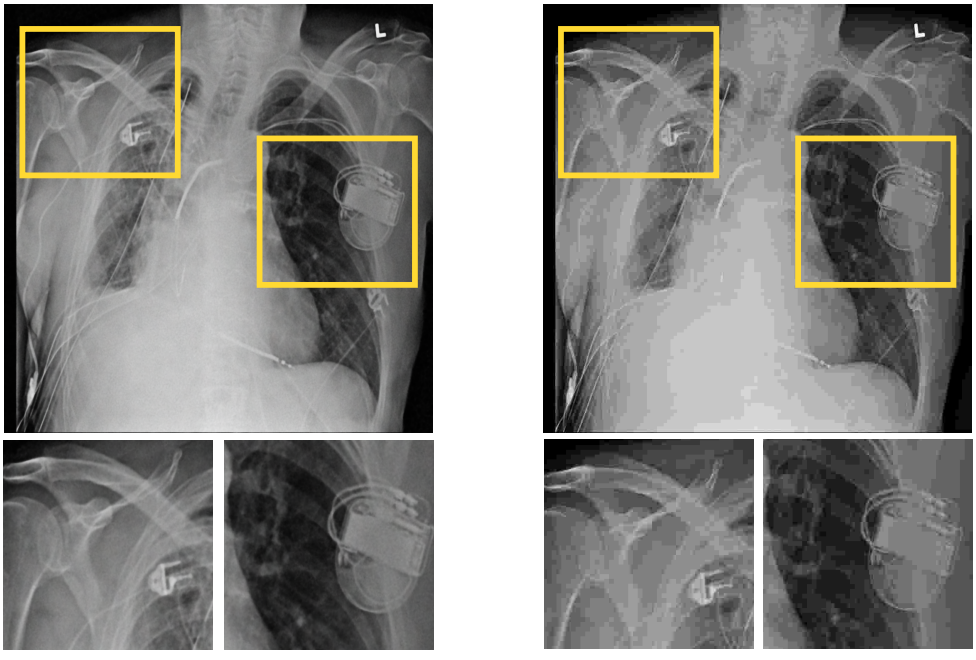


Figure B.15: **CLIC visualization**. We compare PRANC at bpp=0.119 with JPEG at bpp=0.226 on a CLIC image.



Original image



Ours (bpp=0.15, PSNR=36.06)

JPEG (bpp=0.15, PSNR=32.11)

Figure B.16: **Chest X-ray visualization**. We compare PRANC and JPEG on a Chest X-ray image at bpp=0.15

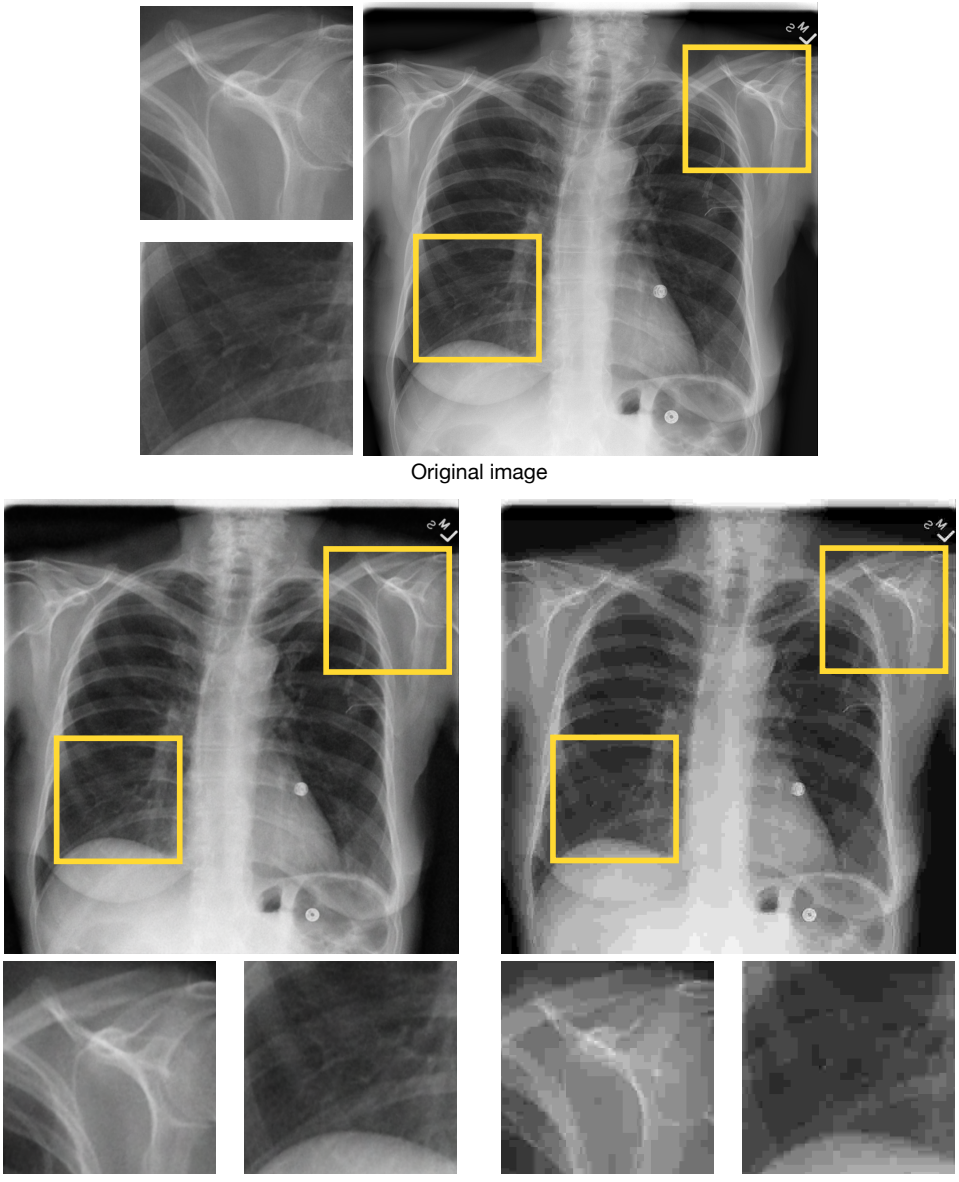
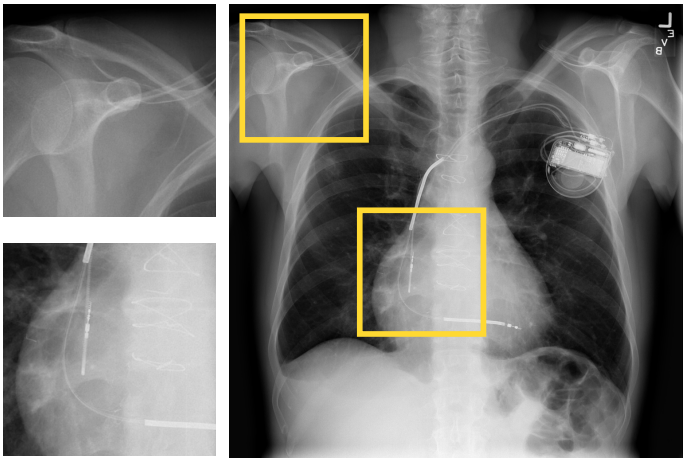
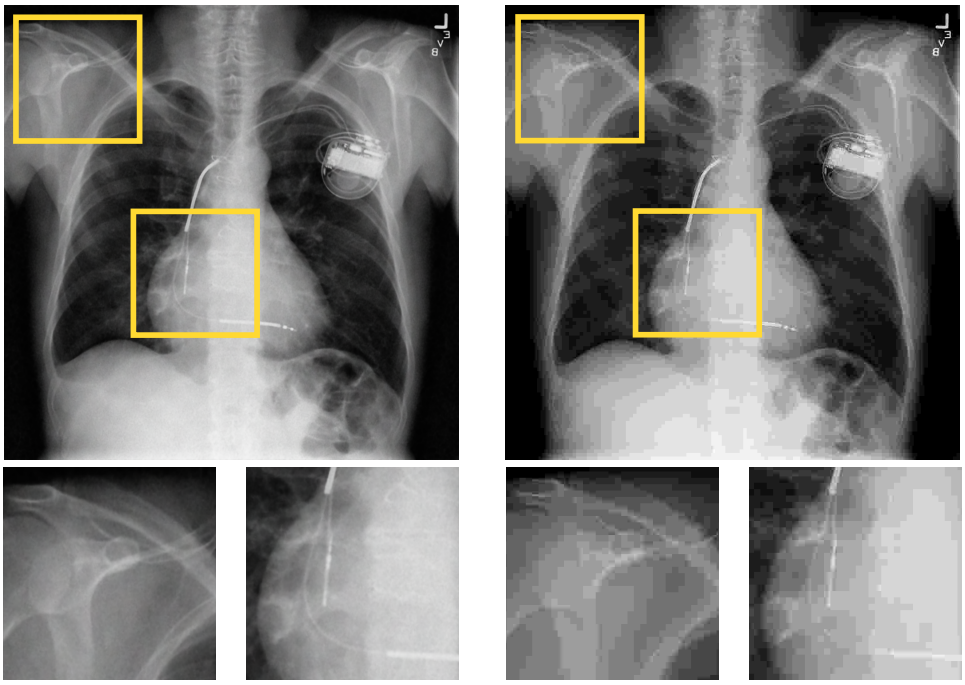


Figure B.17: See Figure B.16.



Original image



Ours (bpp=0.15, PSNR=35.54)

JPEG (bpp=0.15, PSNR=32.20)

Figure B.18: See Figure B.16.

Appendix C

CompGS

In this supplementary pdf, we compare the performance of our CompGS with state-of-the-art approaches on the NeRF-Synthetic dataset (Section C.1). Section C.2 shows exploratory results on generalization of the learnt vector codebook across scenes and Section C.4 provides insights on the learnt codebook assignments. We also provide scene-wise results (section C.5), ablations on baselines (section C.6) and additional visualizations and qualitative comparisons on the ARKit-200 dataset (section C.8). The anonymized code for our approach is provided along with this pdf in the zipped supplementary material file.

C.1 Results on NeRF-Synthetic and DeepBlending Datasets

The results (PSNR) for the NeRF-Synthetic dataset [129] are presented in Table C.1. Our CompGS approach achieves an impressive average improvement of 1.13 points in PSNR compared to the 3DGS-No-SH baseline while using less than half its memory. As reported in the main submission, we report metrics for 3DGS both from the original paper and using our own runs. We observe an improvement for 3DGS [91] over their official reported numbers by 0.5 points. The comparison between our CompGS and SOTA methods for novel view synthesis on Deep Blending dataset is shown in Table C.2. CompGS maintains the speed and performance advantages of 3DGS while being 40× to 50× smaller. CompGS-32K BitQ is the post-training bit quantized version of CompGS-32K, in which position parameters are 16-bits, opacity is 8 bits, and the rest are 32 bits.

Table C.1: **Results on NeRF-Synthetic dataset.** Here, we present the PSNR values for the synthesized novel views on the NeRF-Synthetic dataset [129]. Our CompGS approach achieves an impressive average improvement of 1.13 points in PSNR compared to the 3DGS-No-SH baseline while using less than half its memory. As reported in the main submission, we report metrics for 3DGS both from the original paper and using our own runs. We observe an improvement of 3DGS over the reported numbers by 0.5points. * indicates our own run.

	Mic	Chair	Ship	Materials	Lego	Drums	Ficus	Hotdog	Avg.
Plenoxels	33.26	33.98	29.62	29.14	34.10	25.35	31.83	36.81	31.76
INGP-Base	36.22	35.00	31.10	29.78	36.39	26.02	33.51	37.40	33.18
Mip-Nerf	36.51	35.14	30.41	30.71	35.70	25.48	33.29	37.48	33.09
Point-NeRF	35.95	35.40	30.97	29.61	35.04	26.06	36.13	37.30	33.30
3DGS	35.36	35.83	30.80	30.00	35.78	26.15	34.87	37.72	33.32
3DGS*	36.80	35.51	31.69	30.48	36.06	26.28	35.49	38.06	33.80
3DGS-No-SH	34.37	34.09	29.86	28.42	34.84	25.48	32.30	36.43	31.97
CompGS 4k	35.99	34.92	31.05	29.74	35.09	25.93	35.04	37.04	33.10

C.2 Generalization of codebook across scenes

We train our vector quantization approach including the codebook and the code assignments on a single scene ('Counter') of the Mip-NeRF360 dataset. We then freeze the codebook and learn only assignments for the rest of the eight scenes in the dataset and report the averaged performance metrics over all scenes. In addition to the results in Fig.5 of the main submission, we provide results with our 32K variant here in table C.3. Interestingly, we observe that the shared codebook generalizes well across all scenes with a small drop in performance compared to learning a codebook for each scene. Sharing learnt codebook can further reduce the memory requirement and can help speed up the training of CompGS. The quality of the codebook can be improved by learning it over multiple scenes. Fig. C.3 shows qualitative comparison of the same. There are no apparent differences between CompGS and CompGS-Shared-Codebook approaches.

C.3 Variance in 3DGS Performance

We observe huge differences in the reported performance of 3DGS across different published works. To analyze this, we run the 3DGS baseline method 20 times on all the scenes of Mip-NeRF360 and Tanks&Temples datasets with different seeds and report the statistics of the performance in

Table C.2: **Comparison with SOTA methods for novel view synthesis on Deep Blending dataset.** CompGS is a vector quantized version of 3DGS that maintains the speed and performance advantages of 3DGS while being **40×** to **50× smaller**. CompGS 32K BitQ is the post-training bit quantization version of CompGS 32K, in which position parameters are 16-bits, opacity is 8 bits, and rest are 32 bits. *Reproduced using official code. † Reported from 3DGS [91]. Our timings for 3DGS and CompGS are reported using a RTX6000 GPU while those with † used A6000 GPU. We boldface entries for emphasis.

	Deep Blending					
Method	SSIM↑	PSNR↑	LPIPS↓	FPS	Mem (MB)	Train Time(m)
Plenoxels† [53]	0.795	23.06	0.510	11.2	2700	27.5
INGP-Base† [133]	0.797	23.62	0.423	3.26	13	6.31
INGP-Big † [133]	0.817	24.96	0.390	2.79	48	8.00
M-NeRF360† [12]	0.901	29.40	0.245	0.09	8.6	48h
3DGS † [91]	0.903	29.41	0.243	137	676	36.2
3DGS * [91]	0.899	29.49	0.246	**151**	**662**	19.3
LigthGaussian [48]	-	-	-	-	-	-
CGR [109]	0.900	29.73	0.258	181	23.8	-
CGS [136]	0.898	29.38	0.253	-	25.30	-
CompGS 16K	0.906	29.90	0.252	**485**	**12**	20.3
CompGS 32K	0.907	29.90	0.251	484	13	26.2
CompGS 32K BitQ	0.907	29.89	0.251	**484**	**8**	26.2

table C.4. We only report the PSNRs for brevity. Some of the scenes (Bonsai, Kitchen, Room and Train) have a huge difference in performance between their best and worst runs with a PSNR difference more than 0.4. Thus, one must be careful when comparing performances of methods with small differences. The median of the standard deviation of these runs across all scenes is 0.048.

C.4 Analysis of learnt code assignments

In Fig. C.1, we plot the sorted histogram of the code assignments (cluster to which each Gaussian belongs to) for each parameter on the 'Train' scene of Tanks&Temples dataset. We observe that just a single code out of the 512 in total is assigned to nearly 5% of the Gaussians for both the SH and DC parameters. Similarly, a few clusters dominate even in the case of rotation and scale parameters, albeit to a lower extent. Such a non-uniform distribution of cluster sizes suggest that further compression can be achieved by using Huffman coding to store the assignment indices.

Table C.3: **Effect of shared codebook.** We train our vector quantization approach including the codebook and the code assignments on a single scene ('Counter') of the Mip-NeRF360 dataset. We then freeze the codebook and learn only assignments for the rest of the eight scenes in the dataset and report the averaged performance metrics over all scenes. Interestingly, we observe that the shared codebook generalizes well across all scenes with a small drop in performance compared to learning a codebook for each scene. Sharing learnt codebook can further reduce the memory requirement and can help speed up the training of CompGS. The quality of the codebook can be improved by learning it over multiple scenes.

Dataset	Mip-NeRF360		
Method	SSIM$^{\uparrow}$	PSNR$^{\uparrow}$	LPIPS$^{\downarrow}$
3DGS	0.815	27.21	0.214
3DGS *	0.813	27.42	0.217
CompGS 4K	0.804	26.97	0.234
CompGS 32K	0.806	27.12	0.240
CompGS Shared Codebook 4K	0.797	26.64	0.242
CompGS Shared Codebook 32K	0.800	26.780	0.247

Table C.4: **Variance in 3DGS performance:** We run the 3DGS baseline method 20 times on all the scenes of Mip-NeRF360 and Tanks&Temples datasets with different seeds and report the statistics of the performance. We only report the PSNRs for brevity. Some of the scenes (Bonsai, Kitchen, Room and Train) have a huge difference in performance between their best and worst runs (denoted as 'Diff') with a PSNR difference more than 0.4. The median of the standard deviation of these runs across all scenes is 0.048.

Scene	1	2	3	4	5	6	7	8	9	10	11
Min	25.140	31.774	28.854	21.414	27.084	30.864	31.134	26.522	22.418	21.733	25.250
Max	25.271	32.183	29.068	21.562	27.293	31.425	31.545	26.624	22.585	22.193	25.415
Mean	25.221	32.038	28.997	21.491	27.227	31.228	31.427	26.583	22.495	21.940	25.333
Diff	0.131	0.409	0.215	0.148	0.210	0.561	0.410	0.102	0.167	0.460	0.165
Std	0.032	0.109	0.051	0.037	0.048	0.155	0.105	0.024	0.045	0.123	0.046

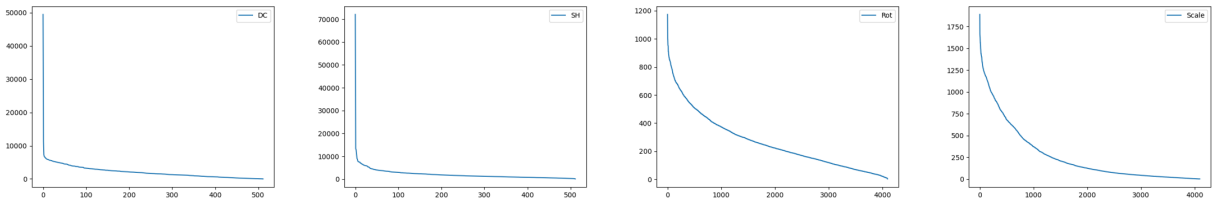


Figure C.1: **Histogram of code assignments.** We plot the sorted histogram of the code assignments (cluster to which each Gaussian belongs to) for each parameter on the 'Train' scene of Tanks&Temples dataset. We observe that just a single code out of the 512 in total is assigned to nearly 5% of the Gaussians for both the SH and DC parameters. Similarly, a few clusters dominate even in the case of rotation and scale parameters, albeit to a lower extent. Such a non-uniform distribution of cluster sizes suggest that further compression can be achieved by using Huffman coding to store the assignment indices.

C.5 Scene-wise Metrics

For brevity, we reported the averaged metrics over all scenes in a dataset in our main submission. Here in table C.5, we provide the detailed scene-wise metrics for MipNerf-360, Tanks and Temples and DeepBlending datasets.

C.6 Ablations for Gaussian count reduction

We perform ablations to choose the right hyperparameters for the baseline approaches to reduce the Gaussian count. The metrics for the chosen settings are reported in table 3 of the main submission. The ablations for minimum opacity, densification interval and end iteration and gradient threshold are shown in tables C.6, C.7, C.8 and C.9 respectively. Among these baselines, modify gradient threshold provides the best trade-off between model size and performance.

C.7 Details of ARKit-200 dataset.

Unlike visual recognition community that uses large scale benchmarks, interestingly, radiance field modeling community traditionally has used small datasets with a handful of 3D scenes: maximum of 13 total real world scenes in several papers (e.g., 3DGS [91]). We believe this is due to the computational cost of NeRF-based methods (several hours of training for each scene). Hence, with the recent advancements in this field including 3DGS that takes only a few minutes to learn a scene, it may be time to move beyond small benchmarks and evaluate methods on larger scale benchmarks. Therefore, we use an existing scene-understanding dataset [14] as a benchmark for radiance field modeling that is an order of magnitude larger than the traditional datasets (200 vs 13 scenes). We believe the community will benefit from using this larger benchmark to evaluate future radiance field methods with a reasonable computational demand. Note that DL3DV-140 is another large dataset that is concurrent work to ours, so we include that one too in the results of our main paper.

ARKit-200 [14] is an indoor scene understanding dataset comprising $5,048$ scans encompassing $1,661$ distinct scenes. The videos are recorded using the 2020 iPad Pro and have a resolution of 1920×1440. We exclusively utilize the RGB frames from each video. To construct the dataset subset for view synthesis, we randomly select 200 raw videos from the ARKit-200 dataset, extracting

Table C.5: **Scene-wise metrics.** We report the scene-wise metrics on all the scenes for both 3DGS and CompGS 32K. As observed in the averaged metrics in the main submission, CompGS achieves a high level of compression and fast rendering without losing much on rendering quality.

Scene	Method	SSIM$^{\uparrow}$	PSNR$^{\uparrow}$	LPIPS$^{\downarrow}$	Train Time(s)	FPS	Mem(MB)	#Gauss
Bicycle	3DGS	0.763	25.169	0.212	1845	68	1422	6026079
	CompGS 32K	0.755	25.068	0.244	2123	242	29	1314018
Bonsai	3DGS	0.940	31.918	0.206	876	276	293	1241520
	CompGS 32K	0.932	31.195	0.223	1405	492	10	377227
Counter	3DGS	0.906	29.018	0.202	968	208	283	1200091
	CompGS 32K	0.895	28.467	0.222	1571	356	10	385515
Flowers	3DGS	0.602	21.456	0.339	1192	133	851	3605827
	CompGS 32K	0.588	21.262	0.367	1744	343	23	1037132
Garden	3DGS	0.863	27.241	0.108	1870	76	1347	5709543
	CompGS 32K	0.848	26.822	0.140	2177	258	30	1370624
Kitchen	3DGS	0.926	31.510	0.127	1206	158	425	1801403
	CompGS 32K	0.918	30.774	0.142	1729	317	13	564382
Room	3DGS	0.917	31.346	0.221	1020	190	364	1541909
	CompGS 32K	0.912	31.131	0.235	1439	444	9	327191
Stump	3DGS	0.772	26.651	0.215	1445	104	1136	4815087
	CompGS 32K	0.770	26.605	0.236	1936	303	27	1226044
Treehill	3DGS	0.633	22.504	0.327	1213	122	879	3723675
	CompGS 32K	0.634	22.747	0.355	1744	336	23	1002290
Train	3DGS	0.811	21.991	0.209	563	253	254	1077461
	CompGS 32K	0.804	21.789	0.231	1169	456	12	500811
Truck	3DGS	0.878	25.385	0.148	897	159	611	2588966
	CompGS 32K	0.872	25.092	0.165	1306	494	13	540081
DrJohnson	3DGS	0.898	29.089	0.247	1312	121	772	3270679
	CompGS 32K	0.906	29.445	0.249	1717	379	17	714902
Playroom	3DGS	0.901	29.903	0.246	1003	181	551	2335846
	CompGS 32K	0.908	30.347	0.253	1422	589	10	393414

Table C.6: **Ablation study on opacity threshold**. We changed the default value of 0.005 for minimum opacity as a baseline approach for compressing 3DGS by reducing the number of Gaussians. Gaussians with opacity values below the threshold are pruned, resulting in smaller models for lower thresholds. Table 3 in main paper shows the results of this experiment when min opacity is 0.1.

	Mip-NeRF360				Tanks&Temples				Deep Blending			
	SSIM	PSNR	LPIPS	#Gauss	SSIM	PSNR	LPIPS	#Gauss	SSIM	PSNR	LPIPS	#Gauss
0.05	0.810	27.308	0.230	1.93M	0.839	23.508	0.194	1.04M	0.902	29.542	0.251	1.47M
0.1	0.802	27.120	0.244	1.46M	0.833	23.439	0.204	780K	0.902	29.504	0.255	1.01M

Table C.7: **Ablation study on densification interval**. We modify the densification interval in 3DGS as a baseline approach for compressing 3DGS by reducing the number of Gaussians. Higher intervals results in less frequent densification and thus smaller number of Gaussians. Table 3 in main paper shows the results of this experiment when interval is 500.

	Mip-NeRF360				Tanks&Temples				Deep Blending			
	SSIM	PSNR	LPIPS	#Gauss	SSIM	PSNR	LPIPS	#Gauss	SSIM	PSNR	LPIPS	#Gauss
300	0.803	27.201	0.241	1.70M	0.837	23.517	0.195	1.00M	0.902	29.705	0.253	1.30M
500	0.794	26.98	0.255	1.07M	0.832	23.36	0.206	709K	0.902	29.76	0.258	844K

a uniform sample of 300 frames from each. Subsequently, we employ the code provided by 3DGS [91] to extract undistorted images and Structure-from-Motion (SfM) information from the input images. The dataset can easily be extended in the future by including more of the remaining scenes from the ARKit dataset.

C.8 Qualitative comparison on ARKit-200 dataset.

Figures C.4 and C.5 provide qualitative results on the ARKit-200 dataset.



Figure C.2: **Visualization of results on Sythetic-NeRF dataset.** We compare the performance of our compressed CompGS with the original 3DGS and 3DGS-No-SH approaches on different scenes of the NeRF-Synthetic dataset. The difference between CompGS and 3DGS-No-SH is apparent in some of these scenes. E.g., 3DGS-No-SH fails to effectively model the brown color of branches and shadows and bright light on the leaves of the 'Ficus' scene. All approaches including 3DGS have imperfect reconstruction in some of the scenes like 'Drums' and 'Lego'. The scenes and views used for visualization were chosen at random.



Figure C.3: **Qualitative analysis of shared codebook.** We show the generalization of codebook learned using a single scene on various scenes of the Mip-NeRF360 dataset. The codebook was trained on the 'Counter' scene (row-1) and frozen for the remaining scenes. The codebooks for all four parameters (DC, SH, Scale, Rot) are shared across scenes. Both CompGS and CompGS-Shared-Codebook are visually similar to the uncompressed 3DGS with no conspicuous differences between them. 3DGS-No-SH requires twice more memory than CompGS while 3DGS is ten times bigger than CompGS . The scenes and views used for visualization were chosen at random.



Figure C.4: **Visualization of ARKit-200 dataset.** ARKit-200 is a 3D indoor scene dataset captured using a iPads/iPhones. The dataset consists of videos of indoor environments like houses and office space from multiple view-points. We uniform sample images from each video to form our benchmark dataset for novel view synthesis. Some sample images from different scenes are shown in this figure. The dataset presents unique challenges such as the presence of motion blur due to the use of videos.

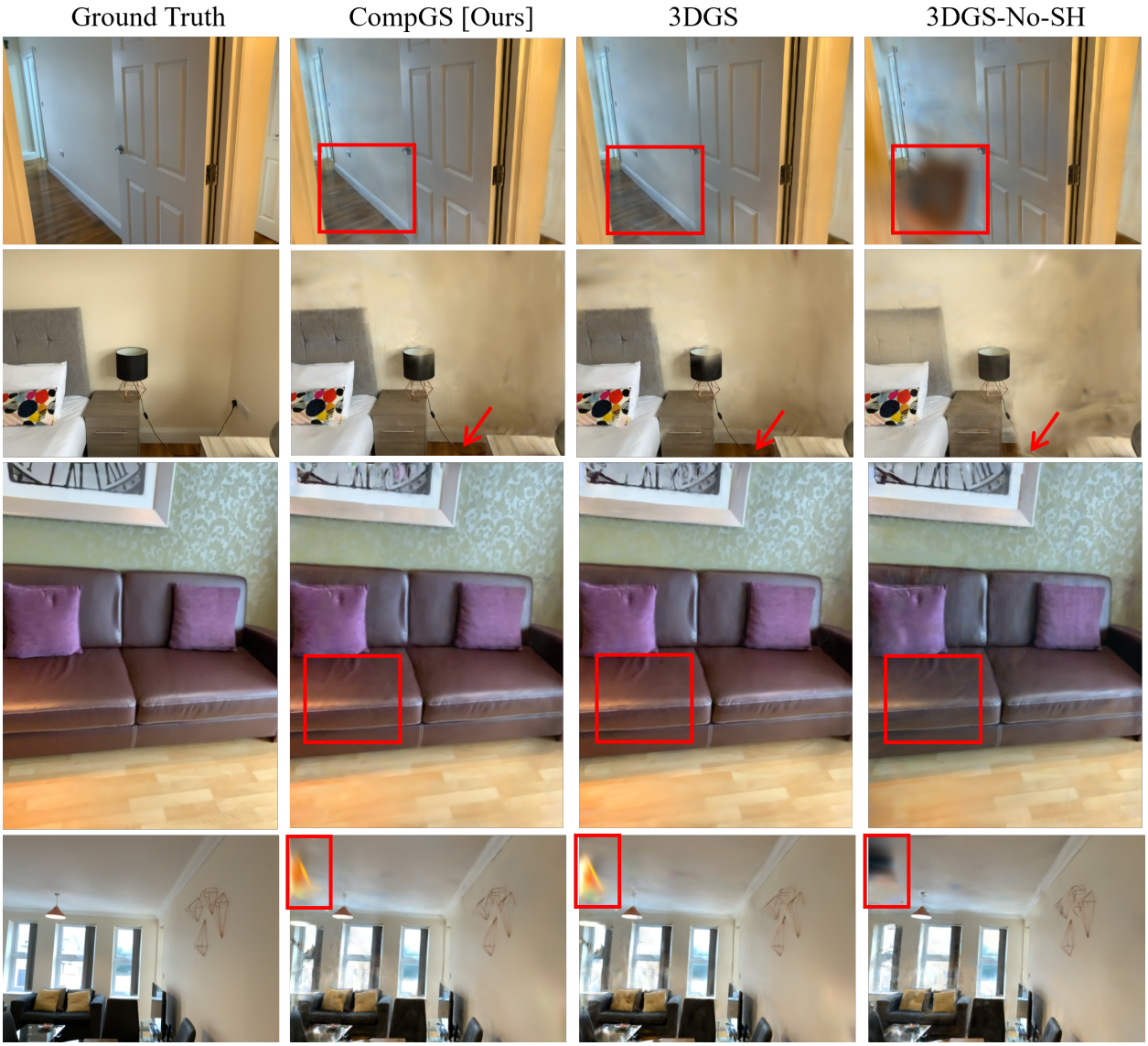


Figure C.5: **Qualitative analysis on ARKit-200 dataset.** We visualize the results of CompGS along with the uncomressed 3DGS and its variant 3DGS-No-SH . Presence of large noisy blobs is a common error mode for 3DGS-No-SH on this dataset. It also fails to faithfully reproduce the colors and lighting in several scenes. The visual quality of the synthesized images for all methods is lower on this dataset compared to the scenes present in standard benchmarks like Mip-NeRF360 , indicating its utility as a novel benchmark. Further comparison with various NeRF based approaches and more analysis can help improve the results on this dataset.

Table C.8: **Ablation study on densification end iteration**. Early or late stopping of densification process impacts the number of Gaussians and the model performance of 3DGS. We report the results with the value set to 3000 in our main submission (table 3).

	Mip-NeRF360				Tanks&Temples				Deep Blending			
	SSIM	PSNR	LPIPS	#Gauss	SSIM	PSNR	LPIPS	#Gauss	SSIM	PSNR	LPIPS	#Gauss
5000	0.797	27.199	0.241	1.92M	0.838	23.599	0.188	1.12M	0.897	29.486	0.256	1.34M
3000	0.780	27.02	0.267	1.12M	0.835	23.55	0.194	810K	0.896	29.42	0.264	795K

Table C.9: **Ablation study on gradient threshold**. We modify the gradient threshold, used as a criterion in the densification process of 3DGS. A higher threshold results in a smaller 3DGS model. Among the baselines considered for reducing Gaussian count, modify gradient threshold provides the best trade-off between model size and performance. Table 3 in main paper shows the results of this experiment for gradient threshold of 0.00045.

	Mip-NeRF360				Tanks&Temples				Deep Blending			
	SSIM	PSNR	LPIPS	#Gauss	SSIM	PSNR	LPIPS	#Gauss	SSIM	PSNR	LPIPS	#Gauss
0.00035	0.786	26.934	0.265	1.27M	0.833	23.579	0.203	833K	0.901	29.574	0.254	1.41M
0.00045	0.769	26.57	0.292	809K	0.825	23.31	0.217	578K	0.900	29.49	0.260	1.01M
0.00055	0.754	26.290	0.312	552K	0.818	23.041	0.228	433K	0.899	29.544	0.265	764K

Bibliography

[1] Official code repository of 3d gaussian splatting for real-time radiance field rendering. https://github.com/graphdeco-inria/gaussian-splatting. 47

[2] Torchvision models. https://pytorch.org/docs/stable/torchvision/models.html. 13, 77

[3] S. Abbasi Koohpayegani, A. Tejankar, and H. Pirsiavash. Compress: Self-supervised learning by compressing representations. *Advances in Neural Information Processing Systems*, 33, 2020. 12, 13, 18, 43

[4] M. Assran, N. Ballas, L. Castrejon, and M. Rabbat. Supervision accelerates pre-training in contrastive semi-supervised learning of visual representations. *arXiv preprint arXiv:2006.10803*, 2020. 8

[5] M. Assran, M. Caron, I. Misra, P. Bojanowski, A. Joulin, N. Ballas, and M. Rabbat. Semi-supervised learning of visual features by non-parametrically predicting view assignments with support samples. *ICCV*, 2021. 8, 18, 19

[6] M. Azabou, M. G. Azar, R. Liu, C.-H. Lin, E. C. Johnson, K. Bhaskaran-Nair, M. Dabagia, B. Avila-Pires, L. Kitchell, K. B. Hengen, et al. Mine your own view: Self-supervised learning through across-sample prediction. *arXiv preprint arXiv:2102.10106*, 2021. 4, 7

[7] J. L. Ba, J. R. Kiros, and G. E. Hinton. Layer normalization. *arXiv preprint arXiv:1607.06450*, 2016. 34

[8] L. J. Ba and R. Caruana. Do deep nets really need to be deep? *arXiv preprint arXiv:1312.6184*, 2013. 43

[9] H. Bagherinezhad, M. Horton, M. Rastegari, and A. Farhadi. Label refinery: Improving imagenet classification through label progression. *arXiv preprint arXiv:1805.02641*, 2018. 7

[10] H. Bagherinezhad, M. Rastegari, and A. Farhadi. Lcnn: Lookup-based convolutional neural network. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 7120–7129, 2017. 25

[11] J. Ballé, D. Minnen, S. Singh, S. J. Hwang, and N. Johnston. Variational image compression with a scale hyperprior. In *International Conference on Learning Representations*, 2018. 26

[12] J. T. Barron, B. Mildenhall, D. Verbin, P. P. Srinivasan, and P. Hedman. Mip-nerf 360: Unbounded anti-aliased neural radiance fields. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 5470–5479, 2022. 42, 45, 47, 48, 97

[13] G. Baruch, Z. Chen, A. Dehghan, T. Dimry, Y. Feigin, P. Fu, T. Gebauer, B. Joffe, D. Kurz, A. Schwartz, and E. Shulman. Arkitscenes - a diverse real-world dataset for 3d indoor scene understanding using mobile rgb-d data. In *NeurIPS*, 2021. 52

[14] G. Baruch, Z. Chen, A. Dehghan, Y. Feigin, P. Fu, T. Gebauer, D. Kurz, T. Dimry, B. Joffe, A. Schwartz, and E. Shulman. ARKitscenes: A diverse real-world dataset for 3d indoor scene understanding using mobile RGB-d data. In *Thirty-fifth Conference on Neural Information Processing Systems Datasets and Benchmarks Track (Round 1)*, 2021. 47, 99

[15] E. Baum and F. Wilczek. Supervised learning of probability distributions by neural networks. In D. Anderson, editor, *Neural Information Processing Systems*. American Institute of Physics, 1988. 15

[16] Y. Bengio, N. Léonard, and A. Courville. Estimating or propagating gradients through stochastic neurons for conditional computation. *arXiv preprint arXiv:1308.3432*, 2013. 45

[17] O. Bohdal, Y. Yang, and T. Hospedales. Flexible dataset distillation: Learn labels instead of images. *arXiv preprint arXiv:2006.08572*, 2020. 26, 32

[18] L. Bossard, M. Guillaumin, and L. Van Gool. Food-101 – mining discriminative components with random forests. In *European Conference on Computer Vision*, 2014. 12

[19] S. R. Bowman, G. Angeli, C. Potts, and C. D. Manning. A large annotated corpus for learning natural language inference. *arXiv preprint arXiv:1508.05326*, 2015. 63

[20] J. Bromley, I. Guyon, Y. LeCun, E. Säckinger, and R. Shah. Signature verification using a" siamese" time delay neural network. *Advances in neural information processing systems*, 6:737–744, 1993. 8

[21] M. Caron, P. Bojanowski, A. Joulin, and M. Douze. Deep clustering for unsupervised learning of visual features. In *Proceedings of the European Conference on Computer Vision (ECCV)*, pages 132–149, 2018. 7, 13

[22] M. Caron, I. Misra, J. Mairal, P. Goyal, P. Bojanowski, and A. Joulin. Unsupervised learning of visual features by contrasting cluster assignments. In *Advances in Neural Information Processing Systems*, pages 9912–9924. Curran Associates, Inc., 2020. 7, 8, 13, 14, 18, 19

[23] M. Caron, H. Touvron, I. Misra, H. Jégou, J. Mairal, P. Bojanowski, and A. Joulin. Emerging properties in self-supervised vision transformers. 2021. 7

[24] G. Cazenavette, T. Wang, A. Torralba, A. A. Efros, and J.-Y. Zhu. Dataset distillation by matching training trajectories. *arXiv preprint arXiv:2203.11932*, 2022. 26, 31, 32

[25] A. Chen, Z. Xu, A. Geiger, J. Yu, and H. Su. Tensorf: Tensorial radiance fields. In *European Conference on Computer Vision*, pages 333–350. Springer, 2022. 43

[26] G. Chen, W. Choi, X. Yu, T. Han, and M. Chandraker. Learning efficient object detection models with knowledge distillation. In *Proceedings of the 31st International Conference on Neural Information Processing Systems*, pages 742–751, 2017. 43

[27] 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. 7, 8, 13, 14, 15, 18

[28] T. Chen, S. Kornblith, K. Swersky, M. Norouzi, and G. E. Hinton. Big self-supervised models are strong semi-supervised learners. *Advances in Neural Information Processing Systems*, 33:22243–22255, 2020. 8, 19

[29] W. Chen, J. Wilson, S. Tyree, K. Weinberger, and Y. Chen. Compressing neural networks with the hashing trick. In *International conference on machine learning*, pages 2285–2294. PMLR, 2015. 23, 24, 33

[30] X. Chen, H. Fan, R. Girshick, and K. He. Improved baselines with momentum contrastive learning. *arXiv preprint arXiv:2003.04297*, 2020. 13, 18

[31] X. Chen and K. He. Exploring simple siamese representation learning. *arXiv preprint arXiv:2011.10566*, 2020. 13

[32] X. Chen, J. Zhang, and Z. Wang. Peek-a-boo: What (more) is disguised in a randomly weighted neural network, and how to find it efficiently. In *International Conference on Learning Representations*, 2021. 24

[33] Z. Chen, T. Funkhouser, P. Hedman, and A. Tagliasacchi. Mobilenerf: Exploiting the polygon rasterization pipeline for efficient neural field rendering on mobile architectures. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 16569–16578, 2023. 42

[34] M. Cho, K. A. Vahid, Q. Fu, S. Adya, C. C. Del Mundo, M. Rastegari, D. Naik, and P. Zatloukal. edkm: An efficient and accurate train-time weight clustering for large language models. *arXiv preprint arXiv:2309.00964*, 2023. 43

[35] S. Chopra, R. Hadsell, and Y. LeCun. Learning a similarity metric discriminatively, with application to face verification. In *2005 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'05)*, volume 1, pages 539–546. IEEE, 2005. 8

[36] M. Cimpoi, S. Maji, I. Kokkinos, S. Mohamed, and A. Vedaldi. Describing textures in the wild. In *Computer Vision and Pattern Recognition*, 2014. 12

[37] P. C. Cosman, K. L. Oehler, E. A. Riskin, and R. M. Gray. Using vector quantization for image processing. *Proceedings of the IEEE*, 81(9):1326–1341, 1993. 43

[38] E. D. Cubuk, B. Zoph, J. Shlens, and Q. Le. Randaugment: Practical automated data augmentation with a reduced search space. In H. Larochelle, M. Ranzato, R. Hadsell, M. F. Balcan, and H. Lin, editors, *Advances in Neural Information Processing Systems*, volume 33, pages 18613–18624. Curran Associates, Inc., 2020. 19, 20

[39] C. L. Deng and E. Tartaglione. Compressing explicit voxel grid representations: fast nerfs become also small. In *Proceedings of the IEEE/CVF Winter Conference on Applications of Computer Vision*, pages 1236–1245, 2023. 43

[40] T. Dettmers, M. Lewis, Y. Belkada, and L. Zettlemoyer. Llm. int8 (): 8-bit matrix multiplication for transformers at scale. *arXiv preprint arXiv:2208.07339*, 2022. 42, 48

[41] T. Dettmers, A. Pagnoni, A. Holtzman, and L. Zettlemoyer. Qlora: Efficient finetuning of quantized llms. In *Advances in Neural Information Processing Systems*, 2023. 59

[42] J. Djelouah and C. Schroers. Content adaptive optimization for neural image compression. In *Proceedings of the CVPR*, 2019. 26

[43] A. Dosovitskiy, J. T. Springenberg, M. Riedmiller, and T. Brox. Discriminative unsupervised feature learning with convolutional neural networks. In *Advances in neural information processing systems*, pages 766–774, 2014. 7

[44] E. Dupont, A. Goliński, M. Alizadeh, Y. W. Teh, and A. Doucet. Coin: Compression with implicit neural representations. *arXiv preprint arXiv:2103.03123*, 2021. 26, 35

[45] D. Dwibedi, Y. Aytar, J. Tompson, P. Sermanet, and A. Zisserman. With a little help from my friends: Nearest-neighbor contrastive learning of visual representations, 2021. 4, 7, 13

[46] W. H. Equitz. A new vector quantization clustering algorithm. *IEEE transactions on acoustics, speech, and signal processing*, 37(10):1568–1575, 1989. 43

[47] 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*, pages 2943–2952. PMLR, 2020. 29, 30

[48] Z. Fan, K. Wang, K. Wen, Z. Zhu, D. Xu, and Z. Wang. Lightgaussian: Unbounded 3d gaussian compression with 15x reduction and 200+ fps. *arXiv preprint arXiv:2311.17245*, 2023. 44, 48, 97

[49] L. Fei-Fei, R. Fergus, and P. Perona. Learning generative visual models from few training examples: An incremental bayesian approach tested on 101 object categories. *Computer Vision and Pattern Recognition Workshop*, 2004. 12

[50] E. Fini, S. Lathuiliere, E. Sangineto, M. Nabi, and E. Ricci. Online continual learning under extreme memory constraints. In *European Conference on Computer Vision*, pages 720–735. Springer, 2020. 24

[51] J. Flynn, I. Neulander, J. Philbin, and N. Snavely. Deepstereo: Learning to predict new views from the world's imagery. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 5515–5524, 2016. 42

[52] P. Foret, A. Kleiner, H. Mobahi, and B. Neyshabur. Sharpness-aware minimization for efficiently improving generalization. In *International Conference on Learning Representations*, 2021. 59, 62

[53] S. Fridovich-Keil, A. Yu, M. Tancik, Q. Chen, B. Recht, and A. Kanazawa. Plenoxels: Radiance fields without neural networks. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 5501–5510, 2022. 42, 43, 48, 97

[54] T. Furlanello, Z. C. Lipton, M. Tschannen, L. Itti, and A. Anandkumar. Born again neural networks, 2018. 7

[55] C. Gallicchio and S. Scardapane. Deep randomized neural networks. *Recent Trends in Learning From Data*, pages 43–68, 2020. 24

[56] S. J. Garbin, M. Kowalski, M. Johnson, J. Shotton, and J. Valentin. Fastnerf: High-fidelity neural rendering at 200fps. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 14346–14355, 2021. 42

[57] A. A. Gargiulo, D. Crisostomi, M. S. Bucarelli, S. Scardapane, F. Silvestri, and E. Rodola. Task singular vectors: Reducing task interference in model merging. In *Proceedings of the Computer Vision and Pattern Recognition Conference*, pages 18695–18705, 2025. 57, 60

[58] A. Gersho and R. M. Gray. *Vector quantization and signal compression*, volume 159. Springer Science & Business Media, 2012. 43

[59] S. Gidaris, A. Bursuc, G. Puy, N. Komodakis, M. Cord, and P. Perez. Obow: Online bag-of-visual-words generation for self-supervised learning. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 6830–6840, June 2021. 13

[60] S. Gidaris, P. Singh, and N. Komodakis. Unsupervised representation learning by predicting image rotations. In *International Conference on Learning Representations*, 2018. 7

[61] S. Girish, K. Gupta, and A. Shrivastava. Eagles: Efficient accelerated 3d gaussians with lightweight encodings. *arXiv preprint arXiv:2312.04564*, 2023. 44

[62] J. Goldberger, G. E. Hinton, S. Roweis, and R. R. Salakhutdinov. Neighbourhood components analysis. *Advances in neural information processing systems*, 17:513–520, 2004. 7

[63] Y. Gong, L. Liu, M. Yang, and L. Bourdev. Compressing deep convolutional networks using vector quantization. *arXiv preprint arXiv:1412.6115*, 2014. 43

[64] R. Gray. Vector quantization. *IEEE Assp Magazine*, 1(2):4–29, 1984. 43

[65] J.-B. Grill, F. Strub, F. Altché, C. Tallec, P. H. Richemond, E. Buchatskaya, C. Doersch, B. A. Pires, Z. D. Guo, M. G. Azar, et al. Bootstrap your own latent: A new approach to self-supervised learning. *arXiv preprint arXiv:2006.07733*, 2020. 7, 9, 13, 14, 15, 18, 19

[66] S. Gu, D. Chen, J. Bao, F. Wen, B. Zhang, D. Chen, L. Yuan, and B. Guo. Vector quantized diffusion model for text-to-image synthesis. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 10696–10706, 2022. 43

[67] T. Guo, J. Wang, Z. Cui, Y. Feng, Y. Ge, and B. Bai. Variable rate image compression with content adaptive optimization. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops*, pages 122–123, 2020. 26

[68] S. Gupta, A. Agrawal, K. Gopalakrishnan, and P. Narayanan. Deep learning with limited numerical precision. In *International conference on machine learning*, pages 1737–1746. PMLR, 2015. 43

[69] J. Hamer, M. Mohri, and A. T. Suresh. Fedboost: A communication-efficient algorithm for federated learning. In *International Conference on Machine Learning*, pages 3973–3983. PMLR, 2020. 24

[70] K. Han, Y. Wang, Q. Tian, J. Guo, C. Xu, and C. Xu. Ghostnet: More features from cheap operations. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 1580–1589, 2020. 25

[71] S. Han, H. Mao, and W. J. Dally. Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding. *arXiv preprint arXiv:1510.00149*, 2015. 25, 42, 43

[72] T. Han, W. Xie, and A. Zisserman. Self-supervised co-training for video representation learning, 2021. 7

[73] M. Havasi, R. Peharz, and J. M. Hernández-Lobato. Minimal random code learning: Getting bits back from compressed model parameters. *arXiv preprint arXiv:1810.00440*, 2018. 25

[74] S. Hayou, J.-F. Ton, A. Doucet, and Y. W. Teh. Robust pruning at initialization. *arXiv preprint arXiv:2002.08797*, 2020. 25

[75] 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. 7, 13

[76] 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. 12

[77] P. Hedman, J. Philip, T. Price, J.-M. Frahm, G. Drettakis, and G. Brostow. Deep blending for free-viewpoint image-based rendering. *ACM Transactions on Graphics (ToG)*, 37(6):1–15, 2018. 42, 47

[78] P. Hedman, P. P. Srinivasan, B. Mildenhall, J. T. Barron, and P. Debevec. Baking neural radiance fields for real-time view synthesis. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 5875–5884, 2021. 42

[79] O. J. Hénaff, A. Srinivas, J. De Fauw, A. Razavi, C. Doersch, S. Eslami, and A. v. d. Oord. Data-efficient image recognition with contrastive predictive coding. *arXiv preprint arXiv:1905.09272*, 2019. 14

[80] P. Henzler, N. J. Mitra, and T. Ritschel. Escaping plato's cave: 3d shape from adversarial rendering. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 9984–9993, 2019. 42

[81] G. Hinton, O. Vinyals, J. Dean, et al. Distilling the knowledge in a neural network. *arXiv preprint arXiv:1503.02531*, 2(7), 2015. 7, 22, 43

[82] N. Houlsby, A. Giurgiu, S. Jastrzebski, B. Morrone, Q. De Laroussilhe, A. Gesmundo, M. Attariyan, and S. Gelly. Parameter-efficient transfer learning for nlp. In *International conference on machine learning*, pages 2790–2799. PMLR, 2019. 57, 59

[83] E. J. Hu, Y. Shen, P. Wallis, Z. Allen-Zhu, Y. Li, S. Wang, L. Wang, W. Chen, et al. Lora: Low-rank adaptation of large language models. *ICLR*, 1(2):3, 2022. 57, 59

[84] B. Huang, X. Yan, A. Chen, S. Gao, and J. Yu. Pref: Phasorial embedding fields for compact neural representations. *arXiv preprint arXiv:2205.13524*, 2022. 43

[85] J. Huang, Q. Dong, S. Gong, and X. Zhu. Unsupervised deep learning via affinity diffusion. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 34, pages 11029–11036, 2020. 7

[86] T. Huynh, S. Kornblith, M. R. Walter, M. Maire, and M. Khademi. Boosting contrastive self-supervised learning with false negative cancellation. *arXiv preprint arXiv:2011.11765*, 2020. 4

[87] G. Ilharco, M. T. Ribeiro, M. Wortsman, L. Schmidt, H. Hajishirzi, and A. Farhadi. Editing models with task arithmetic. In *The Eleventh International Conference on Learning Representations*, 2023. 57, 60

[88] B. Isik, T. Weissman, and A. No. An information-theoretic justification for model pruning. In *International Conference on Artificial Intelligence and Statistics*, pages 3821–3846. PMLR, 2022. 25, 29, 30

[89] B. Jacob, S. Kligys, B. Chen, M. Zhu, M. Tang, A. Howard, H. Adam, and D. Kalenichenko. Quantization and training of neural networks for efficient integer-arithmetic-only inference. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 2704–2713, 2018. 42, 43

[90] Y. Kalantidis, M. B. Sariyildiz, N. Pion, P. Weinzaepfel, and D. Larlus. Hard negative mixing for contrastive learning. *Advances in Neural Information Processing Systems*, 2020. 4

[91] B. Kerbl, G. Kopanas, T. Leimkühler, and G. Drettakis. 3d gaussian splatting for real-time radiance field rendering. *ACM Transactions on Graphics (ToG)*, 42(4):1–14, 2023. 39, 42, 44, 47, 48, 49, 95, 97, 99, 101

[92] P. Khosla, P. Teterwak, C. Wang, A. Sarna, Y. Tian, P. Isola, A. Maschinot, C. Liu, and D. Krishnan. Supervised contrastive learning. *Advances in Neural Information Processing Systems*, 33, 2020. 7, 15

[93] T. Khot, A. Sabharwal, and P. Clark. Scitail: A textual entailment dataset from science question answering. In *Proceedings of the AAAI conference on artificial intelligence*, volume 32, 2018. 63

[94] W. Kim, S. Kim, M. Park, and G. Jeon. Neuron merging: Compensating for pruned neurons. *Advances in Neural Information Processing Systems*, 33:585–595, 2020. 25

[95] D. P. Kingma and J. Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014. 36

[96] A. Knapitsch, J. Park, Q.-Y. Zhou, and V. Koltun. Tanks and temples: Benchmarking large-scale scene reconstruction. *ACM Transactions on Graphics (ToG)*, 36(4):1–13, 2017. 47

[97] Kodak. Kodak Dataset. http://r0k.us/graphics/kodak/, 1991. 35

[98] S. A. Koohpayegani, A. Tejankar, and H. Pirsiavash. Mean shift for self-supervised learning. In *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)*, pages 10326–10335, October 2021. 4, 6, 7, 8, 9, 12, 13, 15, 18, 19, 69, 70, 71

[99] J. Krause, M. Stark, J. Deng, and L. Fei-Fei. 3D object representations for fine-grained categorization. In *Workshop on 3D Representation and Recognition*, Sydney, Australia, 2013. 12

[100] R. Krishnamoorthi. Quantizing deep convolutional networks for efficient inference: A whitepaper. *arXiv preprint arXiv:1806.08342*, 2018. 42, 43

[101] A. Krizhevsky. Learning multiple layers of features from tiny images. Technical report, University of Toronto, 2009. 12

[102] A. Kusupati, V. Ramanujan, R. Somani, M. Wortsman, P. Jain, S. Kakade, and A. Farhadi. Soft threshold weight reparameterization for learnable sparsity. In *Proceedings of the International Conference on Machine Learning*, July 2020. 25, 29, 30

[103] Y. LeCun et al. Lenet-5, convolutional neural networks. *URL: http://yann. lecun. com/exdb/lenet*, 20(5):14, 2015. 30, 33

[104] D.-H. Lee et al. Pseudo-label: The simple and efficient semi-supervised learning method for deep neural networks. In *Workshop on challenges in representation learning, ICML*, volume 3, page 896, 2013. 8

[105] J. Lee, S. Cho, and S.-K. Beack. Context-adaptive entropy model for end-to-end optimized image compression. *arXiv preprint arXiv:1809.10452*, 2018. 26

[106] J. Lee, D. Kim, and B. Ham. Network quantization with element-wise gradient scaling. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 6448–6457, 2021. 22, 25

[107] J. Lee, S. Park, S. Mo, S. Ahn, and J. Shin. Layer-adaptive sparsity for the magnitude-based pruning. *arXiv preprint arXiv:2010.07611*, 2020. 29, 30

[108] J. C. Lee, D. Rho, X. Sun, J. H. Ko, and E. Park. Compact 3d gaussian representation for radiance field. *arXiv preprint arXiv:2311.13681*, 2023. 44

[109] J. C. Lee, D. Rho, X. Sun, J. H. Ko, and E. Park. Compact 3d gaussian representation for radiance field. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 21719–21728, 2024. 48, 97

[110] K. Lee, Y. Zhu, K. Sohn, C.-L. Li, J. Shin, and H. Lee. *i*-mix: A domain-agnostic strategy for contrastive representation learning. In *International Conference on Learning Representations*, 2020. 4

[111] Y. Y. Lee and J. W. Woods. Motion vector quantization for video coding. *IEEE Transactions on Image Processing*, 4(3):378–382, 1995. 43

[112] B. Lester, R. Al-Rfou, and N. Constant. The power of scale for parameter-efficient prompt tuning. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing*, 2021. 59

[113] E. Levin and M. Fleisher. Accelerated learning in layered neural networks. *Complex systems*, 2(625-640):3, 1988. 15

[114] L. Li, L. Qi, Q. Tian, and X. Geng. Model merging in the essential subspace. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 31145–31154, 2026. 57, 60

[115] L. Li, Z. Shen, Z. Wang, L. Shen, and L. Bo. Compressing volumetric radiance fields to 1 mb. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 4222–4231, 2023. 43

[116] L. Li, Z. Shen, Z. Wang, L. Shen, and P. Tan. Streaming radiance fields for 3d video synthesis. *Advances in Neural Information Processing Systems*, 35:13485–13498, 2022. 43

[117] X. L. Li and P. Liang. Prefix-tuning: Optimizing continuous prompts for generation. In *Proceedings of the Annual Meeting of the Association for Computational Linguistics*, 2021. 59

[118] Y. Li, S. Lin, J. Liu, Q. Ye, M. Wang, F. Chao, F. Yang, J. Ma, Q. Tian, and R. Ji. Towards compact cnns via collaborative compression. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 6438–6447, 2021. 25

[119] T. Lin, S. U. Stich, L. Barba, D. Dmitriev, and M. Jaggi. Dynamic model pruning with feedback. *arXiv preprint arXiv:2006.07253*, 2020. 22, 25, 29, 30

[120] L. Ling, Y. Sheng, Z. Tu, W. Zhao, C. Xin, K. Wan, L. Yu, Q. Guo, Z. Yu, Y. Lu, et al. Dl3dv-10k: A large-scale scene dataset for deep learning-based 3d vision. *arXiv preprint arXiv:2312.16256*, 2023. 47, 52

[121] S.-Y. Liu, C.-Y. Wang, H. Yin, P. Molchanov, Y.-C. F. Wang, K.-T. Cheng, and M.-H. Chen. Dora: Weight-decomposed low-rank adaptation. In *Forty-first International Conference on Machine Learning*, 2024. 57, 59

[122] S. Maji, E. Rahtu, J. Kannala, M. B. Blaschko, and A. Vedaldi. Fine-grained visual classification of aircraft. *arXiv preprint arXiv:1306.5151*, 2013. 12

[123] J. Makhoul, S. Roucos, and H. Gish. Vector quantization in speech coding. *Proceedings of the IEEE*, 73(11):1551–1588, 1985. 43

[124] E. Malach, G. Yehudai, S. Shalev-Schwartz, and O. Shamir. Proving the lottery ticket hypothesis: Pruning is all you need. In *International Conference on Machine Learning*, pages 6682–6691. PMLR, 2020. 24

[125] D. Marczak, S. Magistri, S. Cygert, B. Twardowski, A. D. Bagdanov, and J. Van De Weijer. No task left behind: Isotropic model merging with common and task-specific subspaces. In A. Singh, M. Fazel, D. Hsu, S. Lacoste-Julien, F. Berkenkamp, T. Maharaj, K. Wagstaff, and J. Zhu, editors, *Proceedings of the 42nd International Conference on Machine Learning*, volume 267 of *Proceedings of Machine Learning Research*, pages 43177–43199. PMLR, 13–19 Jul 2025. 60

[126] M. Marelli, L. Bentivogli, M. Baroni, R. Bernardi, S. Menini, and R. Zamparelli. Semeval-2014 task 1: Evaluation of compositional distributional semantic models on full sentences through semantic relatedness and textual entailment. In *Proceedings of the 8th international workshop on semantic evaluation (SemEval 2014)*, pages 1–8, 2014. 63

[127] M. S. Matena and C. A. Raffel. Merging models with fisher-weighted averaging. In *Advances in Neural Information Processing Systems*, 2022. 60

[128] F. Meng, Z. Wang, and M. Zhang. Pissa: Principal singular values and singular vectors adaptation of large language models. *arXiv preprint arXiv:2404.02948*, 2024. 59

[129] B. Mildenhall, P. P. Srinivasan, M. Tancik, J. T. Barron, R. Ramamoorthi, and R. Ng. Nerf: Representing scenes as neural radiance fields for view synthesis. In *Proceedings of the European Conference on Computer Vision (ECCV)*, 2020. 39, 42, 95, 96

[130] D. Minnen, J. Ballé, and G. D. Toderici. Joint autoregressive and hierarchical priors for learned image compression. *Advances in neural information processing systems*, 31, 2018. 26

[131] I. Misra and L. van der Maaten. Self-supervised learning of pretext-invariant representations. *arXiv preprint arXiv:1912.01991*, 2019. 18

[132] W. Morgenstern, F. Barthel, A. Hilsmann, and P. Eisert. Compact 3d scene representation via self-organizing gaussian grids. *arXiv preprint arXiv:2312.13299*, 2023. 44

[133] T. Müller, A. Evans, C. Schied, and A. Keller. Instant neural graphics primitives with a multiresolution hash encoding. *ACM Transactions on Graphics (ToG)*, 41(4):1–15, 2022. 42, 43, 48, 97

[134] R. Müller, S. Kornblith, and G. Hinton. When does label smoothing help?, 2020. 7

[135] S. Niedermayr, J. Stumpfegger, and R. Westermann. Compressed 3d gaussian splatting for accelerated novel view synthesis. *arXiv preprint arXiv:2401.02436*, 2023. 44

[136] S. Niedermayr, J. Stumpfegger, and R. Westermann. Compressed 3d gaussian splatting for accelerated novel view synthesis. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 10349–10358, 2024. 48, 97

[137] M.-E. Nilsback and A. Zisserman. Automated flower classification over a large number of classes. In *Indian Conference on Computer Vision, Graphics and Image Processing*, 2008. 12

[138] P. Nooralinejad, A. Abbasi, S. A. Koohpayegani, K. P. Meibodi, R. M. S. Khan, S. Kolouri, and H. Pirsiavash. Pranc: Pseudo random networks for compacting deep models. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 17021–17031, 2023. 43

[139] M. Noroozi and P. Favaro. Unsupervised learning of visual representations by solving jigsaw puzzles. In *European Conference on Computer Vision*, pages 69–84. Springer, 2016. 7

[140] M. Noroozi, H. Pirsiavash, and P. Favaro. Representation learning by learning to count. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 5898–5906, 2017. 7

[141] A. Panariello, D. Marczak, S. Magistri, A. Porrello, B. Twardowski, A. Bagdanov, S. Calderara, and J. van de Weijer. Accurate and efficient low-rank model merging in core space. *Advances in Neural Information Processing Systems*, 38:61793–61825, 2026. 57, 60

[142] O. M. Parkhi, A. Vedaldi, A. Zisserman, and C. V. Jawahar. Cats and dogs. In *Computer Vision and Pattern Recognition*, 2012. 12

[143] S. Peng, C. Jiang, Y. Liao, M. Niemeyer, M. Pollefeys, and A. Geiger. Shape as points: A differentiable poisson solver. *Advances in Neural Information Processing Systems*, 34:13032–13044, 2021. 43

[144] E. Penner and L. Zhang. Soft 3d reconstruction for view synthesis. *ACM Transactions on Graphics (TOG)*, 36(6):1–11, 2017. 42

[145] H. Pham, Z. Dai, Q. Xie, and Q. V. Le. Meta pseudo labels. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 11557–11568, 2021. 8, 19

[146] A. Polino, R. Pascanu, and D. Alistarh. Model compression via distillation and quantization. *arXiv preprint arXiv:1802.05668*, 2018. 43

[147] V. Ramanujan, M. Wortsman, A. Kembhavi, A. Farhadi, and M. Rastegari. What's hidden in a randomly weighted neural network? In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 11893–11902, 2020. 24

[148] 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. 25, 43

[149] M. Rastegari, V. Ordonez, J. Redmon, and A. Farhadi. Xnor-net: Imagenet classification using binary convolutional neural networks, 2016. 41

[150] A. Razavi, A. Van den Oord, and O. Vinyals. Generating diverse high-fidelity images with vq-vae-2. *Advances in neural information processing systems*, 32, 2019. 43

[151] B. Reagan, U. Gupta, B. Adolf, M. Mitzenmacher, A. Rush, G.-Y. Wei, and D. Brooks. Weightless: Lossy weight encoding for deep neural network compression. In *International Conference on Machine Learning*, pages 4324–4333. PMLR, 2018. 25

[152] C. J. Reed, S. Metzger, A. Srinivas, T. Darrell, and K. Keutzer. Selfaugment: Automatic augmentation policies for self-supervised learning. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 2674–2683, 2021. 4

[153] C. Reiser, S. Peng, Y. Liao, and A. Geiger. Kilonerf: Speeding up neural radiance fields with thousands of tiny mlps. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 14335–14345, 2021. 42

[154] G. Riegler and V. Koltun. Free view synthesis. In *Computer Vision–ECCV 2020: 16th European Conference, Glasgow, UK, August 23–28, 2020, Proceedings, Part XIX 16*, pages 623–640. Springer, 2020. 42

[155] D. E. Rumelhart, G. E. Hinton, and R. J. Williams. Learning representations by back-propagating errors. *nature*, 323(6088):533–536, 1986. 15

[156] R. Salakhutdinov and G. Hinton. Learning a nonlinear embedding by preserving class neighbourhood structure. In *Artificial Intelligence and Statistics*, pages 412–419. PMLR, 2007. 7

[157] F. Schroff, D. Kalenichenko, and J. Philbin. Facenet: A unified embedding for face recognition and clustering. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 815–823, 2015. 8

[158] K. Schwarz, A. Sauer, M. Niemeyer, Y. Liao, and A. Geiger. Voxgraf: Fast 3d-aware image synthesis with sparse voxel grids. *Advances in Neural Information Processing Systems*, 35:33999–34011, 2022. 43

[159] J. N. Siems, A. Klein, C. Archambeau, and M. Mahsereci. Dynamic pruning of a neural network via gradient signal-to-noise ratio. In *8th ICML Workshop on Automated Machine Learning (AutoML)*, 2021. 25

[160] V. Sitzmann, J. Martel, A. Bergman, D. Lindell, and G. Wetzstein. Implicit neural representations with periodic activation functions. *Advances in Neural Information Processing Systems*, 33:7462–7473, 2020. 35

[161] V. Sitzmann, J. Thies, F. Heide, M. Nießner, G. Wetzstein, and M. Zollhofer. Deepvoxels: Learning persistent 3d feature embeddings. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 2437–2446, 2019. 42

[162] J. Snell, K. Swersky, and R. S. Zemel. Prototypical networks for few-shot learning. *arXiv preprint arXiv:1703.05175*, 2017. 8, 15

[163] K. Sohn. Improved deep metric learning with multi-class n-pair loss objective. In *Proceedings of the 30th International Conference on Neural Information Processing Systems*, pages 1857–1865, 2016. 8

[164] K. Sohn, D. Berthelot, N. Carlini, Z. Zhang, H. Zhang, C. A. Raffel, E. D. Cubuk, A. Kurakin, and C.-L. Li. Fixmatch: Simplifying semi-supervised learning with consistency and confidence. *Advances in Neural Information Processing Systems*, 33, 2020. 8, 19

[165] G. Stoica, P. Ramesh, B. Ecsedi, L. Choshen, and J. Hoffman. Model merging with SVD to tie the knots. In *The Thirteenth International Conference on Learning Representations*, 2025. 57, 59, 60, 63, 64

[166] C. Subia-Waud and S. Dasmahapatra. Weight fixing networks. In *European Conference on Computer Vision*, pages 415–431. Springer, 2022. 23, 24

[167] I. Sucholutsky and M. Schonlau. Soft-label dataset distillation and text dataset distillation. In *2021 International Joint Conference on Neural Networks (IJCNN)*, pages 1–8. IEEE, 2021. 26, 32

[168] S. Sukhbaatar, J. Bruna, M. Paluri, L. Bourdev, and R. Fergus. Training convolutional networks with noisy labels, 2015. 7

[169] C. Sun, M. Sun, and H.-T. Chen. Direct voxel grid optimization: Super-fast convergence for radiance fields reconstruction. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 5459–5469, 2022. 43

[170] C. Szegedy, V. Vanhoucke, S. Ioffe, J. Shlens, and Z. Wojna. Rethinking the inception architecture for computer vision, 2015. 7

[171] T. Takikawa, A. Evans, J. Tremblay, T. Müller, M. McGuire, A. Jacobson, and S. Fidler. Variable bitrate neural fields. In *ACM SIGGRAPH 2022 Conference Proceedings*, pages 1–9, 2022. 42, 43

[172] M. Tancik, P. Srinivasan, B. Mildenhall, S. Fridovich-Keil, N. Raghavan, U. Singhal, R. Ramamoorthi, J. Barron, and R. Ng. Fourier features let networks learn high frequency functions in low dimensional domains. *Advances in Neural Information Processing Systems*, 33:7537–7547, 2020. 29, 34

[173] J. Tang, X. Chen, J. Wang, and G. Zeng. Compressible-composable nerf via rank-residual decomposition. In *Advances in Neural Information Processing Systems*, 2022. 43

[174] A. Tejankar, S. A. Koohpayegani, V. Pillai, P. Favaro, and H. Pirsiavash. Isd: Self-supervised learning by iterative similarity distillation. In *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)*, pages 9609–9618, October 2021. 13, 18

[175] J. Thies, M. Zollhöfer, and M. Nießner. Deferred neural rendering: Image synthesis using neural textures. *Acm Transactions on Graphics (TOG)*, 38(4):1–12, 2019. 42

[176] Y. Tian, D. Krishnan, and P. Isola. Contrastive multiview coding. *arXiv preprint arXiv:1906.05849*, 2019. 16

[177] Y. Tian, C. Sun, B. Poole, D. Krishnan, C. Schmid, and P. Isola. What makes for good views for contrastive learning? In *Advances in Neural Information Processing Systems*, volume 33, pages 6827–6839. Curran Associates, Inc., 2020. 4

[178] R. Tiwari, U. Bamba, A. Chavan, and D. K. Gupta. Chipnet: Budget-aware pruning with heaviside continuous approximations. *arXiv preprint arXiv:2102.07156*, 2021. 25

[179] G. Toderici, W. Shi, R. Timofte, L. Theis, J. Balle, E. Agustsson, N. Johnston, and F. Mentzer. Workshop and challenge on learned image compression (clic2020), 2020. 35

[180] H. Touvron, A. Sablayrolles, M. Douze, M. Cord, and H. Jégou. Grafit: Learning fine-grained image representations with coarse labels, 2020. 7

[181] Y.-H. H. Tsai, T. Li, W. Liu, P. Liao, R. Salakhutdinov, and L.-P. Morency. Integrating auxiliary information in self-supervised learning, 2021. 7

[182] K. Ullrich, E. Meeds, and M. Welling. Soft weight-sharing for neural network compression. *arXiv preprint arXiv:1702.04008*, 2017. 25

[183] A. Van Den Oord, O. Vinyals, et al. Neural discrete representation learning. *Advances in neural information processing systems*, 30, 2017. 43

[184] V. Vanhoucke, A. Senior, and M. Z. Mao. Improving the speed of neural networks on cpus. 2011. 43

[185] O. Vinyals, C. Blundell, T. Lillicrap, K. Kavukcuoglu, and D. Wierstra. Matching networks for one shot learning, 2017. 8

[186] A. Wang, A. Singh, J. Michael, F. Hill, O. Levy, and S. R. Bowman. Glue: A multi-task benchmark and analysis platform for natural language understanding. *arXiv preprint arXiv:1804.07461*, 2018. 63

[187] F. Wang, H. Liu, D. Guo, and S. Fuchun. Unsupervised representation learning by invariance propagation. In *Advances in Neural Information Processing Systems*, volume 33, pages 3510–3520. Curran Associates, Inc., 2020. 13, 18

[188] G. Wang, K. Wang, G. Wang, P. H. S. Torr, and L. Lin. Solving inefficiency of self-supervised representation learning, 2021. 4, 13

[189] H. Wang, C. Qin, Y. Zhang, and Y. Fu. Neural pruning via growing regularization. *arXiv preprint arXiv:2012.09243*, 2020. 25

[190] K. Wang, N. Dimitriadis, A. Favero, G. Ortiz-Jimenez, F. Fleuret, and P. Frossard. Lines: Post-training layer scaling prevents forgetting and enhances model merging. In *International Conference on Learning Representations*, 2025. 60

[191] K. Wang, B. Zhao, X. Peng, Z. Zhu, S. Yang, S. Wang, G. Huang, H. Bilen, X. Wang, and Y. You. Cafe: Learning to condense dataset by aligning features. *arXiv preprint arXiv:2203.01531*, 2022. 32

[192] L. Wang, J. Zhang, X. Liu, F. Zhao, Y. Zhang, Y. Zhang, M. Wu, J. Yu, and L. Xu. Fourier plenoctrees for dynamic radiance field rendering in real-time. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 13524–13534, 2022. 43

[193] S. Wang, L. Yu, and J. Li. Lora-ga: Low-rank adaptation with gradient approximation. *arXiv preprint arXiv:2407.05000*, 2024. 59

[194] T. Wang, J.-Y. Zhu, A. Torralba, and A. A. Efros. Dataset distillation. *arXiv preprint arXiv:1811.10959*, 2018. 23, 25, 31, 32

[195] X. Wang, Z. Liu, and S. X. Yu. Unsupervised feature learning by cross-level instance-group discrimination. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 12586–12595, June 2021. 7

[196] X. Wang, Y. Peng, L. Lu, Z. Lu, M. Bagheri, and R. M. Summers. Chestx-ray8: Hospital-scale chest x-ray database and benchmarks on weakly-supervised classification and localization of common thorax diseases. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 2097–2106, 2017. 35

[197] Z. Wang, J. Liang, R. He, Z. Wang, and T. Tan. Lora-pro: Are low-rank adapters properly optimized? *arXiv preprint arXiv:2407.18242*, 2024. 59

[198] C. Wei, H. Wang, W. Shen, and A. Yuille. Co2: Consistent contrast for unsupervised visual representation learning. *arXiv preprint arXiv:2010.02217*, 2020. 13, 18

[199] K. Q. Weinberger, J. Blitzer, and L. K. Saul. Distance metric learning for large margin nearest neighbor classification. In *Advances in neural information processing systems*, pages 1473–1480, 2006. 8

[200] W. Wen, C. Wu, Y. Wang, Y. Chen, and H. Li. Learning structured sparsity in deep neural networks. *arXiv preprint arXiv:1608.03665*, 2016. 43

[201] A. Williams, N. Nangia, and S. R. Bowman. A broad-coverage challenge corpus for sentence understanding through inference. *arXiv preprint arXiv:1704.05426*, 2017. 63

[202] M. Wortsman, G. Ilharco, S. Y. Gadre, R. Roelofs, R. Gontijo-Lopes, A. S. Morcos, H. Namkoong, A. Farhadi, Y. Carmon, S. Kornblith, and L. Schmidt. Model soups: Averaging weights of multiple fine-tuned models improves accuracy without increasing inference time. In *International Conference on Machine Learning*, 2022. 60

[203] M. Wortsman, V. Ramanujan, R. Liu, A. Kembhavi, M. Rastegari, J. Yosinski, and A. Farhadi. Supermasks in superposition. *Advances in Neural Information Processing Systems*, 33:15173–15184, 2020. 24

[204] C.-Y. Wu, R. Manmatha, A. J. Smola, and P. Krahenbuhl. Sampling matters in deep embedding learning. In *Proceedings of the IEEE International Conference on Computer Vision (ICCV)*, Oct 2017. 8

[205] X. Wu, J. Xu, Z. Zhu, H. Bao, Q. Huang, J. Tompkin, and W. Xu. Scalable neural indoor scene rendering. *ACM Transactions on Graphics (TOG)*, 2022. 42

[206] Z. Wu, A. A. Efros, and S. X. Yu. Improving generalization via scalable neighborhood component analysis, 2018. 7

[207] J. Xiao, J. Hays, K. A. Ehinger, A. Oliva, and A. Torralba. Sun database: Large-scale scene recognition from abbey to zoo. In *Computer Vision and Pattern Recognition*, 2010. 12

[208] Q. Xie, Z. Dai, E. Hovy, M.-T. Luong, and Q. V. Le. Unsupervised data augmentation for consistency training. *NeurIPS*, 2020. 8, 19

[209] Q. Xu, Z. Xu, J. Philip, S. Bi, Z. Shu, K. Sunkavalli, and U. Neumann. Point-nerf: Point-based neural radiance fields. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 5438–5448, 2022. 43

[210] Y. Xu, Q. Qian, H. Li, R. Jin, and J. Hu. Weakly supervised representation learning with coarse labels, 2021. 7

[211] P. Yadav, D. Tam, L. Choshen, C. A. Raffel, and M. Bansal. Ties-merging: Resolving interference when merging models. *Advances in Neural Information Processing Systems*, 36:7093–7115, 2023. 57, 60, 63

[212] E. Yang, Z. Wang, L. Shen, S. Liu, G. Guo, X. Wang, and D. Tao. Adamerging: Adaptive model merging for multi-task learning. In *The Twelfth International Conference on Learning Representations*, 2024. 60

[213] A. YM., R. C., and V. A. Self-labelling via simultaneous clustering and representation learning. In *International Conference on Learning Representations*, 2020. 13

[214] A. Yu, R. Li, M. Tancik, H. Li, R. Ng, and A. Kanazawa. Plenoctrees for real-time rendering of neural radiance fields. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 5752–5761, 2021. 42, 43

[215] L. Yu, B. Yu, H. Yu, F. Huang, and Y. Li. Language models are super mario: Absorbing abilities from homologous models as a free lunch. In *Forty-first International Conference on Machine Learning*, 2024. 57, 60

[216] S. Yun, D. Han, S. J. Oh, S. Chun, J. Choe, and Y. Yoo. Cutmix: Regularization strategy to train strong classifiers with localizable features, 2019. 7

[217] X. Zhai, A. Oliver, A. Kolesnikov, and L. Beyer. S4l: Self-supervised semi-supervised learning. In *The IEEE International Conference on Computer Vision (ICCV)*, October 2019. 8, 18

[218] H. Zhang, M. Cisse, Y. N. Dauphin, and D. Lopez-Paz. mixup: Beyond empirical risk minimization, 2018. 7

[219] L. Zhang, L. Zhang, S. Shi, X. Chu, and B. Li. Lora-fa: Memory-efficient low-rank adaptation for large language models fine-tuning. *arXiv preprint arXiv:2308.03303*, 2023. 59

[220] Q. Zhang, M. Chen, A. Bukharin, N. Karampatziakis, P. He, Y. Cheng, W. Chen, and T. Zhao. Adalora: Adaptive budget allocation for parameter-efficient fine-tuning. *arXiv preprint arXiv:2303.10512*, 2023. 57, 59

[221] R. Zhang, P. Isola, and A. A. Efros. Colorful image colorization. In *European conference on computer vision*, pages 649–666. Springer, 2016. 7

[222] Z. Zhang and M. R. Sabuncu. Generalized cross entropy loss for training deep neural networks with noisy labels. *arXiv preprint arXiv:1805.07836*, 2018. 7

[223] B. Zhao and H. Bilen. Dataset condensation with differentiable siamese augmentation. In *International Conference on Machine Learning*, pages 12674–12685. PMLR, 2021. 26, 32

[224] B. Zhao and H. Bilen. Dataset condensation with distribution matching. *arXiv preprint arXiv:2110.04181*, 2021. 26, 32

[225] B. Zhao, K. R. Mopuri, and H. Bilen. Dataset condensation with gradient matching. *arXiv preprint arXiv:2006.05929*, 2020. 32

[226] T. Zhou, S. Tulsiani, W. Sun, J. Malik, and A. A. Efros. View synthesis by appearance flow. In *Computer Vision–ECCV 2016: 14th European Conference, Amsterdam, The Netherlands, October 11–14, 2016, Proceedings, Part IV 14*, pages 286–301. Springer, 2016. 42

[227] C. Zhuang, A. L. Zhai, and D. Yamins. Local aggregation for unsupervised learning of visual embeddings. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 6002–6012, 2019. 7