Journal of Machine Learning Research 23 (2022) 1-103 Submitted 6/21; Revised 2/22; Published 3/22

ReduNet: A White-box Deep Network from the Principle of Maximizing Rate Reduction*

Kwan Ho Ryan Chan [†,◇] RYANCHANKH@BERKELEY.EDU
Yaodong Yu [†,◇] YYU@EECS.BERKELEY.EDU
Chong You [†,◇,♮] CYOU@BERKELEY.EDU
Haozhi Qi [†] HQI@BERKELEY.EDU
John Wright [‡] JOHNWRIGHT@EE.COLUMBIA.EDU
Yi Ma [†] YIMA@EECS.BERKELEY.EDU

[†] *Department of Electrical Engineering and Computer Sciences*
University of California, Berkeley, CA 94720-1776, USA
[‡] *Department of Electrical Engineering*
Department of Applied Physics and Applied Mathematics
Columbia University, New York, NY, 10027, USA

Editor: Joan Bruna

Abstract

This work attempts to provide a plausible theoretical framework that aims to interpret modern deep (convolutional) networks from the principles of data compression and discriminative representation. We argue that for high-dimensional multi-class data, the optimal linear discriminative representation maximizes the coding rate difference between the whole dataset and the average of all the subsets. We show that the basic iterative gradient ascent scheme for optimizing the rate reduction objective naturally leads to a multi-layer deep network, named ReduNet, which shares common characteristics of modern deep networks. The deep layered architectures, linear and nonlinear operators, and even parameters of the network are all explicitly constructed layer-by-layer via forward propagation, although they are amenable to fine-tuning via back propagation. All components of so-obtained "white-box" network have precise optimization, statistical, and geometric interpretation. Moreover, all linear operators of the so-derived network naturally become multi-channel convolutions when we enforce classification to be rigorously shift-invariant. The derivation in the invariant setting suggests a trade-off between sparsity and invariance, and also indicates that such a deep convolution network is significantly more efficient to construct and learn in the spectral domain. Our preliminary simulations and experiments clearly verify the effectiveness of both the rate reduction objective and the associated ReduNet. All code and data are available at `https://github.com/Ma-Lab-Berkeley`.

Keywords: rate reduction, white-box deep network, linear discriminative representation, multi-channel convolution, sparsity and invariance trade-off

*. This paper integrates previous two manuscripts: `https://arxiv.org/abs/2006.08558` and `https://arxiv.org/abs/2010.14765`, with significantly improved organization, presentation, and new results. The first manuscript appeared as a conference paper in NeurIPS 2020 (Yu et al., 2020). The second manuscript is the new addition.

◇. The first three authors contributed equally to this work.

♮. Now at Google Research, New York, NY 10011, USA. Contact: cyou@google.com.

1. Introduction and Overview

1.1 Background and Motivation

In the past decade or so, the practice of deep networks has captured people's imagination by its empirical successes in learning useful representations from large-scale real-world data such as images, speech, and natural languages (LeCun et al., 2015). To a large extent, the dramatic revival of deep networks is attributed to remarkable technological advancements in scalable computation platforms (hardware and software) and equally importantly, ample data, real or simulated, for training and evaluation of the networks (Krizhevsky et al., 2012).

The practice of deep networks as a "black box." Nevertheless, the practice of deep networks has been shrouded with mystery from the very beginning. This is largely caused by the fact that deep network architectures and their components are often designed based on years of *trial and error*, then trained from random initialization via back propagation (BP) (Rumelhart et al., 1986), and then deployed as a "black box". Many of the popular techniques and recipes for designing and training deep networks (to be surveyed in the related work below) were developed through heuristic and empirical means, as opposed to rigorous mathematical principles, modeling and analysis. Due to lack of clear mathematical principles that can guide the design and optimization, numerous techniques, tricks, and even hacks need to be added upon the above designing and training process to make deep learning "work at all" or "work better" on real world data and tasks. Practitioners constantly face a series of challenges for any new data and tasks: What architecture or particular components they should use for the network? How wide or deep the network should be? Which parts of the networks need to be trained and which can be determined in advance? Last but not the least, after the network has been trained, learned and tested: how to interpret functions of the operators; what are the roles and relationships among the multiple (convolution) channels; how can they be further improved or adapted to new tasks? Besides empirical evaluation, it is challenging to provide rigorous guarantees for certain properties of so obtained networks, such as invariance and robustness. Recent studies have shown that popular networks in fact are not invariant to common deformations (Azulay and Weiss, 2019; Engstrom et al., 2019); they are prone to overfit noisy or even arbitrary labels (Zhang et al., 2017); and they remain vulnerable to adversarial attacks (Szegedy et al., 2013); or they suffer catastrophic forgetting when incrementally trained to learn new classes/tasks (McCloskey and Cohen, 1989; Delange et al., 2021; Wu et al., 2021). The lack of understanding about the roles of learned operators in the networks often provoke debates among practitioners which architecture is better than others, for example MLPs versus CNNs versus Transformers (Tay et al., 2021) etc. This situation seems in desperate need of improvements if one wishes deep learning to be a science rather than an "alchemy." Therefore, it naturally raises a fundamental question that we aim to address in this work: *how to develop a principled mathematical framework for better understanding and design of deep networks?*

Existing attempts to interpret or understand deep networks. To uncover the mystery of deep networks, a plethora of recent studies on the mathematics of deep learning has, to a large extent, improved our understanding on key aspects of deep models, including over-fitting and over-parameterization (Arora et al., 2019; Belkin et al., 2019; Yang et al., 2020), optimization (Jin et al., 2017, 2018; Gidel et al., 2019; Gunasekar et al., 2018);

expressive power (Rolnick and Tegmark, 2018; Hanin, 2017), generalization bounds (Bartlett et al., 2017; Golowich et al., 2018) etc. Nevertheless, these efforts are typically based on simplified models (e.g. networks with only a handful layers (Zhong et al., 2017; Soltanolkotabi et al., 2018; Zhang et al., 2019c; Mei and Montanari, 2021)); or results are conservative (e.g. generalization bounds (Bartlett et al., 2017; Golowich et al., 2018)); or rely on simplified assumptions (e.g. ultra-wide deep networks (Jacot et al., 2018; Du et al., 2019; Allen-Zhu et al., 2019; Buchanan et al., 2021)); or only explain or justify one of the components or characteristics of the network (e.g. dropout (Cavazza et al., 2018; Mianjy et al., 2018; Wei et al., 2020)). On the other hand, the predominant methodology in practice still remains *trial and error*. In fact, this is often pushed to the extreme by searching for effective network structures and training strategies through extensive random search techniques, such as Neural Architecture Search (Zoph and Le, 2016; Baker et al., 2017), AutoML (Hutter et al., 2019), and Learning to Learn (Andrychowicz et al., 2016).

A new theoretical framework based on data compression and representation. Our approach deviates from the above efforts in a very significant way. Most of the existing theoretical work view the deep networks themselves as the object for study. They try to understand why deep networks work by examining their capabilities in fitting certain input-output relationships (for given class labels or function values). We, in this work, however, advocate to shift the attention of study back to the data and try to understand what deep networks should do. We start our investigation with a fundamental question: What exactly do we try to *learn* from and about the data? With the objective clarified, maybe deep networks, with *all* their characteristics, are simply necessary means to achieve such an objective. More specifically, in this paper, we develop a new theoretical framework for understanding deep networks around the following two questions:

1. *Objective of Representation Learning:* What intrinsic structures of the data should we learn, and how should we represent such structures? What is a principled objective function for learning a good representation of such structures, instead of choosing heuristically or arbitrarily?
2. *Architecture of Deep Networks:* Can we justify the structures of modern deep networks from such a principle? In particular, can the networks' layered architecture and operators (linear or nonlinear) all be derived from this objective, rather than designed heuristically and evaluated empirically?

This paper will provide largely positive and constructive answers to the above questions. We will argue that, at least in the classification setting, a principled objective for a deep network is *to learn a low-dimensional linear discriminative representation of the data* (Section 1.3). The optimality of such a representation can be evaluated by a principled measure from (lossy) data compression, known as *rate reduction* (Section 2). Appropriately structured deep networks can be naturally interpreted as *optimization schemes for maximizing this measure* (Section 3 and 4). Not only does this framework offer new perspectives to understand and interpret modern deep networks, they also provide new insights that can potentially change and improve the practice of deep networks. For instance, the resulting networks will be entirely a "white box" and back propagation from random initialization is no longer the only choice for training the networks (as we will verify through extensive experiments in Section 5).

For the rest of this introduction, we will first provide a brief survey on existing work that are related to the above two questions. Then we will give an overview of our approach and contributions before we delve into the technical details in following sections.

1.2 Related Work

Given a random vector $\boldsymbol{x} \in \mathbb{R}^D$ which is drawn from a mixture of k distributions $\mathcal{D} = \{\mathcal{D}^j\}_{j=1}^k$, one of the most fundamental problems in machine learning is how to effectively and efficiently *learn the distribution* from a finite set of i.i.d samples, say $\boldsymbol{X} = [\boldsymbol{x}^1, \boldsymbol{x}^2, \ldots, \boldsymbol{x}^m] \in \mathbb{R}^{D\times m}$. To this end, we *seek a good representation* through a continuous mapping, $f(\boldsymbol{x}, \boldsymbol{\theta}) : \mathbb{R}^D \to \mathbb{R}^n$, that captures intrinsic structures of $\boldsymbol{x}$ and best facilitates subsequent tasks such as classification or clustering.[1]

1.2.1 Objectives for Deep Learning

Supervised learning via cross entropy. To ease the task of learning $\mathcal{D}$, in the popular supervised setting, a true class label, represented as a one-hot vector $\boldsymbol{y}^i \in \mathbb{R}^k$, is given for each sample $\boldsymbol{x}^i$. Extensive studies have shown that for many practical datasets (images, audio, and natural language, etc.), the mapping from the data $\boldsymbol{x}$ to its class label $\boldsymbol{y}$ can be effectively modeled by training a deep network (Goodfellow et al., 2016), here denoted as $f(\boldsymbol{x}, \boldsymbol{\theta}) : \boldsymbol{x} \mapsto \boldsymbol{y}$ with network parameters $\boldsymbol{\theta} \in \Theta$. This is typically done by minimizing the *cross-entropy loss* over a training set $\{(\boldsymbol{x}^i, \boldsymbol{y}^i)\}_{i=1}^m$, through backpropagation over the network parameters $\boldsymbol{\theta}$:

$$\min_{\boldsymbol{\theta}\in\Theta} \mathrm{CE}(\boldsymbol{\theta}, \boldsymbol{x}, \boldsymbol{y}) \doteq -\mathbb{E}[\langle \boldsymbol{y}, \log[f(\boldsymbol{x}, \boldsymbol{\theta})]\rangle] \approx -\frac{1}{m}\sum_{i=1}^{m} \langle \boldsymbol{y}^i, \log[f(\boldsymbol{x}^i, \boldsymbol{\theta})]\rangle. \tag{1}$$

Despite its effectiveness and enormous popularity, there are two serious limitations with this approach: 1) It aims only to predict the labels $\boldsymbol{y}$ even if they might be mislabeled. Empirical studies show that deep networks, used as a "black box," can even fit random labels (Zhang et al., 2017). 2) With such an end-to-end data fitting, despite plenty of empirical efforts in trying to interpret the so-learned features (Zeiler and Fergus, 2014), it is not clear to what extent the intermediate features learned by the network capture the intrinsic structures of the data that make meaningful classification possible in the first place. Recent work of Papyan et al. (2020); Fang et al. (2021); Zhu et al. (2021) shows that the representations learned via the cross-entropy loss (1) exhibit a *neural collapsing* phenomenon,[2] where within-class variability and structural information are getting suppressed and ignored, as we will also see in the experiments. The precise geometric and statistical properties of the learned features are also often obscured, which leads to the lack of interpretability and subsequent performance

1. Classification is where deep learning demonstrated the initial success that has catalyzed the explosive interest in deep networks (Krizhevsky et al., 2012). Although our study in this paper focuses on classification, we believe the ideas and principles can be naturally generalized to other settings such as regression.
2. Essentially, once an over-parameterized network fits the training data, regularization (such as weight decay) would collapse weight components or features that are not the most relevant for fitting the class labels. Besides the most salient feature, informative and discriminative features that also help define a class can be suppressed.

guarantees (e.g., generalizability, transferability, and robustness, etc.) in deep learning. Therefore, *one of the goals of this work is to address such limitations by reformulating the objective towards learning explicitly meaningful and useful representations for the data $\boldsymbol{x}$, in terms of feature linearity, discriminativeness, and richness.*

Minimal versus low-dimensional representations from deep learning. Based on strong empirical evidence that intrinsic structures of high-dimensional (imagery) data are rather low-dimensional[3], it has been long believed that the role of deep networks is to learn certain (nonlinear) low-dimensional representations of the data, which has an advantage over classical linear dimension reduction methods such as PCA (Hinton and Salakhutdinov, 2006). Following this line of thought, one possible approach to interpret the role of deep networks is to view outputs of intermediate layers of the network as selecting certain low-dimensional latent features $\boldsymbol{z} = f(\boldsymbol{x}, \boldsymbol{\theta}) \in \mathbb{R}^n$ of the data that are discriminative among multiple classes. Learned representations $\boldsymbol{z}$ then facilitate the subsequent classification task for predicting the class label $\boldsymbol{y}$ by optimizing a classifier $g(\boldsymbol{z})$:

$$\boldsymbol{x} \xrightarrow{\quad f(\boldsymbol{x},\boldsymbol{\theta}) \quad} \boldsymbol{z}(\boldsymbol{\theta}) \xrightarrow{\quad g(\boldsymbol{z}) \quad} \boldsymbol{y}. \tag{2}$$

The *information bottleneck* (IB) formulation (Tishby and Zaslavsky, 2015) further hypothesizes that the role of the network is to learn $\boldsymbol{z}$ as the minimal sufficient statistics for predicting $\boldsymbol{y}$. Formally, it seeks to maximize the mutual information $I(\boldsymbol{z}, \boldsymbol{y})$ (Cover and Thomas, 2006) between $\boldsymbol{z}$ and $\boldsymbol{y}$ while minimizing $I(\boldsymbol{x}, \boldsymbol{z})$ between $\boldsymbol{x}$ and $\boldsymbol{z}$:

$$\max_{\boldsymbol{\theta} \in \Theta} \ \mathrm{IB}(\boldsymbol{x}, \boldsymbol{y}, \boldsymbol{z}(\boldsymbol{\theta})) \doteq I(\boldsymbol{z}(\boldsymbol{\theta}), \boldsymbol{y}) - \beta I(\boldsymbol{x}, \boldsymbol{z}(\boldsymbol{\theta})), \quad \beta > 0. \tag{3}$$

Given one can overcome some caveats associated with this framework (Kolchinsky et al., 2019), such as how to accurately evaluate mutual information with finitely samples of degenerate distributions, this framework can be helpful in explaining certain behaviors of deep networks. For example, recent work (Papyan et al., 2020) indeed shows that the representations learned via the cross-entropy loss expose a *neural collapse* phenomenon. That is, features of each class are mapped to a one-dimensional vector whereas all other information of the class is suppressed. More discussion on neural collapse will be given in Section 2.3. But by being task-dependent (depending on the label $\boldsymbol{y}$) and seeking a *minimal* set of most informative features for the task at hand (for predicting the label $\boldsymbol{y}$ only), the so learned network may sacrifice robustness in case the labels can be corrupted or transferrability when we want to use the features for different tasks (Hui et al., 2022; Kornblith et al., 2021). To address this, *our framework uses the label $\boldsymbol{y}$ as only side information to assist learning discriminative yet diverse (not minimal) representations; these representations optimize a different intrinsic objective based on the principle of rate reduction.*[4]

Reconciling contractive and contrastive learning. Complementary to the above supervised discriminative approach, *auto-encoding* (Baldi and Hornik, 1989; Kramer, 1991;

3. For example, the digits in MNIST approximately live on a manifold with intrinsic dimension no larger than 15 (Hein and Audibert, 2005), the images in CIFAR-10 live closely on a 35-dimensional manifold (Spigler et al., 2019), and the images in ImageNet have intrinsic dimension of ~ 40 (Pope et al., 2021).
4. As we will see in the experiments in Section 5, indeed this makes learned features much more robust to mislabeled data.

Hinton and Salakhutdinov, 2006) is another popular *unsupervised* (label-free) framework used to learn good latent representations, which can be viewed as a nonlinear extension to the classical PCA (Jolliffe, 2002). The idea is to learn a compact latent representation $\boldsymbol{z} \in \mathbb{R}^n$ that adequately regenerates the original data $\boldsymbol{x}$ to certain extent, through optimizing decoder or generator $g(\boldsymbol{z}, \boldsymbol{\eta})$:

$$\boldsymbol{x} \xrightarrow{\quad f(\boldsymbol{x},\boldsymbol{\theta}) \quad} \boldsymbol{z}(\boldsymbol{\theta}) \xrightarrow{\quad g(\boldsymbol{z},\boldsymbol{\eta}) \quad} \widehat{\boldsymbol{x}}(\boldsymbol{\theta}, \boldsymbol{\eta}). \tag{4}$$

Typically, such representations are learned in an end-to-end fashion by imposing certain heuristics on geometric or statistical "compactness" of $\boldsymbol{z}$, such as its dimension, energy, or volume. For example, the *contractive* autoencoder (Rifai et al., 2011) penalizes local volume expansion of learned features approximated by the Jacobian

$$\min_{\boldsymbol{\theta}} \left\| \frac{\partial \boldsymbol{z}}{\partial \boldsymbol{\theta}} \right\|. \tag{5}$$

When the data contain complicated *multi-modal low-dimensional* structures, naive heuristics or inaccurate metrics may fail to capture all internal subclass structures or to explicitly discriminate among them for classification or clustering purposes. For example, one consequence of this is the phenomenon of *mode collapsing* in learning generative models for data that have mixed multi-modal structures (Li et al., 2020). To address this, *we propose a principled rate reduction measure (on $\boldsymbol{z}$) that promotes both the within-class compactness and between-class discrimination of the features for data with mixed structures.*

If the above contractive learning seeks to reduce the dimension of the learned representation, *contrastive learning* (Hadsell et al., 2006; Oord et al., 2018; He et al., 2020) seems to do just the opposite. For data that belong to k different classes, a randomly chosen pair $(\boldsymbol{x}^i, \boldsymbol{x}^j)$ is of high probability belonging to difference classes if k is large.[5] Hence it is desirable that the representation $\boldsymbol{z}^i = f(\boldsymbol{x}^i, \boldsymbol{\theta})$ of a sample $\boldsymbol{x}^i$ should be highly incoherent to those $\boldsymbol{z}^j$ of other samples $\boldsymbol{x}^j$ whereas coherent to feature of its transformed version $\tau(\boldsymbol{x}^i)$, denoted as $\boldsymbol{z}(\tau(\boldsymbol{x}^i))$ for τ in certain augmentation set $\mathcal{T}$ in consideration. Hence it was proposed heuristically that, to promote discriminativeness of the learned representation, one may seek to minimize the so-called *contrastive loss*:

$$\min_{\boldsymbol{\theta}} -\log \frac{\exp(\langle \boldsymbol{z}^i, \boldsymbol{z}(\tau(\boldsymbol{x}^i))\rangle)}{\sum_{j \neq i} \exp(\langle \boldsymbol{z}^i, \boldsymbol{z}^j \rangle)}, \tag{6}$$

which is small whenever the inner product $\langle \boldsymbol{z}^i, \boldsymbol{z}(\tau(\boldsymbol{x}^i))\rangle$ is large and $\langle \boldsymbol{z}^i, \boldsymbol{z}^j \rangle$ is small for $i \neq j$.

As we may see from the practice of both contractive learning and contrastive learning, for a good representation of the given data, people have striven to achieve certain trade-off between the compactness and discriminativeness of the representation. Contractive learning aims to compress the features of the entire ensemble, whereas contrastive learning aims to expand features of any pair of samples. Hence it is not entirely clear why either of these two seemingly opposite heuristics seems to help learn good features. Could it be the case that both mechanisms are needed but each acts on different part of the data? *As we will*

5. For example, when $k \geq 100$, a random pair is of probability 99% belonging to different classes.

see, the rate reduction principle precisely reconciles the tension between these two seemingly contradictory objectives by explicitly specifying to compress (or contract) similar features in each class whereas to expand (or contrast) the set of all features in multiple classes.

1.2.2 Architectures for Deep Networks

The ultimate goal of any good theory for deep learning is to facilitate a better understanding of deep networks and to design better network architectures and algorithms with performance guarantees. So far we have surveyed many popular objective functions that promote certain desired properties of the learned representation $\boldsymbol{z} = f(\boldsymbol{x}, \boldsymbol{\theta})$ of the data $\boldsymbol{x}$. The remaining question is how the mapping $f(\boldsymbol{x}, \boldsymbol{\theta})$ should be modeled and learned.

Empirical designs of deep (convolution) neural networks. The current popular practice is to model the mapping with an empirically designed artificial deep neural network and learn the parameters $\boldsymbol{\theta}$ from random initialization via backpropagation (BP) (Rumelhart et al., 1986). Starting with the AlexNet (Krizhevsky et al., 2012), the architectures of modern deep networks continue to be empirically revised and improved. Network architectures such as VGG (Simonyan and Zisserman, 2015), ResNet (He et al., 2016), DenseNet (Huang et al., 2017), Recurrent CNN or LSTM (Hochreiter and Schmidhuber, 1997), and mixture of experts (MoE) (Shazeer et al., 2017) etc. have continued to push the performance envelope.

As part of the effort to improve deep networks' performance, almost every component of the networks has been scrutinized empirically and various revisions and improvements have been proposed. They are not limited to the nonlinear activation functions (Maas et al., 2013; Klambauer et al., 2017; Xu et al., 2015; Nwankpa et al., 2018; Hendrycks and Gimpel, 2016; Martens et al., 2021), skip connections (Ronneberger et al., 2015; He et al., 2016; Zhang et al., 2018a), normalizations (Ioffe and Szegedy, 2015; Ba et al., 2016; Ulyanov et al., 2016; Wu and He, 2018; Miyato et al., 2018), up/down sampling or pooling (Scherer et al., 2010), convolutions (LeCun et al., 1998; Krizhevsky et al., 2012), etc. Nevertheless, almost all such modifications are developed through years of empirical trial and error or ablation study. Some recent practices even take to the extreme by searching for effective network structures and training strategies through extensive random search techniques, such as Neural Architecture Search (Zoph and Le, 2016; Baker et al., 2017), AutoML (Hutter et al., 2019), and Learning to Learn (Andrychowicz et al., 2016).

However, there has been apparent lack of direct justification of the resulting network architectures from the desired learning objectives, e.g. cross entropy or contrastive learning. As a result, it is challenging if not impossible to rigorously justify why the resulting network is the best suited for the objective, let alone to interpret the learned operators and parameters inside. In this work, *we will attempt to derive network architectures and components as entirely a "white box" from the desired objective (say, rate reduction).*

Constructive approaches to deep (convolution) networks. For long, people have noticed structural similarities between deep networks and iterative optimization algorithms, especially those for solving sparse coding. Even before the revival of deep networks, Gregor and LeCun (2010) has argued that algorithms for sparse coding, such as the FISTA algorithm (Beck and Teboulle, 2009), can be viewed as a deep network and be trained using BP for better coding performance, known as LISTA (learned ISTA). Later Papyan et al. (2017); Giryes et al. (2018); Monga et al. (2019); Sun et al. (2020) have proposed similar interpretations

of deep networks as unrolling algorithms for sparse coding in convolutional or recurrent settings. However, it remains unclear about the role of the convolutions (dictionary) in each layer and exactly why such low-level sparse coding is needed for the high-level classification task. To a large extent, *this work will provide a new perspective to elucidate the role of the sparsifying convolutions in a deep network: not only will we reveal why sparsity is needed for ensuring invariant classification but also the (multi-channel) convolution operators can be explicitly derived and constructed.*

Almost all of the above networks inherit architectures and initial parameters from sparse coding algorithms, but still rely on back propagation (Rumelhart et al., 1986) to tune these parameters. There have been efforts that try to construct the network in a *purely forward fashion*, without any back propagation. For example, to ensure translational invariance for a wide range of signals, Bruna and Mallat (2013); Wiatowski and Bölcskei (2018) have proposed to use wavelets to construct convolution networks, known as ScatteringNets. As a ScatteringNet is oblivious to the given data and feature selection for classification, the required number of convolution kernels grow exponentially with the depth. Zarka et al. (2020, 2021) have also later proposed hybrid deep networks based on scattering transform and dictionary learning to alleviate scalability. Alternatively, Chan et al. (2015) has proposed to construct much more compact (arguably the simplest) networks using principal components of the input data as the convolution kernels, known as PCANets. However, in both cases of ScatteringNets and PCANets the forward-constructed networks seek a representation of the data that is not directly related to a specific (classification) task. To resolve limitations of both the ScatteringNet and the PCANet, *this work shows how to construct a data-dependent deep convolution network in a forward fashion that leads to a discriminative representation directly beneficial to the classification task.* More discussion about the relationships between our construction and these networks will be given in Section 4.5 and Appendix E.5.

1.3 A Principled Objective for Discrimiative Representation via Compression

Whether the given data $\boldsymbol{X}$ of a mixed distribution $\mathcal{D} = \{\mathcal{D}^j\}_{j=1}^k$ can be effectively classified depends on how separable (or discriminative) the component distributions $\mathcal{D}^j$ are (or can be made). One popular working assumption is that the distribution of each class has relatively *low-dimensional* intrinsic structures. There are several reasons why this assumption is plausible: 1). High dimensional data are highly redundant; 2). Data that belong to the same class should be similar and correlated to each other; 3). Typically we only care about equivalent structures of $\boldsymbol{x}$ that are invariant to certain classes of deformation and augmentations. Hence we may assume the distribution $\mathcal{D}^j$ of each class has a support on a low-dimensional submanifold, say $\mathcal{M}^j$ with dimension $d_j \ll D$, and the distribution $\mathcal{D}$ of $\boldsymbol{x}$ is supported on the mixture of those submanifolds, $\mathcal{M} = \cup_{j=1}^k \mathcal{M}^j$, in the high-dimensional ambient space $\mathbb{R}^D$, as illustrated in Figure 1 left.

With the manifold assumption in mind, we want to learn a mapping $\boldsymbol{z} = f(\boldsymbol{x}, \boldsymbol{\theta})$ that maps each of the submanifolds $\mathcal{M}^j \subset \mathbb{R}^D$ to a *linear* subspace $\mathcal{S}^j \subset \mathbb{R}^n$ (see Figure 1 middle). To do so, we require our learned representation to have the following properties, called a *linear discriminative representation* (LDR):

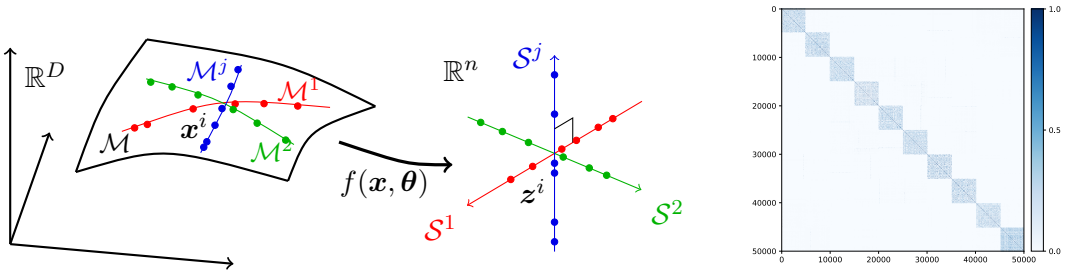


Figure 1: **Left and Middle:** The distribution $\mathcal{D}$ of high-dim data $\boldsymbol{x} \in \mathbb{R}^D$ is supported on a manifold $\mathcal{M}$ and its classes on low-dim submanifolds $\mathcal{M}^j$, we learn a map $f(\boldsymbol{x}, \boldsymbol{\theta})$ such that $\boldsymbol{z}^i = f(\boldsymbol{x}^i, \boldsymbol{\theta})$ are on a union of maximally uncorrelated subspaces $\{\mathcal{S}^j\}$. **Right:** Cosine similarity between learned features by our method for the CIFAR10 training dataset. Each class has 5,000 samples and their features span a subspace of over 10 dimensions (see Figure 12(c)).

1. *Within-Class Compressible:* Features of samples from the same class/cluster should be relatively *correlated* in a sense that they belong to a low-dimensional linear subspace.[6]

2. *Between-Class Discriminative:* Features of samples from different classes/clusters should be highly *uncorrelated* and belong to different low-dimensional linear subspaces.

3. *Diverse Representation:* Dimension (or variance) of features for each class/cluster should be *as large as possible* as long as they stay uncorrelated from the other classes.

The third item is desired because we want the learned features to reveal all possible causes why this class is different from all other classes[7] (see Section 2 for more detailed justification).

Notice that the first two items align well with the spirit of the classic *linear discriminant analysis* (LDA) (Hastie et al., 2009). Here, however, although the intrinsic structures of each class/cluster may be low-dimensional, they are by no means simply linear (or Gaussian) in their original representation $\boldsymbol{x}$ and they need be to made linear through a nonlinear transform $\boldsymbol{z} = f(\boldsymbol{x})$. Unlike LDA (or similarly SVM[8]), here we do not directly seek a discriminant (linear) classifier. Instead, we use the nonlinear transform to seek a *linear discriminative representation*[9] (LDR) for the data such that the subspaces that represent all the classes are maximally incoherent. To some extent, the resulting multiple subspaces $\{\mathcal{S}^j\}$

6. Linearity is a desirable property for many reasons. For example, in engineering, it makes interpolating/extrapolating data easy via superposition which is very useful for generative purposes. There is also scientific evidence that the brain represents objects such as faces as a linear subspace (Chang and Tsao, 2017).
7. For instance, to tell "Apple" from "Orange", not only do we care about the color, but also the shape and the leaves. Interestingly, some images of computers are labeled as "Apple" too.
8. For instance, Vinyals et al. (2012) has proposed to use SVMs in a recursive fashion to build (deep) nonlinear classifiers for complex data.
9. In this work, to avoid confusion, we always use the word "discriminative" to describe a representation, and "discriminant" a classifier. To some extent, one may view classification and representation are *dual* to each other. Discriminant methods (LDA or SVM) are typically more natural for two-class settings (despite many extensions to multiple classes), whereas discriminative representations are natural for multi-class data and help reveal the data intrinsic structures more directly.

can be viewed as discriminative *generalized principal components* (Vidal et al., 2016) or, if orthogonal, *independent components* (Hyvärinen and Oja, 2000) of the resulting features $\boldsymbol{z}$ for the original data $\boldsymbol{x}$. As we will see in Section 3, deep networks precisely play the role of modeling this nonlinear transform from the data to an LDR!

For many clustering or classification tasks (such as object detection in images), we consider two samples as *equivalent* if they differ by certain classes of domain deformations or augmentations $\mathcal{T} = \{\tau\}$. Hence, we are only interested in low-dimensional structures that are *invariant* to such deformations (i.e., $\boldsymbol{x} \in \mathcal{M}$ iff $\tau(\boldsymbol{x}) \in \mathcal{M}$ for all $\tau \in \mathcal{T}$), which are known to have sophisticated geometric and topological structures (Wakin et al., 2005) and can be difficult to learn precisely in practice even with rigorously designed CNNs (Cohen and Welling, 2016; Cohen et al., 2019). In our framework, this is formulated in a very natural way: all equivariant instances are to be embedded into the same subspace, so that the subspace itself is invariant to the transformations under consideration (see Section 4).

There are previous attempts to directly enforce subspace structures on features learned by a deep network for supervised (Lezama et al., 2018) or unsupervised learning (Ji et al., 2017; Zhang et al., 2018b; Peng et al., 2017; Zhou et al., 2018; Zhang et al., 2019b,a; Lezama et al., 2018). However, the *self-expressive* property of subspaces exploited by these work does not enforce all the desired properties listed above as shown by Haeffele et al. (2021). Recently Lezama et al. (2018) has explored a nuclear norm based geometric loss to enforce orthogonality between classes, but does not promote diversity in the learned representations, as we will soon see. Figure 1 right illustrates a representation learned by our method on the CIFAR10 dataset. More details can be found in the experimental Section 5.

In this work, to learn a discriminative linear representation for intrinsic low-dimensional structures from high-dimensional data, we propose an information-theoretic measure that maximizes the coding rate difference between the whole dataset and the sum of each individual class, known as *rate reduction*. This new objective provides a more unifying view of above objectives such as cross-entropy, information bottleneck, contractive and contrastive learning. *We can rigorously show that when the intrinsic dimensions the submanifolds are known and this objective is optimized, the resulting representation indeed has the desired properties listed above.*

1.4 A Constructive Approach to Deep Networks via Optimization

Despite tremendous advances made by numerous empirically designed deep networks, there is still a lack of rigorous theoretical justification of the need or reason for "deep layered" architectures[10] and a lack of fundamental understanding of the roles of the associated operators, e.g. linear (multi-channel convolution) and nonlinear activation in each layer. Although many works mentioned in Section 1.2.2 suggest the layered architectures might be interpreted as unrolled optimization algorithms (say for sparse coding), there is lack of explicit and direct connection between such algorithms and the objective (say, minimizing the cross entropy for classification). As a result, there is lack of principles for network design: How wide or deep should the network be? What has improved about features learned between adjacent layers? Why are multi-channel convolutions necessary for image

10. After all, at least in theory, Barron (1991) already proved back in early 90's that a single-layer neural network can efficiently approximate a very general class of functions or mappings.

classification instead of separable convolution kernels[11]? Can the network parameters be better initialized than purely randomly?

In this paper, we attempt to provide some answers to the above questions and offer a plausible interpretation of deep neural networks by deriving a class of deep (convolution) networks from the perspective of learning an LDR for the data. We contend that all key features and structures of modern deep (convolution) neural networks can be naturally derived from optimizing the *rate reduction* objective, which seeks an optimal (invariant) linear discriminative representation of the data. More specifically, the basic iterative *projected gradient ascent* scheme for optimizing this objective naturally takes the form of a deep neural network, one layer per iteration. In this framework, the width of the network assumes a precise role as the *statistical resource* needed to preserve the low-dimensional (separable) structures of the data whereas the network depth as the *computational resource* needed to map the (possibly nonlinear) structures to a linear discriminative representation.

This principled approach brings a couple of nice surprises: First, architectures, operators, and parameters of the network can be constructed explicitly layer-by-layer in a *forward propagation* fashion, and all inherit precise optimization, statistical and geometric interpretation. As a result, the so constructed "white-box" deep network already gives a rather discriminative representation for the given data even *without any back propagation training* (see Section 3). Nevertheless, the so-obtained network is actually amenable to be further fine-tuned by back propagation for better performance, as our experiments will show. Second, in the case of seeking a representation *rigorously* invariant to shift or translation, the network naturally lends itself to a multi-channel convolutional network (see Section 4). Moreover, the derivation indicates such a convolutional network is computationally more efficient to construct in the *spectral (Fourier) domain*, analogous to how neurons in the visual cortex encode and transit information with their spikes (Eliasmith and Anderson, 2003; Belitski et al., 2008).

2. The Principle of Maximal Coding Rate Reduction

2.1 Measure of Compactness for Linear Representations

Although the three properties listed in Section 1.3 for linear discriminative representations (LDRs) are all highly desirable for the latent representation $\boldsymbol{z}$, they are by no means easy to obtain: Are these properties compatible so that we can expect to achieve them all at once? If so, is there a *simple but principled* objective that can measure the goodness of the resulting representations in terms of all these properties? The key to these questions is to find a principled "measure of compactness" for the distribution of a random variable $\boldsymbol{z}$ or from its finite samples $\boldsymbol{Z}$. Such a measure should directly and accurately characterize intrinsic geometric or statistical properties of the distribution, in terms of its intrinsic dimension or volume. Unlike cross-entropy (1) or information bottleneck (3), such a measure should

11. Convolution kernels/dictionaries such as wavelets have been widely practiced to model and process 2D signals such as images. Convolution kernels used in modern CNNs are nevertheless multi-channel, often involving hundreds of channels altogether at each layer. Most theoretical modeling and analysis for images have been for the 2D kernel case. The precise reason and role for multi-channel convolutions have been elusive and equivocal. This work aims to provide a rigorous explanation.

not depend exclusively on class labels so that it can work in all supervised, self-supervised, semi-supervised, and unsupervised settings.

Measure of compactness from information theory. In information theory (Cover and Thomas, 2006), the notion of entropy $H(\boldsymbol{z})$ is designed to be such a measure. However, entropy is not well-defined for continuous random variables with degenerate distributions. This is unfortunately the case for data with relatively low intrinsic dimension. The same difficulty resides with evaluating mutual information $I(\boldsymbol{x}, \boldsymbol{z})$ for degenerate distributions. To alleviate this difficulty, another related concept in information theory, more specifically in lossy data compression, that measures the "compactness" of a random distribution is the so-called *rate distortion* (Cover and Thomas, 2006): Given a random variable $\boldsymbol{z}$ and a prescribed precision $\epsilon > 0$, the rate distortion $R(\boldsymbol{z}, \epsilon)$ is the minimal number of binary bits needed to encode $\boldsymbol{z}$ such that the expected decoding error is less than ϵ, i.e., the decoded $\widehat{\boldsymbol{z}}$ satisfies $\mathbb{E}[\|\boldsymbol{z} - \widehat{\boldsymbol{z}}\|_2] \leq \epsilon$. This quantity has been shown to be useful in explaining feature selection in deep networks (MacDonald et al., 2019). However, the rate distortion of an arbitrary high-dimensional distribution is intractable, if not impossible, to compute, except for simple distributions such as discrete and Gaussian.[12] Nevertheless, as we have discussed in Section 1.3, our goal here is to learn a final representation of the data as linear subspaces. Hence we only need a measure of compactness/goodness for this class of distributions. Fortunately, as we will explain below, the rate distortions for this class of distributions can be accurately and easily computed, actually in closed form!

Rate distortion for finite samples on a subspace. Another practical difficulty in evaluating the rate distortion is that we normally do not know the distribution of $\boldsymbol{z}$. Instead, we have a finite number of samples as learned representations $\{\boldsymbol{z}^i = f(\boldsymbol{x}^i, \boldsymbol{\theta}) \in \mathbb{R}^n, i = 1, \ldots, m\}$, for the given data samples $\boldsymbol{X} = [\boldsymbol{x}^1, \ldots, \boldsymbol{x}^m]$. Fortunately, Ma et al. (2007) provides a precise estimate on the number of binary bits needed to encode finite samples from a subspace-like distribution. In order to encode the learned representation $\boldsymbol{Z} = [\boldsymbol{z}^1, \ldots, \boldsymbol{z}^m]$ up to a precision, say ϵ, the total number of bits needed is given by the following expression: $\mathcal{L}(\boldsymbol{Z}, \epsilon) \doteq \left(\frac{m+n}{2}\right) \log\det\left(\boldsymbol{I} + \frac{n}{m\epsilon^2}\boldsymbol{Z}\boldsymbol{Z}^*\right)$.[13] This formula can be derived either by packing ϵ-balls into the space spanned by $\boldsymbol{Z}$ as a Gaussian source or by computing the number of bits needed to quantize the SVD of $\boldsymbol{Z}$ subject to the precision, see Ma et al. (2007) for proofs. Therefore, the compactness of learned features *as a whole* can be measured in terms of the average coding length per sample (as the sample size m is large), a.k.a. the *coding rate* subject to the distortion ϵ:

$$R(\boldsymbol{Z}, \epsilon) \doteq \frac{1}{2} \log\det\left(\boldsymbol{I} + \frac{n}{m\epsilon^2}\boldsymbol{Z}\boldsymbol{Z}^*\right). \tag{7}$$

See Figure 2 for an illustration.

Rate distortion of samples on a mixture of subspaces. In general, the features $\boldsymbol{Z}$ of multi-class data may belong to multiple low-dimensional subspaces. To evaluate the rate distortion of such mixed data *more accurately*, we may partition the data $\boldsymbol{Z}$ into multiple

12. The same difficulties lie with the information bottleneck framework (Tishby and Zaslavsky, 2015) where one needs to evaluate (difference of) mutual information for degenerate distributions in a high-dimensional space (3).
13. We use superscript * to indicate (conjugate) transpose of a vector or a matrix

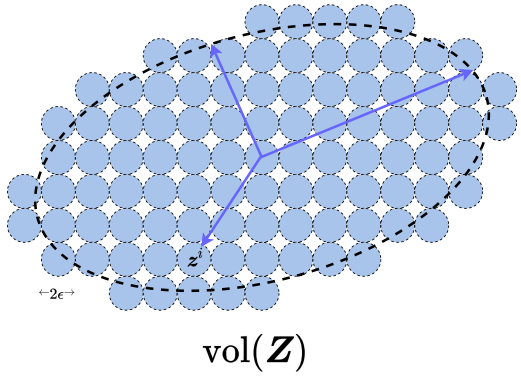


Figure 2: Lossy coding scheme: Given a precision ϵ, we pack the space/volume spanned by the data $\boldsymbol{Z}$ with small balls of diameter 2ϵ. The number of balls needed to pack the space gives the number of bits needed to record the location of each data point $\boldsymbol{z}^i$, up to the given precision $\mathbb{E}[\|\boldsymbol{z} - \widehat{\boldsymbol{z}}\|_2] \leq \epsilon$.

subsets: $\boldsymbol{Z} = \boldsymbol{Z}^1 \cup \boldsymbol{Z}^2 \cup \ldots \cup \boldsymbol{Z}^k$, with each $\boldsymbol{Z}^j$ containing samples in one low-dimensional subspace.[14] So the above coding rate (7) is accurate for each subset. For convenience, let $\boldsymbol{\Pi} = \{\boldsymbol{\Pi}^j \in \mathbb{R}^{m\times m}\}_{j=1}^k$ be a set of diagonal matrices whose diagonal entries encode the membership of the m samples in the k classes. More specifically, the diagonal entry $\boldsymbol{\Pi}^j(i,i)$ of $\boldsymbol{\Pi}^j$ indicates the probability of sample i belonging to subset j. Therefore $\boldsymbol{\Pi}$ lies in a simplex: $\Omega \doteq \{\boldsymbol{\Pi} \mid \boldsymbol{\Pi}^j \geq \boldsymbol{0},\ \boldsymbol{\Pi}^1 + \cdots + \boldsymbol{\Pi}^k = \boldsymbol{I}\}$. Then, according to Ma et al. (2007), with respect to this partition, the average number of bits per sample (the coding rate) is

$$R_c(\boldsymbol{Z}, \epsilon \mid \boldsymbol{\Pi}) \doteq \sum_{j=1}^{k} \frac{\mathsf{tr}(\boldsymbol{\Pi}^j)}{2m} \log\det\left(\boldsymbol{I} + \frac{n}{\mathsf{tr}(\boldsymbol{\Pi}^j)\epsilon^2}\boldsymbol{Z}\boldsymbol{\Pi}^j\boldsymbol{Z}^*\right). \tag{8}$$

When $\boldsymbol{Z}$ is given, $R_c(\boldsymbol{Z}, \epsilon \mid \boldsymbol{\Pi})$ is a concave function of $\boldsymbol{\Pi}$. The function $\log\det(\cdot)$ in the above expressions has been long known as an effective heuristic for rank minimization problems, with guaranteed convergence to local minimum (Fazel et al., 2003). As it nicely characterizes the rate distortion of Gaussian or subspace-like distributions, $\log\det(\cdot)$ can be very effective in clustering or classification of mixed data (Ma et al., 2007; Wright et al., 2008; Kang et al., 2015).

2.2 Principle of Maximal Coding Rate Reduction

On one hand, for learned features to be discriminative, features of different classes/clusters are preferred to be *maximally incoherent* to each other. Hence they together should span a space of the largest possible volume (or dimension) and the coding rate of the whole set $\boldsymbol{Z}$ should be as large as possible. On the other hand, learned features of the same class/cluster should be highly correlated and coherent. Hence each class/cluster should only

14. By a little abuse of notation, we here use $\boldsymbol{Z}$ to denote the set of all samples from all the k classes. For convenience, we often represent $\boldsymbol{Z}$ as a matrix whose columns are the samples.

span a space (or subspace) of a very small volume and the coding rate should be as small as possible. Shortly put, learned features should follow the basic rule that *similarity contracts and dissimilarity contrasts.*

To be more precise, a good (linear) discriminative representation $\boldsymbol{Z}$ of $\boldsymbol{X}$ is one such that, given a partition $\boldsymbol{\Pi}$ of $\boldsymbol{Z}$, achieves a large difference between the coding rate for the whole and that for all the subsets:

$$\Delta R(\boldsymbol{Z}, \boldsymbol{\Pi}, \epsilon) \doteq R(\boldsymbol{Z}, \epsilon) - R_c(\boldsymbol{Z}, \epsilon \mid \boldsymbol{\Pi}). \tag{9}$$

If we choose our feature mapping to be $\boldsymbol{z} = f(\boldsymbol{x}, \boldsymbol{\theta})$ (say modeled by a deep neural network), the overall process of the feature representation and the resulting rate reduction w.r.t. certain partition $\boldsymbol{\Pi}$ can be illustrated by the following diagram:

$$\boldsymbol{X} \xrightarrow{\quad f(\boldsymbol{x}, \boldsymbol{\theta}) \quad} \boldsymbol{Z}(\boldsymbol{\theta}) \xrightarrow{\quad \boldsymbol{\Pi}, \epsilon \quad} \Delta R(\boldsymbol{Z}(\boldsymbol{\theta}), \boldsymbol{\Pi}, \epsilon). \tag{10}$$

The role of normalization. Note that ΔR is *monotonic* in the scale of the features $\boldsymbol{Z}$. So to make the amount of reduction comparable between different representations, we need to *normalize the scale* of the learned features, either by imposing the Frobenius norm of each class $\boldsymbol{Z}^j$ to scale with the number of features in $\boldsymbol{Z}^j \in \mathbb{R}^{n \times m_j}$: $\|\boldsymbol{Z}^j\|_F^2 = m_j$ or by normalizing each feature to be on the unit sphere: $\boldsymbol{z}^i \in \mathbb{S}^{n-1}$. This can be compared to the use of "batch normalization" in the practice of training deep neural networks (Ioffe and Szegedy, 2015).[15] Besides normalizing the scale, normalization could also act as a precondition mechanism that helps accelerate gradient descent (Liu et al., 2021).[16] This interpretation of normalization becomes even more pertinent when we realize deep networks as an iterative scheme to optimize the rate reduction objective in the next two sections. In this work, to simplify the analysis and derivation, we adopt the simplest possible normalization schemes, by simply enforcing each sample on a sphere or the Frobenius norm of each subset being a constant.[17]

Once the representations can be compared fairly, our goal becomes to learn a set of features $\boldsymbol{Z}(\boldsymbol{\theta}) = f(\boldsymbol{X}, \boldsymbol{\theta})$ and their partition $\boldsymbol{\Pi}$ (if not given in advance) such that they maximize the reduction between the coding rate of all features and that of the sum of features w.r.t. their classes:

$$\max_{\boldsymbol{\theta}, \boldsymbol{\Pi}} \Delta R\big(\boldsymbol{Z}(\boldsymbol{\theta}), \boldsymbol{\Pi}, \epsilon\big) = R(\boldsymbol{Z}(\boldsymbol{\theta}), \epsilon) - R_c(\boldsymbol{Z}(\boldsymbol{\theta}), \epsilon \mid \boldsymbol{\Pi}), \quad \text{s.t.} \ \ \|\boldsymbol{Z}^j(\boldsymbol{\theta})\|_F^2 = m_j, \, \boldsymbol{\Pi} \in \Omega. \tag{11}$$

We refer to this as the principle of *maximal coding rate reduction* (MCR^2), an embodiment of Aristotle's famous quote: "*the whole is greater than the sum of the parts.*" Note that for the clustering purpose alone, one may only care about the sign of ΔR for deciding whether to partition the data or not, which leads to the greedy algorithm in (Ma et al., 2007). More

15. Notice that normalizing the scale of the learned representations helps ensure that the mapping of each layer of the network is approximately *isometric*. As it has been shown in the work of Qi et al. (2020), ensuring the isometric property alone is adequate to ensure good performance of deep networks, even without the batch normalization.
16. Similar to the role that preconditioning plays in the classic conjugate gradient descent method (Shewchuk, 1994; Nocedal and Wright, 2006).
17. In practice, to strive for better performance on specific data and tasks, many other normalization schemes can be considered such as layer normalization (Ba et al., 2016), instance normalization (Ulyanov et al., 2016), group normalization (Wu and He, 2018), spectral normalization (Miyato et al., 2018).

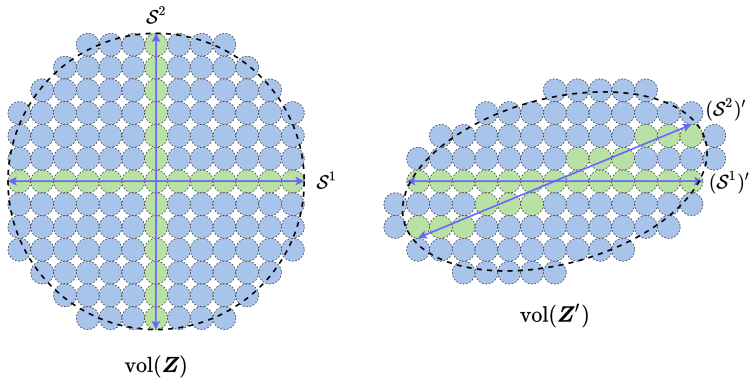


Figure 3: Comparison of two learned representations $\boldsymbol{Z}$ and $\boldsymbol{Z}'$ via reduced rates: R is the number of ϵ-balls packed in the joint distribution and R_c is the sum of the numbers for all the subspaces (the green balls). ΔR is their difference (the number of blue balls). The MCR^2 principle prefers $\boldsymbol{Z}$ (the left one).

specifically, in the context of clustering *finite* samples, one needs to use the more precise measure of the coding length mentioned earlier, see (Ma et al., 2007) for more details. Here to seek or learn the most discriminative representation, we further desire that *the whole is maximally greater than the sum of the parts*. This principle is illustrated with a simple example in Figure 3.

Relationship to information gain. The maximal coding rate reduction can be viewed as a generalization to *information gain* (IG), which aims to maximize the reduction of entropy of a random variable, say $\boldsymbol{z}$, with respect to an observed attribute, say $\boldsymbol{\pi}$: $\max_{\boldsymbol{\pi}} \mathrm{IG}(\boldsymbol{z}, \boldsymbol{\pi}) \doteq H(\boldsymbol{z}) - H(\boldsymbol{z} \mid \boldsymbol{\pi})$, i.e., the *mutual information* between $\boldsymbol{z}$ and $\boldsymbol{\pi}$ (Cover and Thomas, 2006). Maximal information gain has been widely used in areas such as decision trees (Quinlan, 1986). However, MCR^2 is used differently in several ways: 1) One typical setting of MCR^2 is when the data class labels are given, i.e. $\boldsymbol{\Pi}$ is known, MCR^2 focuses on learning representations $\boldsymbol{z}(\boldsymbol{\theta})$ rather than fitting labels. 2) In traditional settings of IG, the number of attributes in $\boldsymbol{z}$ cannot be so large and their values are discrete (typically binary). Here the "attributes" $\boldsymbol{\Pi}$ represent the probability of a multi-class partition for all samples and their values can even be continuous. 3) As mentioned before, entropy $H(\boldsymbol{z})$ or mutual information $I(\boldsymbol{z}, \boldsymbol{\pi})$ (Hjelm et al., 2019) is not well-defined for degenerate continuous distributions or hard to compute for high-dimensional distributions, whereas here the rate distortion $R(\boldsymbol{z}, \epsilon)$ is and can be accurately and efficiently computed for (mixed) subspaces, at least.

2.3 Properties of the Rate Reduction Function

In theory, the MCR^2 principle (11) is very general and can be applied to representations $\boldsymbol{Z}$ of *any* distributions with *any* attributes $\boldsymbol{\Pi}$ as long as the rates R and R_c for the distributions can be accurately and efficiently evaluated. The optimal representation $\boldsymbol{Z}_\star$ and partition $\boldsymbol{\Pi}_\star$ should have some interesting geometric and statistical properties. We here reveal nice properties of the optimal representation with the special case of linear subspaces, which have many important use cases in machine learning. When the desired representation for

$\boldsymbol{Z}$ is multiple subspaces (or Gaussians), the rates R and R_c in (11) are given by (7) and (8), respectively. At any optimal representation, denoted as $\boldsymbol{Z}_\star = \boldsymbol{Z}_\star^1 \cup \cdots \cup \boldsymbol{Z}_\star^k \subset \mathbb{R}^n$, it should achieve the maximal rate reduction. One can show that $\boldsymbol{Z}_\star$ has the following desired properties (see Appendix A for a formal statement and detailed proofs).

Theorem 1 (Informal Statement) *Suppose $\boldsymbol{Z}_\star = \boldsymbol{Z}_\star^1 \cup \cdots \cup \boldsymbol{Z}_\star^k$ is the optimal solution that maximizes the rate reduction* (11) *with the rates R and R_c given by* (7) *and* (8)*. Assume that the optimal solution satisfies $\mathsf{rank}(\boldsymbol{Z}_\star^j) \leq d_j$.*[18] *We have:*

- Between-class Discriminative*: As long as the ambient space is adequately large ($n \geq \sum_{j=1}^k d_j$), the subspaces are all orthogonal to each other,* i.e. *$(\boldsymbol{Z}_\star^i)^* \boldsymbol{Z}_\star^j = \boldsymbol{0}$ for $i \neq j$.*
- Maximally Diverse Representation*: As long as the coding precision is adequately high, i.e., $\epsilon^4 < \min_j \left\{ \frac{m_j}{m} \frac{n^2}{d_j^2} \right\}$, each subspace achieves its maximal dimension, i.e. $\mathsf{rank}(\boldsymbol{Z}_\star^j) = d_j$. In addition, the largest $d_j - 1$ singular values of $\boldsymbol{Z}_\star^j$ are equal.*

In other words, the MCR^2 principle promotes embedding of data into multiple independent subspaces,[19] with features distributed *isotropically* in each subspace (except for possibly one dimension). In addition, among all such discriminative representations, it prefers the one with the highest dimensions in the ambient space (see Section 5.1 and Appendix D for experimental verification). This is substantially different from objectives such as the cross entropy (1) and information bottleneck (3). The optimal representation associated with MCR^2 is indeed an LDR according to definition given in Section 1.3.

Relation to neural collapse. A line of recent work Papyan et al. (2020); Mixon et al. (2020); Han et al. (2021); Fang et al. (2021); Zhu et al. (2021); Zhou et al. (2022); Tirer and Bruna (2022) discovered, both empirically and theoretically, that deep networks trained via cross-entropy or mean squared losses produce *neural collapse* features. That is, features from each class become identical, and different classes are maximally separated from each other. In terms of the coding rate function, the neural collapse solution is preferred by the objective of minimizing the coding rate of the "parts", namely $R_c(\boldsymbol{Z}(\boldsymbol{\theta}), \epsilon \mid \boldsymbol{\Pi})$ (see Table 4 of the Appendix D). However, the neural collapse solution leads to a small coding rate for the "whole", namely $R(\boldsymbol{Z}(\boldsymbol{\theta}))$, hence is not an optimal solution for maximizing the rate reduction. Therefore, the benefit of MCR^2 in preventing the collapsing of the features from each class and producing maximally diverse representations can be attributed to introducing and maximizing the term $R(\boldsymbol{Z}(\boldsymbol{\theta}))$.

Comparison to the geometric OLE loss. To encourage the learned features to be uncorrelated between classes, the work of Lezama et al. (2018) has proposed to maximize the

18. Notice that here we assume we know a good upper bound for the dimension d_j of each class. This requires us to know the intrinsic dimension of the submanifold $\mathcal{M}^j$. In general, this can be a very challenging problem itself even when the submanifold is linear but noisy, which is still an active research topic (Hong et al., 2020). Nevertheless, in practice, we can decide the dimension empirically through ablation experiments, see for example Table 5 for experiments on the CIFAR10 dataset.

19. In this case, the subspaces can be viewed as the *independent components* (Hyvärinen and Oja, 2000) of the so learned features. However, when the condition $n \geq \sum_{j=1}^k d_j$ is violated, the optimal subspaces may not be orthogonal. But experiments show that they tend to be maximally incoherent, see Appendix E.4.

difference between the nuclear norm of the whole $\boldsymbol{Z}$ and its subsets $\boldsymbol{Z}^j$, called the *orthogonal low-rank embedding* (OLE) loss: $\max_{\boldsymbol{\theta}} \text{OLE}(\boldsymbol{Z}(\boldsymbol{\theta}), \boldsymbol{\Pi}) \doteq \|\boldsymbol{Z}(\boldsymbol{\theta})\|_* - \sum_{j=1}^k \|\boldsymbol{Z}^j(\boldsymbol{\theta})\|_*$, added as a regularizer to the cross-entropy loss (1). The nuclear norm $\|\cdot\|_*$ is a *nonsmooth convex* surrogate for low-rankness and the nonsmoothness potentially poses additional difficulties in using this loss to learn features via gradient descent, whereas $\log\det(\cdot)$ is *smooth concave* instead. Unlike the rate reduction ΔR, OLE is always *negative* and achieves the maximal value 0 when the subspaces are orthogonal, regardless of their dimensions. So in contrast to ΔR, this loss serves as a geometric heuristic and does not promote diverse representations. In fact, OLE typically promotes learning one-dimensional representations per class, whereas MCR^2 encourages learning subspaces with maximal dimensions (Figure 7 of Lezama et al. (2018) versus our Figure 18). More importantly, as we will see in the next section, the precise form of the rate distortion plays a crucial role in deriving the deep network operators, with precise statistical and geometrical meaning.

Relation to contractive or contrastive learning. If samples are *evenly* drawn from k classes, a randomly chosen pair $(\boldsymbol{x}^i, \boldsymbol{x}^j)$ is of high probability belonging to different classes if k is large. For example, when $k \geq 100$, a random pair is of probability 99% belonging to different classes. We may view the learned features of two samples together with their augmentations $\boldsymbol{Z}^i$ and $\boldsymbol{Z}^j$ as two classes. Then the rate reduction $\Delta R^{ij} = R(\boldsymbol{Z}^i \cup \boldsymbol{Z}^j, \epsilon) - \frac{1}{2}(R(\boldsymbol{Z}^i, \epsilon) + R(\boldsymbol{Z}^j, \epsilon))$ gives a "distance" measure for how far the two sample sets are. We may try to further "expand" pairs that likely belong to different classes. From Theorem 1, the (averaged) rate reduction ΔR^{ij} is maximized when features from different samples are uncorrelated $(\boldsymbol{Z}^i)^* \boldsymbol{Z}^j = \boldsymbol{0}$ (see Figure 3) and features $\boldsymbol{Z}^i$ from augmentations of the same sample are compressed into the same subspace. Hence, when applied to sample pairs, MCR^2 naturally conducts the so-called *contrastive learning* (Hadsell et al., 2006; Oord et al., 2018; He et al., 2020) and *contractive learning* (Rifai et al., 2011) together that we have discussed in the introduction Section 1.2.1. But MCR^2 is *not* limited to expand or compress pairs of samples and can uniformly conduct "contrastive/contractive learning" for a subset with *any number* of samples as long as we know they likely belong to different (or the same) classes, say by randomly sampling subsets from a large number of classes or with membership derived from a good clustering method.

3. Deep Networks from Maximizing Rate Reduction

In the above section, we have presented rate reduction (11) as a principled objective for learning a linear discriminative representation (LDR) for the data. We have, however, not specified the architecture of the feature mapping $\boldsymbol{z}(\boldsymbol{\theta}) = f(\boldsymbol{x}, \boldsymbol{\theta})$ for extracting such a representation from input data $\boldsymbol{x}$. A straightforward choice is to use a conventional deep network, such as ResNet, for implementing $f(\boldsymbol{x}, \boldsymbol{\theta})$. As we show in the experiments (see Section 5.1), we can effectively optimize the MCR^2 objective with a ResNet architecture and obtain discriminative and diverse representations for real image data sets.

There remain several unanswered problems with using a ResNet. Although the learned feature representation is now more interpretable, the network itself is still *not*. It is unclear why any chosen "black-box" network is able to optimize the desired MCR^2 objective at all. The good empirical results (say with a ResNet) do not necessarily justify the particular choice

in architectures and operators of the network: Why is a deep layered model necessary;[20] what do additional layers try to improve or simplify; how wide and deep is adequate; or is there any rigorous justification for the convolutions (in the popular multi-channel form) and nonlinear operators (e.g. ReLu or softmax) used? In this section, we show that using gradient ascent to maximize the rate reduction $\Delta R(\boldsymbol{Z})$ naturally leads to a "white-box" deep network that represents such a mapping. All network layered architecture, linear/nonlinear operators, and parameters are *explicitly constructed in a purely forward propagation fashion.*

3.1 Gradient Ascent for Rate Reduction on the Training

From the previous section, we see that mathematically, we are essentially seeking a continuous mapping $f(\cdot): \boldsymbol{x} \mapsto \boldsymbol{z}$ from the data $\boldsymbol{X} = [\boldsymbol{x}^1, \ldots, \boldsymbol{x}^m] \in \mathbb{R}^{D \times m}$ (or initial features extracted from the data[21]) to an optimal representation $\boldsymbol{Z} = [\boldsymbol{z}^1, \ldots, \boldsymbol{z}^m] \subset \mathbb{R}^{n \times m}$ that maximizes the following coding rate reduction objective:

$$\begin{aligned}\Delta R(\boldsymbol{Z}, \boldsymbol{\Pi}, \epsilon) &= R(\boldsymbol{Z}, \epsilon) - R_c(\boldsymbol{Z}, \epsilon \mid \boldsymbol{\Pi}) \\ &\doteq \underbrace{\frac{1}{2}\log\det\Big(\boldsymbol{I} + \alpha \boldsymbol{Z}\boldsymbol{Z}^*\Big)}_{R(\boldsymbol{Z},\epsilon)} - \underbrace{\sum_{j=1}^{k} \frac{\gamma_j}{2}\log\det\Big(\boldsymbol{I} + \alpha_j \boldsymbol{Z}\boldsymbol{\Pi}^j\boldsymbol{Z}^*\Big)}_{R_c(\boldsymbol{Z},\epsilon|\boldsymbol{\Pi})},\end{aligned} \tag{12}$$

where for simplicity we denote $\alpha = \frac{n}{m\epsilon^2}$, $\alpha_j = \frac{n}{\mathsf{tr}(\boldsymbol{\Pi}^j)\epsilon^2}$, $\gamma_j = \frac{\mathsf{tr}(\boldsymbol{\Pi}^j)}{m}$ for $j = 1, \ldots, k$.

The question really boils down to whether there is a *constructive* way of finding such a continuous mapping $f(\cdot)$ from $\boldsymbol{x}$ to $\boldsymbol{z}$? To this end, let us consider incrementally maximizing the objective $\Delta R(\boldsymbol{Z})$ as a function of $\boldsymbol{Z} \subset \mathbb{S}^{n-1}$. Although there might be many optimization schemes to choose from, for simplicity we first consider the arguably simplest projected *gradient ascent* (PGA) scheme:[22]

$$\boldsymbol{Z}_{\ell+1} \;\propto\; \boldsymbol{Z}_\ell + \eta \cdot \frac{\partial \Delta R}{\partial \boldsymbol{Z}}\bigg|_{\boldsymbol{Z}_\ell} \quad \text{s.t.} \quad \boldsymbol{Z}_{\ell+1} \subset \mathbb{S}^{n-1},\ \ell = 1, 2, \ldots, \tag{13}$$

for some step size $\eta > 0$ and the iterate starts with the given data $\boldsymbol{Z}_1 = \boldsymbol{X}$[23]. This scheme can be interpreted as how one should incrementally adjust locations of the current features $\boldsymbol{Z}_\ell$, initialized as the input data $\boldsymbol{X}$, in order for the resulting $\boldsymbol{Z}_{\ell+1}$ to improve the rate reduction $\Delta R(\boldsymbol{Z})$, as illustrated in Figure 4.

20. Especially it is already long known that even a single layer neural network is already a universal functional approximator with tractable model complexity (Barron, 1991).
21. As we will see the necessity of such a feature extraction in the next section.
22. Notice that we use superscript j on $\boldsymbol{Z}^j$ to indicate features in the jth class and subscript ℓ on $\boldsymbol{Z}_\ell$ to indicate all features at ℓ-th iteration or layer.
23. Again, for simplicity, we here first assume the initial features $\boldsymbol{Z}_1$ are the data themselves. Hence the data and the features have the same dimension n. This needs not to be the case though. As we will see in the next section, the initial features can be some (lifted) features of the data to begin with and could in principle have a different (much higher) dimension. All subsequent iterates have the same dimension.

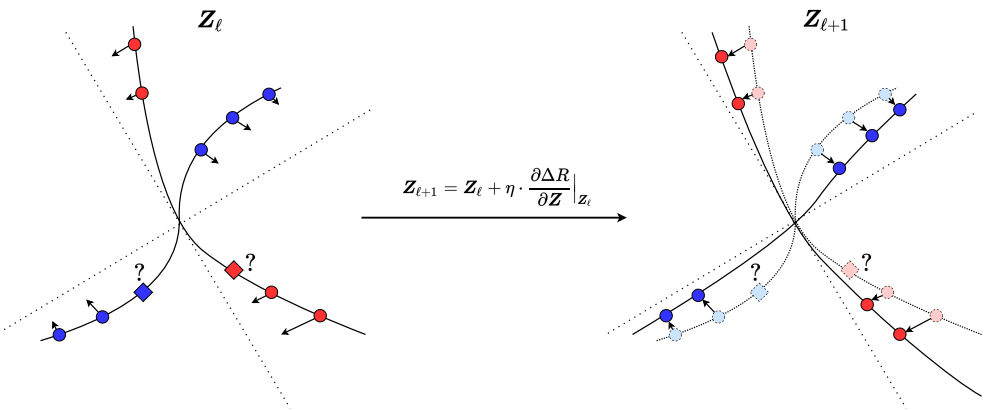


Figure 4: Incremental deformation via gradient flow to both flatten data of each class into a subspace and push different classes apart. Notice that for points whose memberships are unknown, those marked as "$\diamond$", their gradient cannot be directly calculated.

Simple calculation shows that the gradient $\frac{\partial \Delta R}{\partial \boldsymbol{Z}}$ entails evaluating the following derivatives of the two terms in $\Delta R(\boldsymbol{Z})$:

$$\frac{1}{2}\frac{\partial \log\det(\boldsymbol{I}+\alpha \boldsymbol{Z}\boldsymbol{Z}^*)}{\partial \boldsymbol{Z}}\bigg|_{\boldsymbol{Z}_\ell} = \underbrace{\alpha(\boldsymbol{I}+\alpha \boldsymbol{Z}_\ell \boldsymbol{Z}_\ell^*)^{-1}}_{\boldsymbol{E}_\ell \in \mathbb{R}^{n\times n}} \boldsymbol{Z}_\ell, \tag{14}$$

$$\frac{1}{2}\frac{\partial \left(\gamma_j \log\det(\boldsymbol{I}+\alpha_j \boldsymbol{Z}\boldsymbol{\Pi}^j \boldsymbol{Z}^*)\right)}{\partial \boldsymbol{Z}}\bigg|_{\boldsymbol{Z}_\ell} = \gamma_j \underbrace{\alpha_j(\boldsymbol{I}+\alpha_j \boldsymbol{Z}_\ell \boldsymbol{\Pi}^j \boldsymbol{Z}_\ell^*)^{-1}}_{\boldsymbol{C}_\ell^j \in \mathbb{R}^{n\times n}} \boldsymbol{Z}_\ell \boldsymbol{\Pi}^j. \tag{15}$$

Notice that in the above, the matrix $\boldsymbol{E}_\ell$ only depends on $\boldsymbol{Z}_\ell$ and it aims to *expand* all the features to increase the overall coding rate; the matrix $\boldsymbol{C}_\ell^j$ depends on features from each class and aims to *compress* them to reduce the coding rate of each class. Then the complete gradient $\frac{\partial \Delta R}{\partial \boldsymbol{Z}}\big|_{\boldsymbol{Z}_\ell} \in \mathbb{R}^{n\times m}$ is of the form:

$$\frac{\partial \Delta R}{\partial \boldsymbol{Z}}\bigg|_{\boldsymbol{Z}_\ell} = \underbrace{\boldsymbol{E}_\ell}_{\text{Expansion}} \boldsymbol{Z}_\ell - \sum_{j=1}^{k} \gamma_j \underbrace{\boldsymbol{C}_\ell^j}_{\text{Compression}} \boldsymbol{Z}_\ell \boldsymbol{\Pi}^j. \tag{16}$$

Remark 2 (Interpretation of $\boldsymbol{E}_\ell$ and $\boldsymbol{C}_\ell^j$ as Linear Operators) *For any $\boldsymbol{z}_\ell \in \mathbb{R}^n$,*

$$\boldsymbol{E}_\ell \boldsymbol{z}_\ell = \alpha(\boldsymbol{z}_\ell - \boldsymbol{Z}_\ell [\boldsymbol{q}_\ell]_\star) \quad \textit{where} \quad [\boldsymbol{q}_\ell]_\star \doteq \operatorname*{argmin}_{\boldsymbol{q}_\ell} \alpha\|\boldsymbol{z}_\ell - \boldsymbol{Z}_\ell \boldsymbol{q}_\ell\|_2^2 + \|\boldsymbol{q}_\ell\|_2^2. \tag{17}$$

Notice that $[\boldsymbol{q}_\ell]_\star$ is exactly the solution to the ridge regression by all the data points $\boldsymbol{Z}_\ell$ concerned. Therefore, $\boldsymbol{E}_\ell$ (similarly for $\boldsymbol{C}_\ell^j$) is approximately (i.e. when m is large enough) the projection onto the orthogonal complement of the subspace spanned by columns of $\boldsymbol{Z}_\ell$.

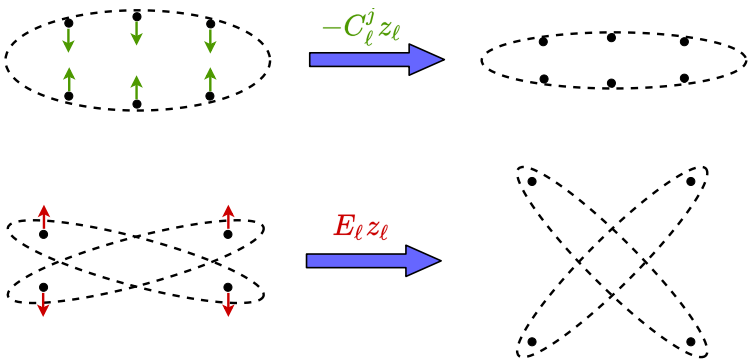


Figure 5: Interpretation of $\boldsymbol{C}_\ell^j$ and $\boldsymbol{E}_\ell$: $\boldsymbol{C}_\ell^j$ compresses each class by contracting the features to a low-dimensional subspace; $\boldsymbol{E}_\ell$ expands all features by contrasting and repelling features across different classes.

Another way to interpret the matrix $\boldsymbol{E}_\ell$ is through eigenvalue decomposition of the covariance matrix $\boldsymbol{Z}_\ell \boldsymbol{Z}_\ell^*$. Assuming that $\boldsymbol{Z}_\ell \boldsymbol{Z}_\ell^* \doteq \boldsymbol{U}_\ell \boldsymbol{\Lambda}_\ell \boldsymbol{U}_\ell^*$ where $\boldsymbol{\Lambda}_\ell \doteq \mathsf{diag}\{\lambda_\ell^1, \ldots, \lambda_\ell^n\}$, we have

$$\boldsymbol{E}_\ell = \alpha\, \boldsymbol{U}_\ell\, \mathsf{diag}\left\{ \frac{1}{1+\alpha\lambda_\ell^1}, \ldots, \frac{1}{1+\alpha\lambda_\ell^n} \right\} \boldsymbol{U}_\ell^*. \tag{18}$$

Therefore, the matrix $\boldsymbol{E}_\ell$ operates on a vector $\boldsymbol{z}_\ell$ by stretching in a way that directions of large variance are shrunk while directions of vanishing variance are kept. These are exactly the directions (14) *in which we move the features so that the overall volume expands and the coding rate will increase, hence the positive sign. To the opposite effect, the directions associated with* (15) *are "residuals" of features of each class deviate from the subspace to which they are supposed to belong. These are exactly the directions in which the features need to be compressed back onto their respective subspace, hence the negative sign (see Figure 5).*

Essentially, the linear operations $\boldsymbol{E}_\ell$ and $\boldsymbol{C}_\ell^j$ in gradient ascend for rate reduction are determined by training data conducting "auto-regressions". The recent renewed understanding about ridge regression in an over-parameterized setting (Yang et al., 2020; Wu and Xu, 2020) indicates that using seemingly redundantly sampled data (from each subspaces) as regressors do not lead to overfitting.

3.2 Gradient-Guided Feature Map Increment

Notice that in the above, the gradient ascent considers all the features $\boldsymbol{Z}_\ell = [\boldsymbol{z}_\ell^1, \ldots, \boldsymbol{z}_\ell^m]$ as free variables. The increment $\boldsymbol{Z}_{\ell+1} - \boldsymbol{Z}_\ell = \eta \frac{\partial \Delta R}{\partial \boldsymbol{Z}}\big|_{\boldsymbol{Z}_\ell}$ does not yet give a transform on the entire feature domain $\boldsymbol{z}_\ell \in \mathbb{R}^n$. According to equation (16), the gradient cannot be evaluated at a point whose membership is not known, as illustrated in Figure 4. Hence, in order to find the optimal $f(\boldsymbol{x}, \boldsymbol{\theta})$ explicitly, we may consider constructing a small increment transform $g(\cdot, \boldsymbol{\theta}_\ell)$ on the ℓ-th layer feature $\boldsymbol{z}_\ell$ to emulate the above (projected) gradient scheme:

$$\boldsymbol{z}_{\ell+1} \;\propto\; \boldsymbol{z}_\ell + \eta \cdot g(\boldsymbol{z}_\ell, \boldsymbol{\theta}_\ell) \quad \text{subject to} \quad \boldsymbol{z}_{\ell+1} \in \mathbb{S}^{n-1} \tag{19}$$

such that: $\big[g(\boldsymbol{z}_\ell^1, \boldsymbol{\theta}_\ell), \ldots, g(\boldsymbol{z}_\ell^m, \boldsymbol{\theta}_\ell)\big] \approx \frac{\partial \Delta R}{\partial \boldsymbol{Z}}\big|_{\boldsymbol{Z}_\ell}$. That is, we need to approximate the gradient flow $\frac{\partial \Delta R}{\partial \boldsymbol{Z}}$ that locally deforms all (training) features $\{\boldsymbol{z}_\ell^i\}_{i=1}^m$ with a continuous mapping $g(\boldsymbol{z})$

defined on the entire feature space $\boldsymbol{z}_\ell \in \mathbb{R}^n$. Notice that one may interpret the increment (19) as a discretized version of a continuous differential equation:

$$\dot{\boldsymbol{z}} = g(\boldsymbol{z}, \theta). \tag{20}$$

Hence the (deep) network so constructed can be interpreted as certain neural ODE (Chen et al., 2018a). Nevertheless, unlike neural ODE where the flow g is chosen to be some generic structures, here our $g(\boldsymbol{z}, \theta)$ is to emulate the gradient flow of the rate reduction on the feature set (as shown in Figure 4):

$$\dot{\boldsymbol{Z}} = \frac{\partial \Delta R}{\partial \boldsymbol{Z}}, \tag{21}$$

and its structure is entirely derived and fully determined from this objective, without any other priors or heuristics.

By inspecting the structure of the gradient (16), it suggests that a natural candidate for the increment transform $g(\boldsymbol{z}_\ell, \boldsymbol{\theta}_\ell)$ is of the form:

$$g(\boldsymbol{z}_\ell, \boldsymbol{\theta}_\ell) \;\doteq\; \boldsymbol{E}_\ell \boldsymbol{z}_\ell - \sum_{j=1}^{k} \gamma_j \boldsymbol{C}_\ell^j \boldsymbol{z}_\ell \boldsymbol{\pi}^j(\boldsymbol{z}_\ell) \quad \in \mathbb{R}^n, \tag{22}$$

where $\boldsymbol{\pi}^j(\boldsymbol{z}_\ell) \in [0, 1]$ indicates the probability of $\boldsymbol{z}_\ell$ belonging to the j-th class. The increment depends on: First, a set of linear maps represented by $\boldsymbol{E}_\ell$ and $\{\boldsymbol{C}_\ell^j\}_{j=1}^k$ that depend only on statistics of features of the training $\boldsymbol{Z}_\ell$; Second, membership $\{\boldsymbol{\pi}^j(\boldsymbol{z}_\ell)\}_{j=1}^k$ of any feature $\boldsymbol{z}_\ell$. Notice that on the training samples $\boldsymbol{Z}_\ell$, for which the memberships $\boldsymbol{\Pi}^j$ are known, the so defined $g(\boldsymbol{z}_\ell, \boldsymbol{\theta})$ gives exactly the values for the gradient $\frac{\partial \Delta R}{\partial \boldsymbol{Z}}\big|_{\boldsymbol{Z}_\ell}$.

Since we only have the membership $\boldsymbol{\pi}^j$ for the training samples, the function $g(\cdot)$ defined in (22) can only be evaluated on the training. To extrapolate $g(\cdot)$ to the entire feature space, we need to estimate $\boldsymbol{\pi}^j(\boldsymbol{z}_\ell)$ in its second term. In the conventional deep learning, this map is typically modeled as a deep network and learned from the training data, say via *back propagation*. Nevertheless, our goal here is not to learn a precise classifier $\boldsymbol{\pi}^j(\boldsymbol{z}_\ell)$ already. Instead, we only need a good enough estimate of the class information in order for $g(\cdot)$ to approximate the gradient $\frac{\partial \Delta R}{\partial \boldsymbol{Z}}$ well.

From the geometric interpretation of the linear maps $\boldsymbol{E}_\ell$ and $\boldsymbol{C}_\ell^j$ given by Remark 2, the term $\boldsymbol{p}_\ell^j \doteq \boldsymbol{C}_\ell^j \boldsymbol{z}_\ell$ can be viewed as (approximately) the projection of $\boldsymbol{z}_\ell$ onto the orthogonal complement of each class j. Therefore, $\|\boldsymbol{p}_\ell^j\|_2$ is small if $\boldsymbol{z}_\ell$ is in class j and large otherwise. This motivates us to estimate its membership based on the following softmax function:

$$\widehat{\boldsymbol{\pi}}^j(\boldsymbol{z}_\ell) \doteq \frac{\exp\left(-\lambda \|\boldsymbol{C}_\ell^j \boldsymbol{z}_\ell\|\right)}{\sum_{j=1}^k \exp\left(-\lambda \|\boldsymbol{C}_\ell^j \boldsymbol{z}_\ell\|\right)} \in [0, 1]. \tag{23}$$

Hence the second term of (22) can be approximated by this estimated membership:

$$\sum_{j=1}^{k} \gamma_j \boldsymbol{C}_\ell^j \boldsymbol{z}_\ell \boldsymbol{\pi}^j(\boldsymbol{z}_\ell) \;\approx\; \sum_{j=1}^{k} \gamma_j \boldsymbol{C}_\ell^j \boldsymbol{z}_\ell \cdot \widehat{\boldsymbol{\pi}}^j(\boldsymbol{z}_\ell) \;\doteq\; \boldsymbol{\sigma}\Big([\boldsymbol{C}_\ell^1 \boldsymbol{z}_\ell, \ldots, \boldsymbol{C}_\ell^k \boldsymbol{z}_\ell]\Big), \tag{24}$$

which is denoted as a nonlinear operator $\boldsymbol{\sigma}(\cdot)$ on outputs of the feature $\boldsymbol{z}_\ell$ through k groups of filters: $[\boldsymbol{C}_\ell^1, \ldots, \boldsymbol{C}_\ell^k]$. Notice that the nonlinearity arises due to a "soft" assignment of class membership based on the feature responses from those filters.

Remark 3 (Approximate Membership with a ReLU Network) *The choice of the softmax is mostly for its simplicity as it is widely used in other (forward components of) deep networks for purposes such as clustering, gating (Shazeer et al., 2017) and routing (Sabour et al., 2017). In practice, there are many other simpler nonlinear activation functions that one can use to approximate the membership $\widehat{\boldsymbol{\pi}}(\cdot)$ and subsequently the nonlinear operation $\boldsymbol{\sigma}$ in* (24)*. Notice that the geometric meaning of $\boldsymbol{\sigma}$ in* (24) *is to compute the "residual" of each feature against the subspace to which it belongs. There are many different ways one may approximate this quantity. For example, when we restrict all our features to be in the first (positive) quadrant of the feature space,*[24] *one may approximate this residual using the rectified linear units operation, ReLUs, on $\boldsymbol{p}_j = \boldsymbol{C}_\ell^j \boldsymbol{z}_\ell$ or its orthogonal complement:*

$$\boldsymbol{\sigma}(\boldsymbol{z}_\ell) \;\propto\; \boldsymbol{z}_\ell - \sum_{j=1}^{k} ReLU\big(\boldsymbol{P}_\ell^j \boldsymbol{z}_\ell\big), \tag{25}$$

where $\boldsymbol{P}_\ell^j = (\boldsymbol{C}_\ell^j)^\perp$ is (approximately) the projection onto the j-th class and $ReLU(x) = \max(0, x)$. The above approximation is good under the more restrictive assumption that projection of $\boldsymbol{z}_\ell$ on the correct class via $\boldsymbol{P}_\ell^j$ is mostly large and positive and yet small or negative for other classes.

Overall, combining (19), (22), and (24), the increment feature transform from $\boldsymbol{z}_\ell$ to $\boldsymbol{z}_{\ell+1}$ now becomes:

$$\begin{aligned} \boldsymbol{z}_{\ell+1} &\propto \boldsymbol{z}_\ell + \eta \cdot \boldsymbol{E}_\ell \boldsymbol{z}_\ell - \eta \cdot \boldsymbol{\sigma}\big([\boldsymbol{C}_\ell^1 \boldsymbol{z}_\ell, \ldots, \boldsymbol{C}_\ell^k \boldsymbol{z}_\ell]\big) \\ &= \boldsymbol{z}_\ell + \eta \cdot g(\boldsymbol{z}_\ell, \boldsymbol{\theta}_\ell) \quad \text{s.t.} \quad \boldsymbol{z}_{\ell+1} \in \mathbb{S}^{n-1}, \end{aligned} \tag{26}$$

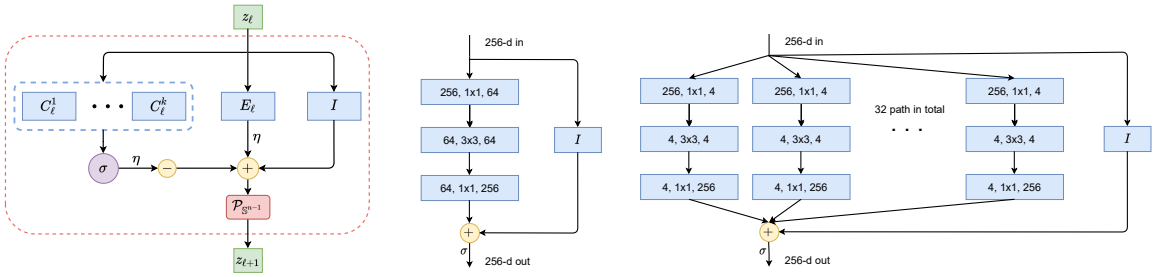
with the nonlinear function $\boldsymbol{\sigma}(\cdot)$ defined above and $\boldsymbol{\theta}_\ell$ collecting all the layer-wise parameters. That is $\boldsymbol{\theta}_\ell = \big\{\boldsymbol{E}_\ell, \boldsymbol{C}_\ell^1, \ldots, \boldsymbol{C}_\ell^k, \gamma_j, \lambda\big\}$. Note features at each layer are always "normalized" by projecting onto the unit sphere $\mathbb{S}^{n-1}$, denoted as $\mathcal{P}_{\mathbb{S}^{n-1}}$. The form of increment in (26) can be illustrated by a diagram in Figure 6 left.

3.3 Deep Network for Optimizing Rate Reduction

Notice that the increment is constructed to emulate the gradient ascent for the rate reduction ΔR. Hence by transforming the features iteratively via the above process, we expect the rate reduction to increase, as we will see in the experimental section. This iterative process, once converged say after L iterations, gives the desired feature map $f(\boldsymbol{x}, \boldsymbol{\theta})$ on the input $\boldsymbol{z}_1 = \boldsymbol{x}$, precisely in the form of a *deep network*, in which each layer has the structure shown in Figure 6 left:

$$\begin{aligned} f(\boldsymbol{x}, \boldsymbol{\theta}) &= \phi_L \circ \phi_{L-1} \circ \cdots \circ \phi_2 \circ \phi_1(\boldsymbol{z}_1), \\ \phi_\ell(\boldsymbol{z}_\ell, \boldsymbol{\theta}_\ell) &\doteq \boldsymbol{z}_{\ell+1} = \mathcal{P}_{\mathbb{S}^{n-1}}[\boldsymbol{z}_\ell + \eta \cdot g(\boldsymbol{z}_\ell, \boldsymbol{\theta}_\ell)], \\ g(\boldsymbol{z}_\ell, \boldsymbol{\theta}_\ell) &= \boldsymbol{E}_\ell \boldsymbol{z}_\ell - \boldsymbol{\sigma}\big([\boldsymbol{C}_\ell^1 \boldsymbol{z}_\ell, \ldots, \boldsymbol{C}_\ell^k \boldsymbol{z}_\ell]\big). \end{aligned} \tag{27}$$

24. Most current neural networks seem to adopt this regime.



(a) **ReduNet** (b) **ResNet** and **ResNeXt**.

Figure 6: Network Architectures of the ReduNet and comparison with others. **(a)**: Layer structure of the **ReduNet** derived from one iteration of gradient ascent for optimizing rate reduction. **(b)** (left): A layer of ResNet (He et al., 2016); and **(b)** (right): A layer of ResNeXt (Xie et al., 2017). As we will see in Section 4, the linear operators $\boldsymbol{E}_\ell$ and $\boldsymbol{C}_\ell^j$ of the ReduNet naturally become (multi-channel) convolutions when shift-invariance is imposed.

Algorithm 1 Training Algorithm for ReduNet

Input: $\boldsymbol{X} = [\boldsymbol{x}^1, \ldots, \boldsymbol{x}^m] \in \mathbb{R}^{D\times m}$, $\boldsymbol{\Pi}$, $\epsilon > 0$, feature dimension n, λ, and a learning rate η.

Set $\alpha = n/(m\epsilon^2)$, $\{\alpha_j = n/(\mathsf{tr}\left(\boldsymbol{\Pi}^j\right)\epsilon^2)\}_{j=1}^k$, $\{\gamma_j = \mathsf{tr}\left(\boldsymbol{\Pi}^j\right)/m\}_{j=1}^k$.
Set $\boldsymbol{Z}_1 \doteq [\boldsymbol{z}_1^1, \ldots, \boldsymbol{z}_1^m] = \boldsymbol{X} \in \mathbb{R}^{n\times m}$ (assuming $n = D$ for simplicity).
for $\ell = 1, 2, \ldots, L$ **do**
  `# Step 1:Compute network parameters` $\boldsymbol{E}_\ell$ `and` $\{\boldsymbol{C}_\ell^j\}_{j=1}^k$.
  $\boldsymbol{E}_\ell \doteq \alpha(\boldsymbol{I} + \alpha \boldsymbol{Z}_\ell \boldsymbol{Z}_\ell^*)^{-1} \in \mathbb{R}^{n\times n}$, $\{\boldsymbol{C}_\ell^j \doteq \alpha_j(\boldsymbol{I} + \alpha_j \boldsymbol{Z}_\ell \boldsymbol{\Pi}^j \boldsymbol{Z}_\ell^*)^{-1} \in \mathbb{R}^{n\times n}\}_{j=1}^k$;
  `# Step 2:Update feature` $\boldsymbol{Z}_\ell$.
  for $i = 1, \ldots, m$ **do**
    `# Compute soft assignment` $\{\widehat{\boldsymbol{\pi}}^j(\boldsymbol{z}_\ell^i)\}_{j=1}^k$.
    $\left\{\widehat{\boldsymbol{\pi}}^j(\boldsymbol{z}_\ell^i) \doteq \frac{\exp\left(-\lambda\|\boldsymbol{C}_\ell^j \boldsymbol{z}_\ell^i\|\right)}{\sum_{j=1}^k \exp\left(-\lambda\|\boldsymbol{C}_\ell^j \boldsymbol{z}_\ell^i\|\right)} \in [0,1]\right\}_{j=1}^k$;
    `# Update feature` $\boldsymbol{z}_\ell^i$.
    $\boldsymbol{z}_{\ell+1}^i = \mathcal{P}_{\mathbb{S}^{n-1}}\left(\boldsymbol{z}_\ell^i + \eta\left(\boldsymbol{E}_\ell \boldsymbol{z}_\ell^i - \sum_{j=1}^k \gamma_j \boldsymbol{C}_\ell^j \boldsymbol{z}_\ell^i \cdot \widehat{\boldsymbol{\pi}}^j(\boldsymbol{z}_\ell^i)\right)\right) \in \mathbb{R}^n$;
  end for
end for

Output: features $\boldsymbol{Z}_{L+1}$, the learned parameters $\{\boldsymbol{E}_\ell\}_{\ell=1}^L$ and $\{\boldsymbol{C}_\ell^j\}_{j=1,\ell=1}^{k,L}$, $\{\gamma_j\}_{j=1}^k$.

As this deep network is derived from maximizing the rate **redu**ced, we call it the **ReduNet**. We summarize the training and evaluation of ReduNet in Algorithm 1 and Algorithm 2, respectively. Notice that all parameters of the network are explicitly constructed layer by layer in a *forward propagation* fashion. The construction does not need any back propagation! The so learned features can be directly used for classification, say via a nearest subspace classifier.

Algorithm 2 Evaluation Algorithm for ReduNet

Input: $\boldsymbol{x} \in \mathbb{R}^D$, network parameters $\{\boldsymbol{E}_\ell\}_{\ell=1}^L$ and $\{\boldsymbol{C}_\ell^j\}_{j=1,\ell=1}^{k,L}$, $\{\gamma_j\}_{j=1}^k$, feature dimension n, λ, and a learning rate η.

1: Set $\boldsymbol{z}_1 = \boldsymbol{x} \in \mathbb{R}^n$ (assuming $n = D$ for simplicity).
2: **for** $\ell = 1, 2, \ldots, L$ **do**
3: `# Compute soft assignment` $\{\widehat{\boldsymbol{\pi}}^j(\boldsymbol{z}_\ell)\}_{j=1}^k$`.`
4: $\left\{\widehat{\boldsymbol{\pi}}^j(\boldsymbol{z}_\ell) \doteq \frac{\exp\left(-\lambda\|\boldsymbol{C}_\ell^j \boldsymbol{z}_\ell\|\right)}{\sum_{j=1}^k \exp\left(-\lambda\|\boldsymbol{C}_\ell^j \boldsymbol{z}_\ell\|\right)} \in [0,1]\right\}_{j=1}^k$;
5: `# Update feature` $\boldsymbol{z}_\ell$`.`
6: $\boldsymbol{z}_{\ell+1} = \mathcal{P}_{\mathbb{S}^{n-1}}\left(\boldsymbol{z}_\ell + \eta\left(\boldsymbol{E}_\ell \boldsymbol{z}_\ell - \sum_{j=1}^k \gamma_j \boldsymbol{C}_\ell^j \boldsymbol{z}_\ell \cdot \widehat{\boldsymbol{\pi}}^j(\boldsymbol{z}_\ell)\right)\right) \quad \in \mathbb{R}^n$;
7: **end for**

Output: feature $\boldsymbol{z}_{L+1}$

3.4 Comparison with Other Approaches and Architectures

Like all networks that are inspired by unfolding certain iterative optimization schemes, the structure of the ReduNet naturally contains a skip connection between adjacent layers as in the ResNet (He et al., 2016) (see Figure 6 middle). Empirically, people have found that additional skip connections across multiple layers may improve the network performance, e.g. the Highway networks (Srivastava et al., 2015) and DenseNet (Huang et al., 2017). In our framework, the role of each layer is precisely interpreted as one iterative gradient ascent step for the objective function ΔR. In our experiments (see Section 5), we have observed that the basic gradient scheme sometimes converges slowly, resulting in deep networks with hundreds of layers (iterations)! To improve the efficiency of the basic ReduNet, one may consider in the future accelerated gradient methods such as the Nesterov acceleration (Nesterov, 1983) or perturbed accelerated gradient descent (Jin et al., 2018). Say to minimize or maximize a function $h(\boldsymbol{z})$, such accelerated methods usually take the form:

$$\begin{cases} \boldsymbol{q}_{\ell+1} &= \boldsymbol{z}_\ell + \beta_\ell \cdot (\boldsymbol{z}_\ell - \boldsymbol{z}_{\ell-1}), \\ \boldsymbol{z}_{\ell+1} &= \boldsymbol{q}_{\ell+1} + \eta \cdot \nabla h(\boldsymbol{q}_{\ell+1}). \end{cases} \tag{28}$$

They require introducing additional skip connections among three layers $\ell - 1$, ℓ and $\ell + 1$. For typical convex or nonconvex programs, the above accelerated schemes can often reduce the number of iterations by a magnitude (Wright and Ma, 2021).

Notice that, structure wise, the $k + 1$ parallel groups of channels $\boldsymbol{E}, \boldsymbol{C}^j$ of the ReduNet correspond to the "residual" channel of the ResNet (Figure 6 middle). Remarkably, here in ReduNet, we know they precisely correspond to *the residual of data auto-regression* (see Remark 2). Moreover, the multiple parallel groups actually draw resemblance to the parallel structures that people later empirically found to further improve the ResNet, e.g. ResNeXt (Xie et al., 2017) (shown in Figure 6 right) or the mixture of experts (MoE) module adopted in Shazeer et al. (2017).[25] Now in ReduNet, each of those groups $\boldsymbol{C}^j$ can be precisely interpreted as *an expert classifier for each class of objects*. But a major difference here is

25. The latest large language model, the switched transformer (Fedus et al., 2021), adopts the MoE architecture, in which the number of parallel banks (or experts) k are in the thousands and the total number of parameters of the network is about 1.7 trillion.

that all above networks need to be initialized randomly and trained via back propagation whereas all components (layers, operators, and parameters) of the ReduNet are by explicit construction in a forward propagation. They all have precise optimization, statistical and geometric interpretation.

Of course, like any other deep networks, the so-constructed ReduNet is amenable to fine-tuning via back-propagation if needed. A recent study from Giryes et al. (2018) has shown that such fine-tuning may achieve a better trade off between accuracy and efficiency of the unrolled network (say when only a limited number of layers, or iterations are allowed in practice). Nevertheless, for the ReduNet, one can start with the nominal values obtained from the forward construction, instead of random initialization. Benefits of fine-tuning and initialization will be verified in the experimental section (see Table 2).

4. Deep Convolution Networks from Invariant Rate Reduction

So far, we have considered the data $\boldsymbol{x}$ and their features $\boldsymbol{z}$ as vectors. In many applications, such as serial data or imagery data, the semantic meaning (labels) of the data are *invariant* to certain transformations $\mathfrak{g} \in \mathbb{G}$ (for some group $\mathbb{G}$) (Cohen and Welling, 2016; Zaheer et al., 2017). For example, the meaning of an audio signal is invariant to shift in time; and the identity of an object in an image is invariant to translation in the image plane. Hence, we prefer the feature mapping $f(\boldsymbol{x}, \boldsymbol{\theta})$ is rigorously invariant to such transformations:

$$\textit{Group Invariance: } f(\boldsymbol{x} \circ \mathfrak{g}, \boldsymbol{\theta}) \sim f(\boldsymbol{x}, \boldsymbol{\theta}), \quad \forall \mathfrak{g} \in \mathbb{G}, \tag{29}$$

where "$\sim$" indicates two features belonging to the same equivalent class. Although to ensure invariance or equivarience, convolutional operators has been common practice in deep networks (Cohen and Welling, 2016), it remains challenging in practice to train an (empirically designed) convolution network from scratch that can *guarantee* invariance even to simple transformations such as translation and rotation (Azulay and Weiss, 2019; Engstrom et al., 2019). An alternative approach is to carefully design convolution filters of each layer so as to ensure translational invariance for a wide range of signals, say using wavelets as in ScatteringNet (Bruna and Mallat, 2013) and followup works (Wiatowski and Bölcskei, 2018). However, in order to ensure invariance to generic signals, the number of convolutions needed usually grows exponentially with network depth. That is the reason why this type of network cannot be constructed so deep, usually only several layers.

In this section, we show that the MCR^2 principle is compatible with invariance in a very natural and precise way: we only need to assign all transformed versions $\{\boldsymbol{x} \circ \mathfrak{g} \mid \mathfrak{g} \in \mathbb{G}\}$ into the same class as the data $\boldsymbol{x}$ and map their features $\boldsymbol{z}$ all to the same subspace $\mathcal{S}$. Hence, all group equivariant information is encoded only inside the subspace, and any classifier defined on the resulting set of subspaces will be automatically invariant to such group transformations. See Figure 7 for an illustration of the examples of 1D rotation and 2D translation. We will rigorously show in the next two sections (as well as Appendix B and Appendix C) that, when the group $\mathbb{G}$ is circular 1D shifting or 2D translation, the resulting deep network naturally becomes a *multi-channel convolution network*! Because the so-constructed network only needs to ensure invariance for the given data $\boldsymbol{X}$ or their features $\boldsymbol{Z}$, the number of convolutions needed actually remain constant through a very deep network, as oppose to the ScatteringNet.

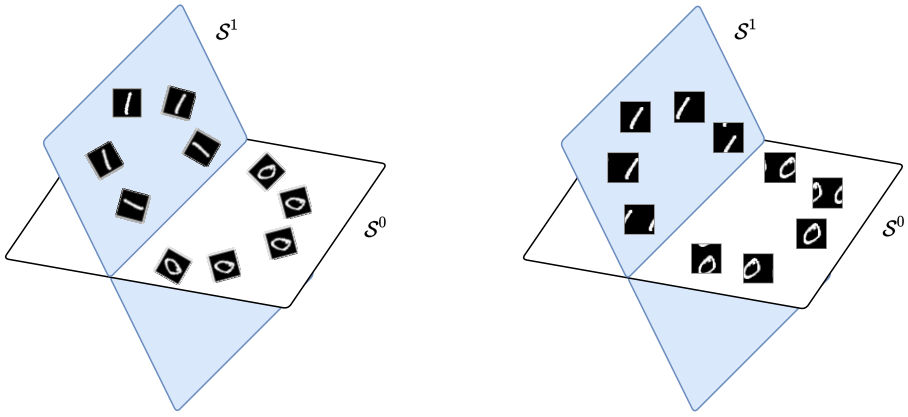


Figure 7: Illustration of the sought representation that is equivariant/invariant to image rotation (left) or translation (right): all transformed images of each class are mapped into the same subspace that are incoherent to other subspaces. The features embedded in each subspace are equivariant to transformation group whereas each subspace is invariant to such transformations.

4.1 1D Serial Data and Shift Invariance

To classify one-dimensional data $\boldsymbol{x} = [x(0), x(1), \ldots, x(D-1)] \in \mathbb{R}^D$ invariant under shifting, we take $\mathbb{G}$ to be the group of all circular shifts. Each observation $\boldsymbol{x}^i$ generates a family $\{\boldsymbol{x}^i \circ \mathfrak{g} \,|\, \mathfrak{g} \in \mathbb{G}\}$ of shifted copies, which are the columns of the circulant matrix $\mathsf{circ}(\boldsymbol{x}^i) \in \mathbb{R}^{D\times D}$ given by

$$\mathsf{circ}(\boldsymbol{x}) \doteq \begin{bmatrix} x(0) & x(D-1) & \ldots & x(2) & x(1) \\ x(1) & x(0) & x(D-1) & \cdots & x(2) \\ \vdots & x(1) & x(0) & \ddots & \vdots \\ x(D-2) & \vdots & \ddots & \ddots & x(D-1) \\ x(D-1) & x(D-2) & \ldots & x(1) & x(0) \end{bmatrix} \in \mathbb{R}^{D\times D}. \tag{30}$$

We refer the reader to Appendix B.1 or Kra and Simanca (2012) for properties of circulant matrices. For simplicity, let $\boldsymbol{Z}_1 \doteq [\boldsymbol{z}_1^1, \ldots, \boldsymbol{z}_1^m] = \boldsymbol{X} \in \mathbb{R}^{n\times m}$[26]. Then what happens if we construct the ReduNet from their circulant families $\mathsf{circ}(\boldsymbol{Z}_1) = \big[\mathsf{circ}(\boldsymbol{z}_1^1), \ldots, \mathsf{circ}(\boldsymbol{z}_1^m)\big] \in \mathbb{R}^{n\times nm}$? That is, we want to compress and map all these into the same subspace by the ReduNet.

Notice that now the data covariance matrix:

$$\begin{aligned} \mathsf{circ}(\boldsymbol{Z}_1)\mathsf{circ}(\boldsymbol{Z}_1)^* &= \big[\mathsf{circ}(\boldsymbol{z}^1), \ldots, \mathsf{circ}(\boldsymbol{z}^m)\big]\big[\mathsf{circ}(\boldsymbol{z}^1), \ldots, \mathsf{circ}(\boldsymbol{z}^m)\big]^* & (31) \\ &= \sum_{i=1}^m \mathsf{circ}(\boldsymbol{z}_1^i)\mathsf{circ}(\boldsymbol{z}_1^i)^* \in \mathbb{R}^{n\times n} & (32) \end{aligned}$$

26. Again, to simplify discussion, we assume for now that the initial features $\boldsymbol{Z}_1$ are $\boldsymbol{X}$ themselves hence have the same dimension n. But that does not need to be the case as we will soon see that we need to lift $\boldsymbol{X}$ to a higher dimension.

associated with this family of samples is *automatically* a (symmetric) circulant matrix. Moreover, because the circulant property is preserved under sums, inverses, and products, the matrices $\boldsymbol{E}_1$ and $\boldsymbol{C}_1^j$ are also automatically circulant matrices, whose application to a feature vector $\boldsymbol{z} \in \mathbb{R}^n$ can be implemented using circular convolution "$\circledast$". Specifically, we have the following proposition.

Proposition 4 (Convolution structures of $\boldsymbol{E}_1$ and $\boldsymbol{C}_1^j$) *The matrix*

$$\boldsymbol{E}_1 = \alpha\big(\boldsymbol{I} + \alpha\mathsf{circ}(\boldsymbol{Z}_1)\mathsf{circ}(\boldsymbol{Z}_1)^*\big)^{-1} \tag{33}$$

is a circulant matrix and represents a circular convolution:

$$\boldsymbol{E}_1\boldsymbol{z} = \boldsymbol{e}_1 \circledast \boldsymbol{z},$$

where $\boldsymbol{e}_1 \in \mathbb{R}^n$ is the first column vector of $\boldsymbol{E}_1$ and "$\circledast$" is circular convolution defined as

$$(\boldsymbol{e}_1 \circledast \boldsymbol{z})_i \doteq \sum_{j=0}^{n-1} e_1(j)x(i+n-j \ \mathit{mod}\ n).$$

Similarly, the matrices $\boldsymbol{C}_1^j$ associated with any subsets of $\boldsymbol{Z}_1$ are also circular convolutions.

Not only the first-layer parameters $\boldsymbol{E}_1$ and $\boldsymbol{C}_1^j$ of the ReduNet become circulant convolutions but also the next layer features remain circulant matrices. That is, the incremental feature transform in (26) applied to all shifted versions of a $\boldsymbol{z}_1 \in \mathbb{R}^n$, given by

$$\mathsf{circ}(\boldsymbol{z}_1) + \eta \cdot \boldsymbol{E}_1\mathsf{circ}(\boldsymbol{z}_1) - \eta \cdot \boldsymbol{\sigma}\Big([\boldsymbol{C}_1^1\mathsf{circ}(\boldsymbol{z}_1), \ldots, \boldsymbol{C}_1^k\mathsf{circ}(\boldsymbol{z}_1)]\Big), \tag{34}$$

is a circulant matrix. This implies that there is no need to construct circulant families from the second layer features as we did for the first layer. By denoting

$$\boldsymbol{z}_2 \propto \boldsymbol{z}_1 + \eta \cdot g(\boldsymbol{z}_1, \boldsymbol{\theta}_1) = \boldsymbol{z}_1 + \eta \cdot \boldsymbol{e}_1 \circledast \boldsymbol{z}_1 - \eta \cdot \boldsymbol{\sigma}\Big([\boldsymbol{c}_1^1 \circledast \boldsymbol{z}_1, \ldots, \boldsymbol{c}_1^k \circledast \boldsymbol{z}_1]\Big), \tag{35}$$

the features at the next level can be written as

$$\mathsf{circ}(\boldsymbol{Z}_2) = \big[\mathsf{circ}(\boldsymbol{z}_2^1), \ldots, \mathsf{circ}(\boldsymbol{z}_2^m)\big] = \big[\mathsf{circ}(\boldsymbol{z}_1^1 + \eta g(\boldsymbol{z}_1^1, \boldsymbol{\theta}_1)), \ldots, \mathsf{circ}(\boldsymbol{z}_1^m + \eta g(\boldsymbol{z}_1^m, \boldsymbol{\theta}_1))\big].$$

Continuing inductively, we see that all matrices $\boldsymbol{E}_\ell$ and $\boldsymbol{C}_\ell^j$ based on such $\mathsf{circ}(\boldsymbol{Z}_\ell)$ are circulant, and so are all features. By virtue of the properties of the data, ReduNet has taken the form of a convolutional network, *with no need to explicitly choose this structure!*

4.2 A Fundamental Trade-off between Invariance and Sparsity

There is one problem though: In general, the set of all circular permutations of a vector $\boldsymbol{z}$ gives a full-rank matrix. That is, the n "augmented" features associated with each sample (hence each class) typically already span the entire space $\mathbb{R}^n$. For instance, all shifted versions of a delta function $\delta(n)$ can generate any other signal as their (dense) weighted superposition. The MCR^2 objective (11) will not be able to distinguish classes as different subspaces.

One natural remedy is to improve the separability of the data by "lifting" the original signal to a higher dimensional space, e.g., by taking their responses to multiple, filters $\boldsymbol{k}_1, \ldots, \boldsymbol{k}_C \in \mathbb{R}^n$:

$$\boldsymbol{z}[c] = \boldsymbol{k}_c \circledast \boldsymbol{x} = \mathsf{circ}(\boldsymbol{k}_c)\boldsymbol{x} \quad \in \mathbb{R}^n, \quad c = 1, \ldots, C. \tag{36}$$

The filters can be pre-designed invariance-promoting filters,[27] or adaptively learned from the data,[28] or randomly selected as we do in our experiments. This operation lifts each original signal $\boldsymbol{x} \in \mathbb{R}^n$ to a C-channel feature, denoted as $\bar{\boldsymbol{z}} \doteq [\boldsymbol{z}[1], \ldots, \boldsymbol{z}[C]]^* \in \mathbb{R}^{C \times n}$. Then, we may construct the ReduNet on vector representations of $\bar{\boldsymbol{z}}$, denoted as $\mathsf{vec}(\bar{\boldsymbol{z}}) \doteq [\boldsymbol{z}[1]^*, \ldots, \boldsymbol{z}[C]^*] \in \mathbb{R}^{nC}$. The associated circulant version $\mathsf{circ}(\bar{\boldsymbol{z}})$ and its data covariance matrix, denoted as $\bar{\boldsymbol{\Sigma}}(\bar{\boldsymbol{z}})$, for all its shifted versions are given as:

$$\mathsf{circ}(\bar{\boldsymbol{z}}) \doteq \begin{bmatrix} \mathsf{circ}(\boldsymbol{z}[1]) \\ \vdots \\ \mathsf{circ}(\boldsymbol{z}[C]) \end{bmatrix} \in \mathbb{R}^{nC \times n}, \quad \bar{\boldsymbol{\Sigma}}(\bar{\boldsymbol{z}}) \doteq \begin{bmatrix} \mathsf{circ}(\boldsymbol{z}[1]) \\ \vdots \\ \mathsf{circ}(\boldsymbol{z}[C]) \end{bmatrix} \begin{bmatrix} \mathsf{circ}(\boldsymbol{z}[1])^*, \ldots, \mathsf{circ}(\boldsymbol{z}[C])^* \end{bmatrix} \in \mathbb{R}^{nC \times nC}, \tag{37}$$

where $\mathsf{circ}(\boldsymbol{z}[c]) \in \mathbb{R}^{n \times n}$ with $c \in [C]$ is the circulant version of the c-th channel of the feature $\bar{\boldsymbol{z}}$. Then the columns of $\mathsf{circ}(\bar{\boldsymbol{z}})$ will only span at most an n-dimensional proper subspace in $\mathbb{R}^{nC}$.

However, this simple lifting operation (if linear) is not sufficient to render the classes separable yet—features associated with other classes will span the *same* n-dimensional subspace. This reflects a fundamental conflict between invariance and linear (subspace) modeling: *one cannot hope for arbitrarily shifted and superposed signals to belong to the same class.*

One way of resolving this conflict is to leverage additional structure within each class, in the form of *sparsity*: Signals within each class are not generated as arbitrary linear superposition of some base atoms (or motifs), but only *sparse* combinations of them and their shifted versions, as shown in Figure 8. More precisely, let $\boldsymbol{D}^j = [\boldsymbol{d}_1^j, \ldots, \boldsymbol{d}_c^j]$ denote a matrix with a collection of atoms associated for class j, also known as a dictionary, then each signal $\boldsymbol{x}$ in this class is sparsely generated as:

$$\boldsymbol{x} = \boldsymbol{d}_1^j \circledast \boldsymbol{z}_1 + \ldots + \boldsymbol{d}_c^j \circledast \boldsymbol{z}_c = \mathsf{circ}(\boldsymbol{D}^j)\boldsymbol{z}, \tag{38}$$

for some sparse vector $\boldsymbol{z}$. Signals in different classes are then generated by different dictionaries whose atoms (or motifs) are incoherent from one another. Due to incoherence, signals in one class are unlikely to be sparsely represented by atoms in any other class. Hence all signals in the k class can be represented as

$$\boldsymbol{x} = \big[\mathsf{circ}(\boldsymbol{D}^1), \mathsf{circ}(\boldsymbol{D}^2), \ldots, \mathsf{circ}(\boldsymbol{D}^k)\big]\bar{\boldsymbol{z}}, \tag{39}$$

where $\bar{\boldsymbol{z}}$ is sparse.[29] There is a vast literature on how to learn the most compact and optimal sparsifying dictionaries from sample data, e.g. (Li and Bresler, 2019; Qu et al., 2019) and

27. For 1D signals like audio, one may consider the conventional short time Fourier transform (STFT); for 2D images, one may consider 2D wavelets as in the ScatteringNet (Bruna and Mallat, 2013).
28. For learned filters, one can learn filters as the principal components of samples as in the PCANet (Chan et al., 2015) or from convolution dictionary learning (Li and Bresler, 2019; Qu et al., 2019).
29. Notice that similar sparse representation models have long been proposed and used for classification purposes in applications such a face recognition, demonstrating excellent effectiveness (Wright et al., 2009; Wagner et al., 2012). Recently, the convolution sparse coding model has been proposed by Papyan et al. (2017) as a framework for interpreting the structures of deep convolution networks.

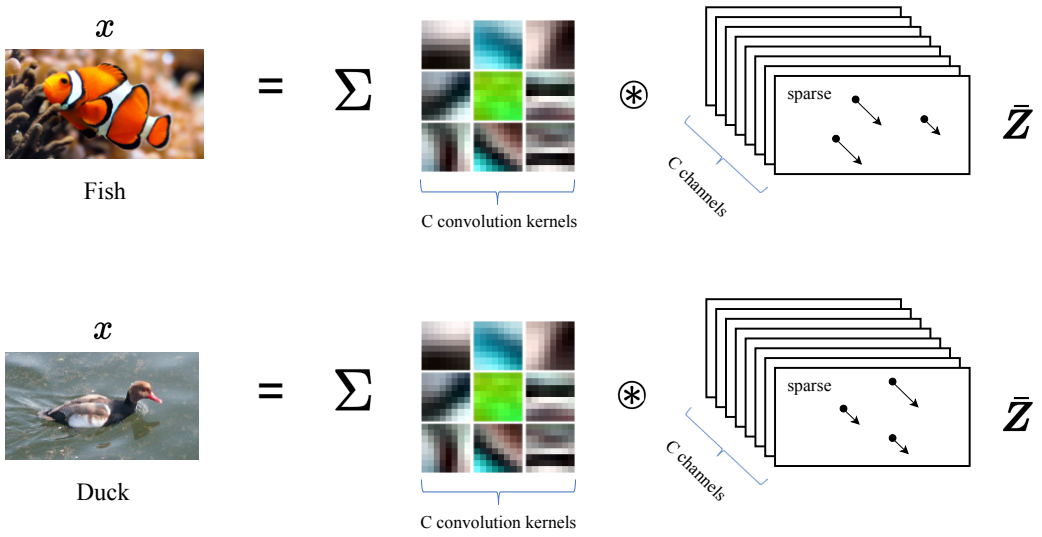


Figure 8: Each input signal $\boldsymbol{x}$ (an image here) can be represented as a superposition of sparse convolutions with multiple kernels $\boldsymbol{d}_c$ in a dictionary $\boldsymbol{D}$.

subsequently solve the inverse problem and compute the associated sparse code $\boldsymbol{z}$ or $\bar{\boldsymbol{z}}$. Recent studies of Qu et al. (2020a,b) even show under broad conditions the convolution dictionary learning problem can be solved effectively and efficiently.

Nevertheless, for tasks such as classification, we are not necessarily interested in the precise optimal dictionary nor the precise sparse code for each individual signal. We are mainly interested if collectively the set of sparse codes for each class are adequately separable from those of other classes. Under the assumption of the sparse generative model, if the convolution kernels $\{\boldsymbol{k}_c\}_{c=1}^C$ match well with the "transpose" or "inverse" of the above sparsifying dictionaries $\boldsymbol{D} = [\boldsymbol{D}^1, \ldots, \boldsymbol{D}^k]$, also known as the *analysis filters* (Nam et al., 2013; Rubinstein and Elad, 2014), signals in one class will only have high responses to a small subset of those filters and low responses to others (due to the incoherence assumption). Nevertheless, in practice, often a sufficient number of, say C, random filters $\{\boldsymbol{k}_c\}_{c=1}^C$ suffice the purpose of ensuring so extracted C-channel features:

$$\big[\boldsymbol{k}_1 \circledast \boldsymbol{x}, \boldsymbol{k}_2 \circledast \boldsymbol{x}, \ldots, \boldsymbol{k}_C \circledast \boldsymbol{x}\big]^* = \big[\mathsf{circ}(\boldsymbol{k}_1)\boldsymbol{x}, \ldots, \mathsf{circ}(\boldsymbol{k}_C)\boldsymbol{x}\big]^* \in \mathbb{R}^{C \times n} \tag{40}$$

for different classes have different response patterns to different filters hence make different classes separable (Chan et al., 2015).

Therefore, in our framework, to a large extent the number of channels (or the width of the network) truly plays the role as the *statistical resource* whereas the number of layers (the depth of the network) plays the role as the *computational resource*. The theory of compressive sensing precisely characterizes how many measurements are needed in order to preserve the intrinsic low-dimensional structures (including separability) of the data (Wright

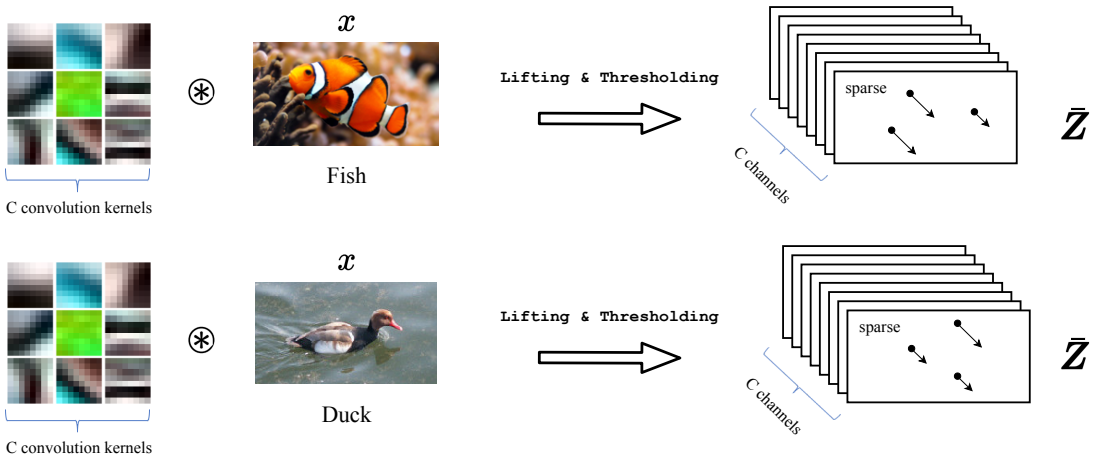


Figure 9: Estimate the sparse code $\bar{\boldsymbol{z}}$ of an input signal $\boldsymbol{x}$ (an image here) by taking convolutions with multiple kernels $\boldsymbol{k}_c$ and then sparsifying.

and Ma, 2021). As optimal sparse coding is not the focus of this paper, we will use the simple random filter design in our experiments, which is adequate to verify the concept.[30]

The multi-channel responses $\bar{\boldsymbol{z}}$ should be sparse. So to approximate the sparse code $\bar{\boldsymbol{z}}$, we may take an entry-wise *sparsity-promoting nonlinear thresholding*, say $\boldsymbol{\tau}(\cdot)$, on the above filter outputs by setting low (say absolute value below ϵ) or negative responses to be zero:

$$\bar{\boldsymbol{z}} \doteq \boldsymbol{\tau}\left(\left[\mathsf{circ}(\boldsymbol{k}_1)\boldsymbol{x}, \ldots, \mathsf{circ}(\boldsymbol{k}_C)\boldsymbol{x}\right]^*\right) \quad \in \mathbb{R}^{C\times n}. \tag{41}$$

Figure 9 illustrates the basic ideas. One may refer to Rubinstein and Elad (2014) for a more systematical study on the design of the sparsifying thresholding operator. Nevertheless, here we are not so interested in obtaining the best sparse codes as long as the codes are sufficiently separable. Hence the nonlinear operator $\boldsymbol{\tau}$ can be simply chosen to be a soft thresholding or a ReLU. These presumably sparse features $\bar{\boldsymbol{z}}$ can be assumed to lie on a lower-dimensional (nonlinear) submanifold of $\mathbb{R}^{nC}$, which can be linearized and separated from the other classes by subsequent ReduNet layers, as illustrated later in Figure 11.

The ReduNet constructed from circulant version of these multi-channel features $\bar{\boldsymbol{Z}} \doteq [\bar{\boldsymbol{z}}^1, \ldots, \bar{\boldsymbol{z}}^m] \in \mathbb{R}^{C\times n\times m}$, i.e., $\mathsf{circ}(\bar{\boldsymbol{Z}}) \doteq [\mathsf{circ}(\bar{\boldsymbol{z}}^1), \ldots, \mathsf{circ}(\bar{\boldsymbol{z}}^m)] \in \mathbb{R}^{nC\times nm}$, retains the good invariance properties described above: the linear operators, now denoted as $\bar{\boldsymbol{E}}$ and $\bar{\boldsymbol{C}}^j$, remain block circulant, and represent *multi-channel 1D circular convolutions*. Specifically, we have the following result (see Appendix B.2 for a proof).

Proposition 5 (Multi-channel convolution structures of $\bar{\boldsymbol{E}}$ and $\bar{\boldsymbol{C}}^j$) *The matrix*

$$\bar{\boldsymbol{E}} \doteq \alpha\left(\boldsymbol{I} + \alpha\mathsf{circ}(\bar{\boldsymbol{Z}})\mathsf{circ}(\bar{\boldsymbol{Z}})^*\right)^{-1} \tag{42}$$

30. Although better learned or designed sparsifying dictionaries and sparse coding schemes may surely lead to better classification performance, at a higher computational cost.

is block circulant, i.e.,

$$\bar{\boldsymbol{E}} = \begin{bmatrix} \bar{\boldsymbol{E}}_{1,1} & \cdots & \bar{\boldsymbol{E}}_{1,C} \\ \vdots & \ddots & \vdots \\ \bar{\boldsymbol{E}}_{C,1} & \cdots & \bar{\boldsymbol{E}}_{C,C} \end{bmatrix} \in \mathbb{R}^{nC \times nC},$$

where each $\bar{\boldsymbol{E}}_{c,c'} \in \mathbb{R}^{n \times n}$ is a circulant matrix. Moreover, $\bar{\boldsymbol{E}}$ represents a multi-channel circular convolution, i.e., for any multi-channel signal $\bar{\boldsymbol{z}} \in \mathbb{R}^{C \times n}$ we have

$$\bar{\boldsymbol{E}} \cdot \mathsf{vec}(\bar{\boldsymbol{z}}) = \mathsf{vec}(\bar{\boldsymbol{e}} \circledast \bar{\boldsymbol{z}}).$$

In above, $\bar{\boldsymbol{e}} \in \mathbb{R}^{C \times C \times n}$ is a multi-channel convolutional kernel with $\bar{\boldsymbol{e}}[c, c'] \in \mathbb{R}^n$ being the first column vector of $\bar{\boldsymbol{E}}_{c,c'}$, and $\bar{\boldsymbol{e}} \circledast \bar{\boldsymbol{z}} \in \mathbb{R}^{C \times n}$ is the multi-channel circular convolution defined as

$$(\bar{\boldsymbol{e}} \circledast \bar{\boldsymbol{z}})[c] \doteq \sum_{c'=1}^{C} \bar{\boldsymbol{e}}[c, c'] \circledast \bar{\boldsymbol{z}}[c'], \quad \forall c = 1, \ldots, C.$$

Similarly, the matrices $\bar{\boldsymbol{C}}^j$ associated with any subsets of $\bar{\boldsymbol{Z}}$ are also multi-channel circular convolutions.

From Proposition 5, shift invariant ReduNet is a deep convolutional network for multi-channel 1D signals by construction. Notice that even if the initial lifting kernels are separated (41), the matrix inverse in (42) for computing $\bar{\boldsymbol{E}}$ (similarly for $\bar{\boldsymbol{C}}^j$) introduces "cross talk" among all C channels. This multi-channel mingling effect will become more clear when we show below how to compute $\bar{\boldsymbol{E}}$ and $\bar{\boldsymbol{C}}^j$ efficiently in the frequency domain. Hence, unlike Xception nets (Chollet, 2017), these multi-channel convolutions in general are *not* depth-wise separable.[31]

4.3 Fast Computation in the Spectral Domain

The calculation of $\bar{\boldsymbol{E}}$ in (42) requires inverting a matrix of size $nC \times nC$, which has complexity $O(n^3C^3)$. By using the relationship between circulant matrix and Discrete Fourier Transform (DFT) of a 1D signal, this complexity can be significantly reduced.

Specifically, let $\boldsymbol{F} \in \mathbb{C}^{n \times n}$ be the DFT matrix,[32] and $\mathrm{DFT}(\boldsymbol{z}) \doteq \boldsymbol{F}\boldsymbol{z} \in \mathbb{C}^{n \times n}$ be the DFT of $\boldsymbol{z} \in \mathbb{R}^n$, where $\mathbb{C}$ denotes the set of complex numbers. We know all circulant matrices can be simultaneously diagonalized by the discrete Fourier transform matrix $\boldsymbol{F}$:

$$\mathsf{circ}(\boldsymbol{z}) = \boldsymbol{F}^* \mathsf{diag}(\mathrm{DFT}(\boldsymbol{z}))\boldsymbol{F}. \tag{43}$$

We refer the reader to Fact 5 of the Appendix B.3 for more detailed properties of circulant matrices and DFT. Hence the covariance matrix $\bar{\boldsymbol{\Sigma}}(\bar{\boldsymbol{z}})$ of the form (37) can be converted to a standard "blocks of diagonals" form:

$$\bar{\boldsymbol{\Sigma}}(\bar{\boldsymbol{z}}) = \begin{bmatrix} \boldsymbol{F}^* & \boldsymbol{0} & \boldsymbol{0} \\ \boldsymbol{0} & \ddots & \boldsymbol{0} \\ \boldsymbol{0} & \boldsymbol{0} & \boldsymbol{F}^* \end{bmatrix} \begin{bmatrix} \boldsymbol{D}_{11}(\bar{\boldsymbol{z}}) & \cdots & \boldsymbol{D}_{1C}(\bar{\boldsymbol{z}}) \\ \vdots & \ddots & \vdots \\ \boldsymbol{D}_{C1}(\bar{\boldsymbol{z}}) & \cdots & \boldsymbol{D}_{CC}(\bar{\boldsymbol{z}}) \end{bmatrix} \begin{bmatrix} \boldsymbol{F} & \boldsymbol{0} & \boldsymbol{0} \\ \boldsymbol{0} & \ddots & \boldsymbol{0} \\ \boldsymbol{0} & \boldsymbol{0} & \boldsymbol{F} \end{bmatrix} \in \mathbb{R}^{nC \times nC}, \tag{44}$$

31. It remains open what additional structures on the data would lead to depth-wise separable convolutions.
32. Here we scaled the matrix $\boldsymbol{F}$ to be unitary, hence it differs from the conventional DFT matrix by a $1/\sqrt{n}$.

where $\boldsymbol{D}_{cc'}(\bar{\boldsymbol{z}}) \doteq \mathsf{diag}(\mathrm{DFT}(\boldsymbol{z}[c])) \cdot \mathsf{diag}(\mathrm{DFT}(\boldsymbol{z}[c']))^* \in \mathbb{C}^{n\times n}$ is a diagonal matrix. The middle of RHS of (44) is a block diagonal matrix after a permutation of rows and columns.

Given a collection of multi-channel features $\{\bar{\boldsymbol{z}}^i \in \mathbb{R}^{C\times n}\}_{i=1}^m$, we can use the relation in (44) to compute $\bar{\boldsymbol{E}}$ (and similarly for $\bar{\boldsymbol{C}}^j$) as

$$\bar{\boldsymbol{E}} = \begin{bmatrix} \boldsymbol{F}^* & \boldsymbol{0} & \boldsymbol{0} \\ \boldsymbol{0} & \ddots & \boldsymbol{0} \\ \boldsymbol{0} & \boldsymbol{0} & \boldsymbol{F}^* \end{bmatrix} \cdot \alpha \left(\boldsymbol{I} + \alpha \sum_{i=1}^m \begin{bmatrix} \boldsymbol{D}_{11}(\bar{\boldsymbol{z}}^i) & \cdots & \boldsymbol{D}_{1C}(\bar{\boldsymbol{z}}^i) \\ \vdots & \ddots & \vdots \\ \boldsymbol{D}_{C1}(\bar{\boldsymbol{z}}^i) & \cdots & \boldsymbol{D}_{CC}(\bar{\boldsymbol{z}}^i) \end{bmatrix} \right)^{-1} \cdot \begin{bmatrix} \boldsymbol{F} & \boldsymbol{0} & \boldsymbol{0} \\ \boldsymbol{0} & \ddots & \boldsymbol{0} \\ \boldsymbol{0} & \boldsymbol{0} & \boldsymbol{F} \end{bmatrix} \in \mathbb{R}^{nC\times nC}. \tag{45}$$

The matrix in the inverse operator is a block diagonal matrix with n blocks of size $C \times C$ after a permutation of rows and columns. Hence, to compute $\bar{\boldsymbol{E}}$ and $\bar{\boldsymbol{C}}^j \in \mathbb{R}^{nC\times nC}$, we only need to compute in the frequency domain the inverse of $C \times C$ blocks for n times and the overall complexity is $O(nC^3)$. [33]

The benefit of computation with DFT motivates us to construct the ReduNet in the spectral domain. Let us consider the *shift invariant coding rate reduction* objective for shift invariant features $\{\bar{\boldsymbol{z}}^i \in \mathbb{R}^{C\times n}\}_{i=1}^m$:

$$\begin{aligned} &\Delta R_{\mathsf{circ}}(\bar{\boldsymbol{Z}}, \boldsymbol{\Pi}) \doteq \frac{1}{n}\Delta R(\mathsf{circ}(\bar{\boldsymbol{Z}}), \bar{\boldsymbol{\Pi}}) \\ &= \frac{1}{2n}\log\det\left(\boldsymbol{I} + \alpha \mathsf{circ}(\bar{\boldsymbol{Z}})\mathsf{circ}(\bar{\boldsymbol{Z}})^*\right) - \sum_{j=1}^k \frac{\gamma_j}{2n}\log\det\left(\boldsymbol{I} + \alpha_j \mathsf{circ}(\bar{\boldsymbol{Z}})\bar{\boldsymbol{\Pi}}^j\mathsf{circ}(\bar{\boldsymbol{Z}})^*\right), \end{aligned} \tag{46}$$

where $\alpha = \frac{Cn}{mn\epsilon^2} = \frac{C}{m\epsilon^2}$, $\alpha_j = \frac{Cn}{\mathsf{tr}(\boldsymbol{\Pi}^j)n\epsilon^2} = \frac{C}{\mathsf{tr}(\boldsymbol{\Pi}^j)\epsilon^2}$, $\gamma_j = \frac{\mathsf{tr}(\boldsymbol{\Pi}^j)}{m}$, and $\bar{\boldsymbol{\Pi}}^j$ is augmented membership matrix in an obvious way. The normalization factor n is introduce because the circulant matrix $\mathsf{circ}(\bar{\boldsymbol{Z}})$ contains n (shifted) copies of each signal. Next, we derive the ReduNet for maximizing $\Delta R_{\mathsf{circ}}(\bar{\boldsymbol{Z}}, \boldsymbol{\Pi})$.

Let $\mathrm{DFT}(\bar{\boldsymbol{Z}}) \in \mathbb{C}^{C\times n\times m}$ be data in spectral domain obtained by taking DFT on the second dimension and denote $\mathrm{DFT}(\bar{\boldsymbol{Z}})(p) \in \mathbb{C}^{C\times m}$ the p-th slice of $\mathrm{DFT}(\bar{\boldsymbol{Z}})$ on the second dimension. Then, the gradient of $\Delta R_{\mathsf{circ}}(\bar{\boldsymbol{Z}}, \boldsymbol{\Pi})$ w.r.t. $\bar{\boldsymbol{Z}}$ can be computed from the expansion $\bar{\mathcal{E}} \in \mathbb{C}^{C\times C\times n}$ and compression $\bar{\mathcal{C}}^j \in \mathbb{C}^{C\times C\times n}$ operators in the spectral domain, defined as

$$\begin{aligned} \bar{\mathcal{E}}(p) &\doteq \alpha \cdot \left[\boldsymbol{I} + \alpha \cdot \mathrm{DFT}(\bar{\boldsymbol{Z}})(p) \cdot \mathrm{DFT}(\bar{\boldsymbol{Z}})(p)^*\right]^{-1} \quad \in \mathbb{C}^{C\times C}, \\ \bar{\mathcal{C}}^j(p) &\doteq \alpha_j \cdot \left[\boldsymbol{I} + \alpha_j \cdot \mathrm{DFT}(\bar{\boldsymbol{Z}})(p) \cdot \boldsymbol{\Pi}_j \cdot \mathrm{DFT}(\bar{\boldsymbol{Z}})(p)^*\right]^{-1} \quad \in \mathbb{C}^{C\times C}. \end{aligned} \tag{47}$$

In above, $\bar{\mathcal{E}}(p)$ (resp., $\bar{\mathcal{C}}^j(p)$) is the p-th slice of $\bar{\mathcal{E}}$ (resp., $\bar{\mathcal{C}}^j$) on the last dimension. Specifically, we have the following result (see Appendix B.3 for a complete proof).

Theorem 6 (Computing multi-channel convolutions $\bar{\boldsymbol{E}}$ and $\bar{\boldsymbol{C}}^j$) *Let $\bar{\boldsymbol{U}} \in \mathbb{C}^{C\times n\times m}$ and $\bar{\boldsymbol{W}}^j \in \mathbb{C}^{C\times n\times m}, j = 1, \ldots, k$ be given by*

$$\bar{\boldsymbol{U}}(p) \doteq \bar{\mathcal{E}}(p) \cdot \mathrm{DFT}(\bar{\boldsymbol{Z}})(p), \tag{48}$$

$$\bar{\boldsymbol{W}}^j(p) \doteq \bar{\mathcal{C}}^j(p) \cdot \mathrm{DFT}(\bar{\boldsymbol{Z}})(p), \tag{49}$$

33. There is strong scientific evidence that neurons in the visual cortex encode and transmit information in the rate of spiking, hence the so-called spiking neurons (Softky and Koch, 1993; Eliasmith and Anderson, 2003). Nature might be exploiting the computational efficiency in the frequency domain for achieving shift invariance.

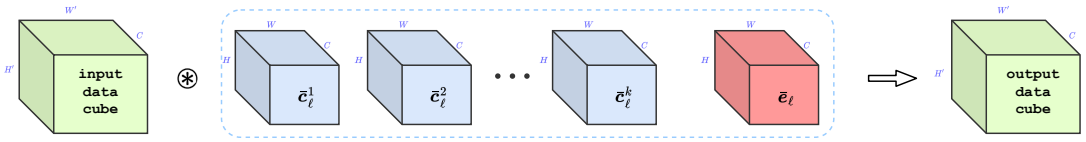


Figure 10: For invariance to 2D translation, $\bar{\boldsymbol{E}}$ and $\bar{\boldsymbol{C}}^j$ are automatically multi-channel 2D convolutions.

for each $p \in \{0, \ldots, n-1\}$. *Then, we have*

$$\frac{1}{2n}\frac{\partial \log\det(\boldsymbol{I} + \alpha \cdot \mathsf{circ}(\bar{\boldsymbol{Z}})\mathsf{circ}(\bar{\boldsymbol{Z}})^*)}{\partial \bar{\boldsymbol{Z}}} = \mathrm{IDFT}(\bar{\boldsymbol{U}}), \tag{50}$$

$$\frac{\gamma_j}{2n}\frac{\partial \log\det(\boldsymbol{I} + \alpha_j \cdot \mathsf{circ}(\bar{\boldsymbol{Z}})\bar{\boldsymbol{\Pi}}^j\mathsf{circ}(\bar{\boldsymbol{Z}})^*)}{\partial \bar{\boldsymbol{Z}}} = \gamma_j \cdot \mathrm{IDFT}(\bar{\boldsymbol{W}}^j\boldsymbol{\Pi}^j). \tag{51}$$

In above, $\mathrm{IDFT}(\bar{\boldsymbol{U}})$ *is the time domain signal obtained by taking inverse DFT on each channel of each signal in* $\bar{\boldsymbol{U}}$.

By this result, the gradient ascent update in (13) (when applied to $\Delta R_{\mathsf{circ}}(\bar{\boldsymbol{Z}}, \boldsymbol{\Pi})$) can be equivalently expressed as an update in spectral domain on $\bar{\boldsymbol{V}}_\ell \doteq \mathrm{DFT}(\bar{\boldsymbol{Z}}_\ell)$ as

$$\bar{\boldsymbol{V}}_{\ell+1}(p) \propto \bar{\boldsymbol{V}}_\ell(p) + \eta\left(\bar{\mathcal{E}}_\ell(p)\cdot\bar{\boldsymbol{V}}_\ell(p) - \sum_{j=1}^{k}\gamma_j\bar{\mathcal{C}}_\ell^j(p)\cdot\bar{\boldsymbol{V}}_\ell(p)\boldsymbol{\Pi}^j\right), \quad p = 0, \ldots, n-1, \tag{52}$$

and a ReduNet can be constructed in a similar fashion as before. For implementation details, we refer the reader to Algorithm 3 of Appendix B.3.

4.4 2D Translation Invariance

In the case of classifying images invariant to arbitrary 2D translation, we may view the image (feature) $\boldsymbol{z} \in \mathbb{R}^{(W\times H)\times C}$ as a function defined on a torus $\mathcal{T}^2$ (discretized as a $W \times H$ grid) and consider $\mathbb{G}$ to be the (Abelian) group of all 2D (circular) translations on the torus. As we will show in the Appendix C, the associated linear operators $\bar{\boldsymbol{E}}$ and $\bar{\boldsymbol{C}}^j$'s act on the image feature $\boldsymbol{z}$ as *multi-channel 2D circular convolutions*, as shown in Figure 10. The resulting network will be a deep convolutional network that shares the same multi-channel convolution structures as empirically designed CNNs for 2D images (LeCun et al., 1995; Krizhevsky et al., 2012) or ones suggested for promoting sparsity (Papyan et al., 2017)! The difference is that, again, the convolution architectures and parameters of our network are derived from the rate reduction objective, and so are all the nonlinear activations. Like the 1D signal case, the derivation in Appendix C shows that this convolutional network can be constructed much more efficiently in the spectral domain. See Theorem 17 of Appendix C for a rigorous statement and justification.

4.5 Overall Network Architecture and Comparison

Following the above derivation, we see that in order to find a linear discriminative representation (LDR) for multiple classes of signals/images that is invariant to translation,

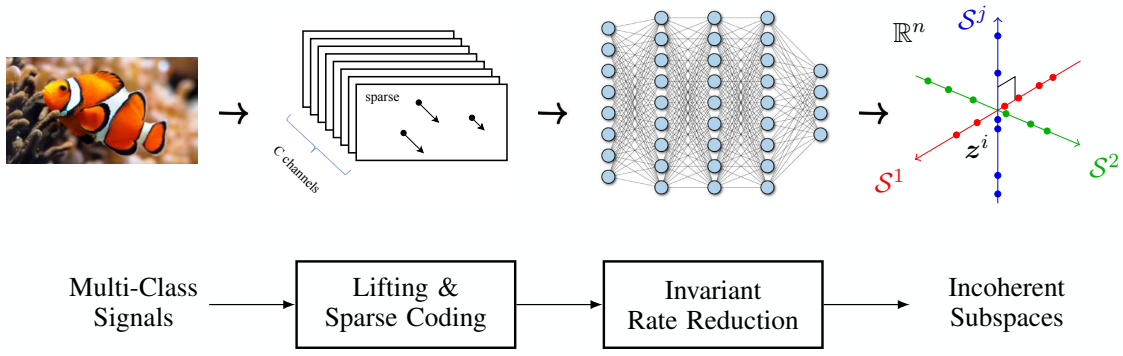


Figure 11: The overall process for classifying multi-class signals with shift invariance: Multi-channel lifting, sparse coding, followed by a multi-channel convolution ReduNet for invariant rate reduction. These components are *necessary* in order to map shift-invariant multi-class signals to incoherent (linear) subspaces as an LDR. Note that the architectures of most modern deep neural networks resemble this process. The so-learned LDR facilitates subsequent tasks such as classification.

sparse coding, a multi-layer architecture with multi-channel convolutions, different nonlinear activation, and spectrum computing all become *necessary* components for achieving the objective effectively and efficiently. Figure 11 illustrates the overall process of learning such a representation via invariant rate reduction on the input sparse codes.

Connections to convolutional and recurrent sparse coding. As we have discussed in the introduction, it has been long noticed that there are connections between sparse coding and deep networks, including the work of Learned ISTA (Gregor and LeCun, 2010) and many of its convolutional and recurrent variants (Wisdom et al., 2016; Papyan et al., 2017; Sulam et al., 2018; Monga et al., 2019). Although both sparsity and convolution are advocated as desired characteristics for such networks, their precise roles for the classification task have never been fully revealed. For instance, Papyan et al. (2017) has suggested a convolutional sparse coding framework for the CNNs. It actually aligns well with the early lifting and sparse coding stage of the overall process. Nevertheless, our new framework suggests, once such sparse codes are obtained, one needs subsequent ReduNet to transform them to the desired LDRs. Through our derivations, we see how lifting and sparse coding (via random filtering or sparse deconvolution), multi-channel convolutions $(\bar{\boldsymbol{E}}, \bar{\boldsymbol{C}}^j)$, and different nonlinear operations: grouping $\widehat{\boldsymbol{\pi}}^j(\cdot)$, normalization $\mathcal{P}_{\mathbb{S}^{n-1}}(\cdot)$, and soft thresholding $\boldsymbol{\tau}(\cdot)$ are all derived as *necessary processes* from the objective of maximizing rate reduction of the learned features while enforcing shift invariance.

Comparison with ScatteringNet and PCANet. Using convolutional operators/with pooling to ensure equivarience/invariance have been common practice in deep networks (LeCun and Bengio, 1995; Cohen and Welling, 2016), but the number of convolutions needed has never been clear and their parameters need to be learned via back propagation from randomly initialized ones. Of course, one may also choose a complete basis for the convolution filters in each layer to ensure translational equivariance/invariance for a wide range of signals.

ScatteringNet (Bruna and Mallat, 2013) and many followup works (Wiatowski and Bölcskei, 2018) have shown to use modulus of 2D wavelet transform to construct invariant features. However, the number of convolutions needed usually grow *exponentially* in the number of layers. That is the reason why ScatteringNet type networks cannot be so deep and typically limited to only 2-3 layers. In practice, it is often used in a hybrid setting Zarka et al. (2020, 2021) for better performance and scalability. On the other hand, PCANet (Chan et al., 2015) argues that one can significantly reduce the number of convolution channels by learning features directly from the data. In contrast to ScatteringNet and PCANet, the proposed invariant ReduNet *by construction* learns equivariant features from the data that are discriminative between classes. As experiments on MNIST in Appendix E.5 show, the representations learned by ReduNet indeed preserve translation information. In fact, scattering transform can be used with ReduNet in a complementary fashion. One may replace the aforementioned random (lifting) filters with the scattering transform. As experiments in the Appendix E.5 show this can yield significantly better classification performance.

Notice that, none of the previous "convolution-by-design" approaches explain why we need *multi-channel* (3D) convolutions instead of separable (2D) ones, let alone how to design them. In contrast, in the new rate reduction framework, we see that both the forms and roles of the multi-channel convolutions $(\bar{\boldsymbol{E}}, \bar{\boldsymbol{C}}^j)$ are explicitly derived and justified, the number of filters (channels) remains constant through all layers, and even values of their parameters are determined by the data of interest. Of course, as mentioned before the values of the parameters can be further fine-tuned to improve performance, as we will see in the experimental section (Table 2).

Sparse coding, spectral computing, and subspace embedding in nature. Notice that *sparse coding* has long been hypothesized as the guiding organization principle for the visual cortex of primates (Olshausen and Field, 1996; Chen et al., 2018b). Through years of evolution, the visual cortex has learned to sparsely encode the visual input with all types of localized and oriented filters. Interstingly, there have been strong scientific evidences that neurons in the visual cortex transmit and process information in terms of *rates of spiking*, i.e. in the spectrum rather than the magnitude of signals, hence the so-called "spiking neurons" (Softky and Koch, 1993; Eliasmith and Anderson, 2003; Belitski et al., 2008). Even more interestingly, recent studies in neuroscience have started to reveal how these mechanisms might be integrated in the inferotemporal (IT) cortex, where neurons encode and process information about high-level object identity (e.g. face recognition), invariant to various transformations (Majaj et al., 2015; Chang and Tsao, 2017). In particular, Chang and Tsao (2017) went even further to hypothesize that high-level neurons encode the face space as a *linear subspace* with each cell likely encoding one axis of the subspace (rather than previously thought "an exemplar"). The framework laid out in this paper suggests that such a "high-level" compact (linear and discriminative) representation can be efficiently and effectively learned in the spectrum domain via an arguably much simpler and more natural "forward propagation" mechanism. Maybe, just maybe, nature has already learned to exploit what mathematics reveals as the most parsimonious and economic.

5. Experimental Verification

In this section, we conduct experiments to (1). *Validate* the effectiveness of the proposed maximal coding rate reduction ($\mathbf{MCR}^2$) principle analyzed in Section 2; and (2). *Verify* whether the constructed **ReduNet**, including the basic vector case ReduNet in Section 3 and the invariance ReduNet in Section 4, achieves its design objectives through experiments. Our goal in this work is not to push the state of the art performance on any real datasets with additional engineering ideas and heuristics, although the experimental results clearly suggest this potential in the future. All code is implemented in Python mainly using NumPy and PyTorch. All of our experiments are conducted in a computing node with 2.1 GHz Intel Xeon Silver CPU, 256GB of memory and 2 Nvidia RTX2080. Implementation details and many more experiments and can be found in Appendix D and E.

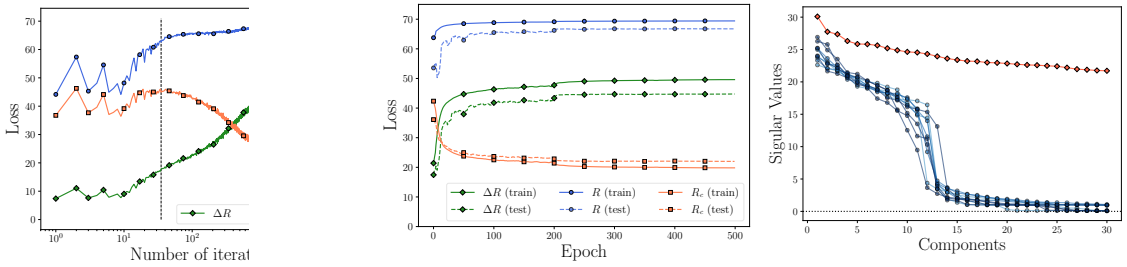
5.1 Experimental Verification of the MCR^2 Objective

In this subsection, we present experimental results on investigating the MCR^2 objective function for training neural networks. Our theoretical analysis in Section 2 shows how the *maximal coding rate reduction* (MCR^2) is a principled measure for learning discriminative and diverse representations for mixed data. In this section, we demonstrate experimentally how this principle alone, *without any other heuristics,* is adequate to learning good representations. More specifically, we apply the widely adopted neural network architectures (such as ResNet (He et al., 2016)) as the feature mapping $\boldsymbol{z} = f(\boldsymbol{x}, \boldsymbol{\theta})$ and optimize the neural network parameters $\boldsymbol{\theta}$ to achieve maximal coding rate reduction. Our goal here is to validate effectiveness of this principle through its most basic usage and fair comparison with existing frameworks. More implementation details and experiments are given in Appendix D. The code for reproducing the results on the effectiveness of MCR^2 objective in this section can be found in `https://github.com/Ma-Lab-Berkeley/MCR2`.

Supervised learning via rate reduction. When class labels are provided during training, we assign the membership (diagonal) matrix $\boldsymbol{\Pi} = \{\boldsymbol{\Pi}^j\}_{j=1}^k$ as follows: for each sample $\boldsymbol{x}^i$ with label j, set $\boldsymbol{\Pi}^j(i,i) = 1$ and $\boldsymbol{\Pi}^l(i,i) = 0, \forall l \neq j$. Then the mapping $f(\cdot, \boldsymbol{\theta})$ can be learned by optimizing (11), where $\boldsymbol{\Pi}$ remains constant. We apply stochastic gradient descent to optimize MCR^2, and for each iteration we use mini-batch data $\{(\boldsymbol{x}^i, \boldsymbol{y}^i)\}_{i=1}^m$ to approximate the MCR^2 loss.

Evaluation via classification. As we will see, in the supervised setting, the learned representation has very clear subspace structures. So to evaluate the learned representations, we consider a natural nearest subspace classifier. For each class of learned features $\boldsymbol{Z}^j$, let $\boldsymbol{\mu}^j \in \mathbb{R}^n$ be the mean of the representation vectors of j-th class and $\boldsymbol{U}^j \in \mathbb{R}^{n \times r_j}$ be the first r_j principal components for $\boldsymbol{Z}^j$, where r_j is the estimated dimension of class j. The predicted label of a test data $\boldsymbol{x}'$ is given by $j' = \operatorname{argmin}_{j \in \{1,\ldots,k\}} \|(\boldsymbol{I} - \boldsymbol{U}^j(\boldsymbol{U}^j)^*)(f(\boldsymbol{x}', \boldsymbol{\theta}) - \boldsymbol{\mu}^j)\|_2^2$.

Experiments on real data. We consider CIFAR10 dataset (Krizhevsky, 2009) and ResNet-18 (He et al., 2016) for $f(\cdot, \boldsymbol{\theta})$. We replace the last linear layer of ResNet-18 by a two-layer fully connected network with ReLU activation function such that the output dimension is 128. We set the mini-batch size as $m = 1,000$ and the precision parameter $\epsilon^2 = 0.5$. More results can be found in Appendix D.3.2.



(a) Evolution of $R, R_c, \Delta R$ during the training process.

(b) Training loss versus testing loss.

(c) PCA: (**red**) overall data; (**blue**) individual classes.

Figure 12: Evolution of the rates of MCR^2 in the training process and principal components of learned features.

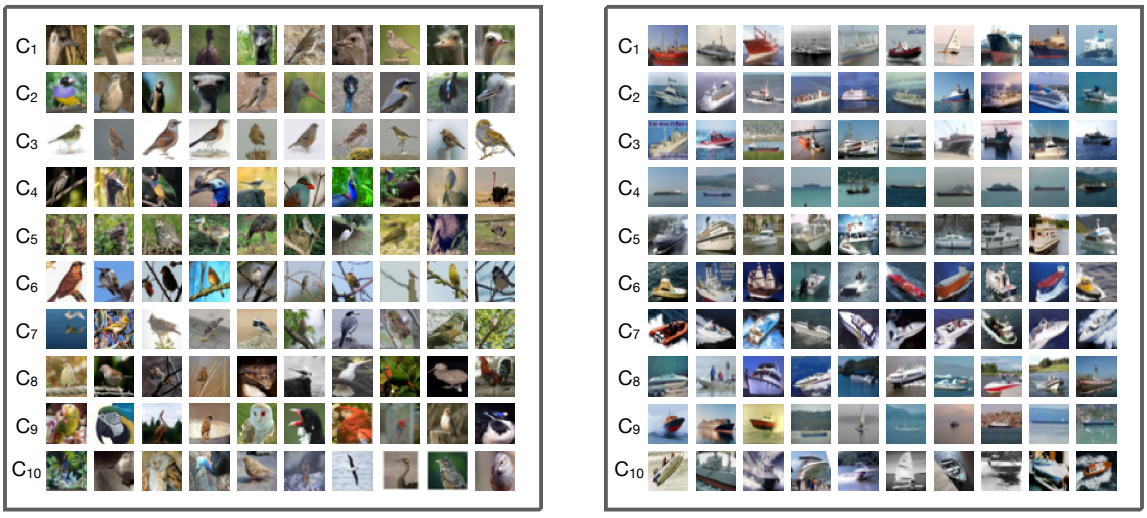
Figure 12(a) illustrates how the two rates and their difference (for both training and test data) evolves over epochs of training: After an initial phase, R gradually increases while R_c decreases, indicating that features $\boldsymbol{Z}$ are expanding as a whole while each class $\boldsymbol{Z}^j$ is being compressed. Figure 12(c) shows the distribution of singular values per $\boldsymbol{Z}^j$ and Figure 1 (right) shows the angles of features sorted by class. Compared to the geometric loss (Lezama et al., 2018), our features are *not only orthogonal but also of much higher dimension*. We compare the singular values of representations, both overall data and individual classes, learned by using cross-entropy and MCR^2 in Figure 18 and Figure 19 in Appendix D.3.1. We find that the representations learned by using MCR^2 loss are much more diverse than the ones learned by using cross-entropy loss. In addition, we find that we are able to select diverse images from the same class according to the "principal" components of the learned features (see Figure 13 and Figure 20).

One potential caveat of MCR^2 training is how to optimally select the output dimension n and training batch size m. For a given output dimension n, a sufficiently large batch size m is needed in order to achieve good classification performance. More detail study on varying output dimension n and batch size m is listed in Table 5 of Appendix D.3.2.

	Ratio=0.0	Ratio=0.1	Ratio=0.2	Ratio=0.3	Ratio=0.4	Ratio=0.5
CE Training	0.939	0.909	0.861	0.791	0.724	0.603
MCR^2 Training	**0.940**	**0.911**	**0.897**	**0.881**	**0.866**	**0.843**

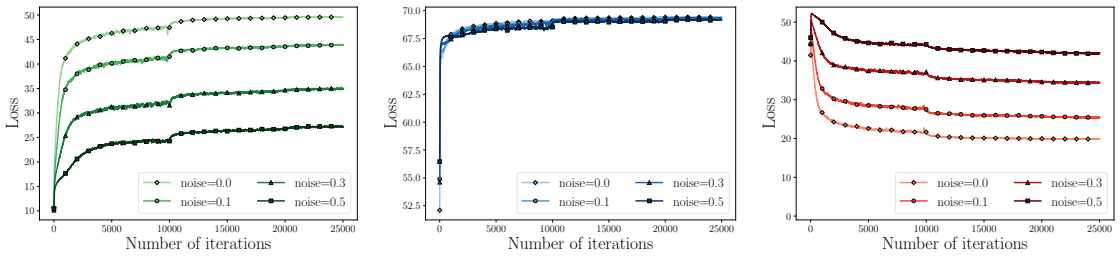
Table 1: Classification results with features learned with labels corrupted at different levels.

Robustness to corrupted labels. Because MCR^2 by design encourages richer representations that preserves intrinsic structures from the data $\boldsymbol{X}$, training relies less on class labels than traditional loss such as cross-entropy (CE). To verify this, we train the same network using both CE and MCR^2 with certain ratios of *randomly corrupted* training labels. Figure 14 illustrates the learning process: for different levels of corruption, while the rate for the whole set always converges to the same value, the rates for the classes are inversely proportional to the ratio of corruption, indicating our method only compresses samples with valid labels. The classification results are summarized in Table 1. By applying *exact*



(b) Ship

Figure 13: Visualization of principal components learned for class 2-'Bird' and class 8-'Ship'. For each class j, we first compute the top-10 singular vectors of the SVD of the learned features $\boldsymbol{Z}^j$. Then for the l-th singular vector of class j (denoted by $\boldsymbol{u}_j^l$), and for the feature of the i-th image of class j (denoted by $\boldsymbol{z}_j^i$), we calculate the absolute value of inner product, $|\langle \boldsymbol{z}_j^i, \boldsymbol{u}_j^l \rangle|$, then we select the top-10 images according to $|\langle \boldsymbol{z}_j^i, \boldsymbol{u}_j^l \rangle|$ for each singular vector. In the above two figures, each row corresponds to one singular vector (component C_l). The rows are sorted based on the magnitude of the associated singular values, from large to small.



(a) $\Delta R\big(\boldsymbol{Z}(\boldsymbol{\theta}), \boldsymbol{\Pi}, \epsilon\big)$. (b) $R(\boldsymbol{Z}(\boldsymbol{\theta}), \epsilon)$. (c) $R_c(\boldsymbol{Z}(\boldsymbol{\theta}), \epsilon \mid \boldsymbol{\Pi})$.

Figure 14: Evolution of rates $R, R_c, \Delta R$ of MCR^2 during training with corrupted labels.

the same training parameters, MCR^2 is significantly more robust than CE, especially with higher ratio of corrupted labels. This can be an advantage in the settings of self-supervised learning or constrastive learning when the grouping information can be very noisy. More detailed comparison between MCR^2 and OLE (Lezama et al., 2018), Large Margin Deep Networks (Elsayed et al., 2018), and ITLM (Shen and Sanghavi, 2019) on learning from noisy labels can be found in Appendix D.4 (Table 8).

Beside the supervised learning setting, we explore the MCR^2 objective in the self-supervised learning setting. We find that the MCR^2 objective can learn good representations

without using any label and achieve better performance over other highly engineered methods on clustering tasks. More details on self-supervised learning can be found in Section D.6.

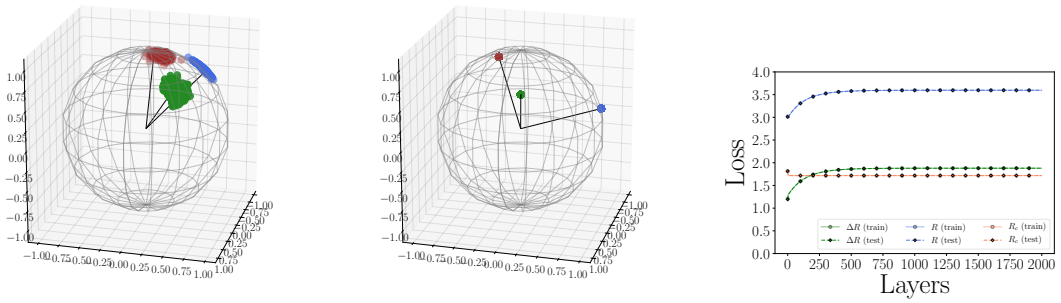
5.2 Experimental Verification of the ReduNet

In this section, we *verify* whether the so constructed ReduNet (developed in Section 3 and 4) achieves its design objectives through experiments on synthetic data and real images. The datasets and experiments are chosen to clearly demonstrate the properties and behaviors of the proposed ReduNet, in terms of learning the correct truly invariant discriminative (orthogonal) representation for the given data. Implementation details and more experiments and can be found in Appendix E. The code for reproducing the ReduNet results can be found in `https://github.com/Ma-Lab-Berkeley/ReduNet`.

Learning Mixture of Gaussians in $\mathbb{S}^2$. Consider a mixture of three Gaussian distributions in $\mathbb{R}^3$ that is projected onto $\mathbb{S}^2$. We first generate data points from these two distributions, $\boldsymbol{X}^1 = [\boldsymbol{x}_1^1, \ldots, \boldsymbol{x}_1^m] \in \mathbb{R}^{3\times m}$, $\boldsymbol{x}_1^i \sim \mathcal{N}(\boldsymbol{\mu}_1, \sigma_1^2\boldsymbol{I})$, and $\boldsymbol{\pi}(\boldsymbol{x}_1^i) = 1$; $\boldsymbol{X}^2 = [\boldsymbol{x}_2^1, \ldots, \boldsymbol{x}_2^m] \in \mathbb{R}^{2\times m}$, $\boldsymbol{x}_2^i \sim \mathcal{N}(\boldsymbol{\mu}_2, \sigma_2^2\boldsymbol{I})$, and $\boldsymbol{\pi}(\boldsymbol{x}_2^i) = 2$; $\boldsymbol{X}^3 = [\boldsymbol{x}_3^1, \ldots, \boldsymbol{x}_3^m] \in \mathbb{R}^{3\times m}$, $\boldsymbol{x}_3^i \sim \mathcal{N}(\boldsymbol{\mu}_3, \sigma_3^2\boldsymbol{I})$, and $\boldsymbol{\pi}(\boldsymbol{x}_2^i) = 3$. We set $m = 500, \sigma_1 = \sigma_2 = \sigma_3 = 0.1$ and $\boldsymbol{\mu}^1, \boldsymbol{\mu}^2, \boldsymbol{\mu}^3 \in \mathbb{S}^2$. Then we project all the data points onto $\mathbb{S}^2$, i.e., $\boldsymbol{x}_j^i/\|\boldsymbol{x}_j^i\|_2$. To construct the network (computing $\boldsymbol{E}_\ell, \boldsymbol{C}_\ell^j$ for ℓ-the layer), we set the number of iterations/layers $L = 2,000$[34], step size $\eta = 0.5$, and precision $\epsilon = 0.1$. As shown in Figure 15(a)-15(b), we can observe that after the mapping $f(\cdot, \boldsymbol{\theta})$, samples from the same class converge to a single cluster and the angle between two different clusters is approximately $\pi/4$, which is well aligned with the optimal solution $\boldsymbol{Z}_\star$ of the MCR^2 loss in $\mathbb{S}^2$. MCR^2 loss of features on different layers can be found in Figure 15(c). Empirically, we find that our constructed network is able to maximize MCR^2 loss and converges stably and samples from the same class converge to one cluster and different clusters are orthogonal to each other. Moreover, we sample new data points from the same distributions for both cases and find that new samples form the same class consistently converge to the same cluster center as the training samples. More simulation examples and details can be found in Appendix E.4.

Rotational Invariance on MNIST Digits. We now study the ReduNet on learning *rotation* invariant features on the real 10-class MNIST dataset (LeCun, 1998). We impose a polar grid on the image $\boldsymbol{x} \in \mathbb{R}^{H\times W}$, with its geometric center being the center of the 2D polar grid (as illustrated in Figure 22 in the Appendix). For each radius r_i, $i \in [C]$, we can sample Γ pixels with respect to each angle $\gamma_l = l \cdot (2\pi/\Gamma)$ with $l \in [\Gamma]$. Then given a sample image $\boldsymbol{x}$ from the dataset, we represent the image in the (sampled) polar coordinate as a multi-channel signal $\boldsymbol{x}_p \in \mathbb{R}^{\Gamma\times C}$. Our goal is to learn a rotation invariant representation, i.e., we expect to learn $f(\cdot, \boldsymbol{\theta})$ such that $\{f(\boldsymbol{x}_p \circ \mathfrak{g}, \boldsymbol{\theta})\}_{\mathfrak{g}\in\mathbb{G}}$ lie in the same subspace, where $\mathfrak{g}$ is the cyclic-shift in polar angle. We use $m = 100$ training samples (10 from each class) and set $\Gamma = 200$, $C = 15$ for polar sampling. By performing the above sampling in polar coordinate, we can obtain the data matrix $\boldsymbol{X}_p \in \mathbb{R}^{(\Gamma\cdot C)\times m}$. For the ReduNet, we set the

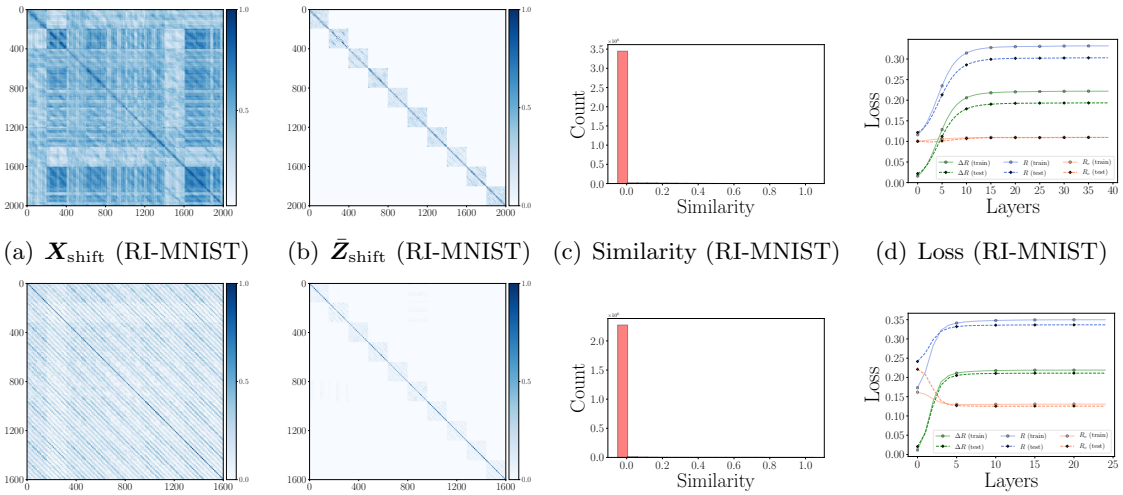
34. We do this only to demonstrate our framework leads to stable deep networks even with thousands of layers! In practice this is not necessary and one can stop whenever adding new layers gives diminishing returns. For this example, a couple of hundred is sufficient. Hence the clear optimization objective gives a natural criterion for the depth of the network needed. Remarks in Section 3.4 provide possible ideas to further reduce the number of layers.



(a) $\boldsymbol{X}_{\text{train}}$ $(3D)$ (b) $\boldsymbol{Z}_{\text{train}}$ $(3D)$ (c) Loss $(3D)$

Figure 15: Original samples and learned representations for 3D Mixture of Gaussians. We visualize data points $\boldsymbol{X}$ (before mapping $f(\cdot, \boldsymbol{\theta})$) and learned features $\boldsymbol{Z}$ (after mapping $f(\cdot, \boldsymbol{\theta})$) by scatter plot. In each scatter plot, each color represents one class of samples. We also show the plots for the progression of values of the objective functions.

number of layers/iterations $L = 40$, precision $\epsilon = 0.1$, step size $\eta = 0.5$. Before the first layer, we perform lifting of the input by 1D circulant-convolution with 20 random Gaussian kernels of size 5.



(a) $\boldsymbol{X}_{\text{shift}}$ (RI-MNIST) (b) $\bar{\boldsymbol{Z}}_{\text{shift}}$ (RI-MNIST) (c) Similarity (RI-MNIST) (d) Loss (RI-MNIST)

(e) $\boldsymbol{X}_{\text{shift}}$ (TI-MNIST) (f) $\bar{\boldsymbol{Z}}_{\text{shift}}$ (TI-MNIST) (g) Similarity (TI-MNIST) (h) Loss (TI-MNIST)

Figure 16: (a)(b) and (e)(f) are heatmaps of cosine similarity among shifted training data $\boldsymbol{X}_{\text{shift}}$ and learned features $\bar{\boldsymbol{Z}}_{\text{shift}}$, for rotation and translation invariance respectively. (c)(g) are histograms of the cosine similarity (in absolute value) between all pairs of features across different classes: for each pair, one sample is from the training dataset (including all shifts) and one sample is from another class in the test dataset (including all possible shifts). There are 4×10^6 pairs for the rotation (c) and 2.56×10^6 pairs for the translation (g).

To evaluate the learned representation, each training sample is augmented by 20 of its rotated version, each shifted with stride=10. We compute the cosine similarities among the $m \times 20$ augmented training inputs $\boldsymbol{X}_{\text{shift}}$ and the results are shown in Figure 16(a). We compare the cosine similarities among the learned features of all the augmented versions,

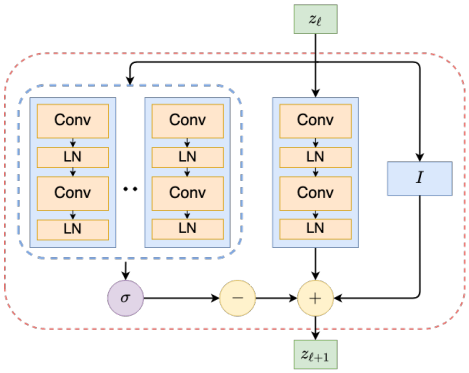
Initialization	Backpropagation	Test Accuracy
✓	✗	0.898
✗	✓	0.932
✓	✓	0.978

Table 2: Test accuracy of 2D translation-invariant ReduNet, ReduNet-bp (without initialization), and ReduNet-bp (with initialization) on the MNIST dataset.

i.e., $\bar{\boldsymbol{Z}}_{\text{shift}}$ and summarize the results in Figure 16(b). As we see, the so constructed rotation-invariant ReduNet is able to map the training data (as well as all its rotated versions) from the 10 different classes into 10 nearly orthogonal subspaces. That is, the learnt subspaces are truly invariant to shift transformation in polar angle. Next, we randomly draw another 100 test samples followed by the same augmentation procedure. We compute the cosine similarity histogram between features of all shifted training and those of the new test samples in Figure 16(c). In Figure 16(d), we visualize the MCR^2 loss on the ℓ-th layer representation of the ReduNet on the training and test dataset. From Figure 16(c) and Figure 16(d), we can find that the constructed ReduNet is indeed able to maximize the MCR^2 loss as well as generalize to the test data.

2D Translation Invariance on MNIST Digits. In this part, we provide experimental results to use the invariant ReduNet to learn representations for images that are invariant to the 2D cyclic translation. Essentially we view the image as painted on a torus and can be translated arbitrarily, as illustrated in Figure 23 in the Appendix. Again we use the 10-class MNIST dataset. We use $m = 100$ training samples (10 samples from each class) for constructing the ReduNet and use another 100 samples (10 samples from each class) as the test dataset. We apply 2D circulant convolution to the (1-channel) inputs with 75 random Gaussian kernels of size 9×9 before the first layer of ReduNet. For the translation-invariant ReduNet, we set $L = 25$, step size $\eta = 0.5$, precision $\epsilon = 0.1$. Similar to the rotational invariance task, to evaluate the performance of ReduNet with regard to translation, we augment each training/testing sample by 16 of its translational shifted version (with stride=7). The heatmap of the similarities among the $m \times 16$ augmented training inputs $\boldsymbol{X}_{\text{shift}}$ and the learned features $\mathsf{circ}(\bar{\boldsymbol{Z}})_{\text{shift}}$ are compared in Figure 16(e) and Figure 16(f). Similar to the rotation case, Figure 16(g) shows the histogram of similarity between features of all shifted training samples and test samples. Clearly, the ReduNet can map training samples from the 10 classes to 10 nearly orthogonal subspaces and invariant to all possible 2D translations on the training dataset. Also, the MCR^2 loss in Figure 16(h) is increasing with the increased layer. This verifies that the proposed ReduNet can indeed maximize the objective and be invariant to transformations as it is designed to.

Back Propagation on ReduNet. Once the $\{\boldsymbol{E}_\ell\}_{\ell=1}^L$ and $\{\boldsymbol{C}_\ell^1, \ldots, \boldsymbol{C}_\ell^k\}_{\ell=1}^L$ of the ReduNet are constructed, we here further test whether the architecture is amenable to fine-tuning via back propagation. For the 2D translation-invariant ReduNet, we add a fully-connected layer with weight having dimensions $CHW \times k$ after the last layer of the ReduNet, where k is the number of classes. We denote this modified architecture as *ReduNet-bp*. We consider the 10-class classification problem on the MNIST dataset, and compare three



ReLU	Train Acc	Test Acc
✓	0.9997	0.8327
✗	0.9970	0.6542

Figure 17: (**Left**) *ReduNet-inspired architecture*: Network architecture of convolutional ReduLayer, where each linear operator $\boldsymbol{E}_\ell$ or $\boldsymbol{C}_\ell^j$ is replaced by convolutions and layer normalization. (**Right**) Classification performance of ReduNet-inspired architecture on CIFAR10. ReLU indicates whether the (first layer of) the deep network architecture contains nonlinear ReLU activation or not.

networks: (1). ReduNet by the forward construction, (2). ReduNet-bp initialized by the construction, (3). ReduNet-bp with random initialization using the same backbone architecture. To initialize the networks, we use $m = 500$ samples (50 from each class) and set number of layers/iterations $L = 30$, step size $\eta = 0.5$, precision $\epsilon = 0.1$. Before the first layer, we perform 2D circulant-convolution to the inputs with 16 channels with 7×7 random Gaussian kernels. For ReduNet-bp, we further train the constructed ReduNet via back propagation by adding an extra fully connected layer and using cross-entropy loss. We update the model parameters by using SGD to minimize the loss on the entire MNIST training data. To evaluate the three networks, we compare the standard test accuracy on $10,000$ MNIST test samples. For ReduNet, we apply the nearest subspace classifier for prediction. For ReduNet-bp with and without initialization, we take the argmax of the final fully connected layer as the prediction. The results are summarized in Table 2. We find that (1). The ReduNet architecture *can* be optimized by SGD and achieve better standard accuracy after back propagation; (2). Using constructed ReduNet for initialization can achieve better performance compared with the same architecture with random initialization.

ReduNet-inspired Architecture and Back Propagation on CIFAR10. So far, our ReduNet architecture is constructed in a forward manner using a fixed number of training samples. In order to scale up to larger datasets and leverage backpropgation for improving model performance, we introduce a *ReduNet-inspired* network by slightly modifying the ReduNet meta-structure. Specifically, we replace $\boldsymbol{E}_\ell$ and $\boldsymbol{C}_\ell^j$ by concatenation of two convolution operators (`ksize=3`, `stride=1`, `padding=1`), and we apply the LayerNorm (Ba et al., 2016) after each convolution layer. The network architecture, a *ReduLayer block*, is presented in Figure 17 (**Left**). We follow a similar structure as ResNet18 by replacing the residual blocks with a trainable *ReduLayer block* of the same input/output dimension. The resulting network includes a convolution layer followed by an optional single ReLU nonlinearity, four ReduLayer blocks, and a linear layer. For each ReduLayer, we fix step

size $\eta = 1.0$ and $\lambda = 1.0$. We use 50,000 CIFAR10 training samples for training the network and 10,000 testing samples for evaluation. We train the network using cross-entropy loss and SGD with batch size of 256, learning rate `lr=0.001`, momentum `mom=0.9` and weight decay `wd=5e-4`. For optimization purposes, we also use the Cosine Annealing learning rate scheduler and a gradual learning rate warmup (Goyal et al., 2017) for the first 100 epochs. The classification performance of such networks is summarized in Figure 17 (Right). Surprisingly, we found that our ReduNet-inspired deep network is able to achieve decent performance on CIFAR10, including training ∼100% training accuracy and ∼65% test accuracy, even without the ReLU nonlinearity. The only nonlinear operators are softmax functions in ReduLayer. This indicates the expressive power of our proposed ReduNet-inspired deep networks.

6. Conclusions and Discussions

In this paper, we have laid out a theoretical and computational framework based on data compression which allows us to understand and interpret not only the characteristics of modern deep networks but also reveal their purposes and functions as a white box. From this new perspective, we see that at a high level, the objective of seeking a linear discriminative representation (via deep learning) aligns well with the objectives of the classic *linear discriminant analysis* (Hastie et al., 2009), *independent component analysis* (Hyvärinen and Oja, 2000), and *generalized principal component analysis* (Vidal et al., 2016). The main difference is that now we are able to conduct such analyses through the lens of a constructive nonlinear mapping and an intrinsic measure. This renders all these analyses so much more general hence practical for real-world data.

6.1 Comparison with Existing Practice of Deep Networks

Despite the long history of practicing artificial (deep) neural networks since their inception in 1940-1950's (McCulloch and Pitts, 1943; Rosenblatt, 1958), the architectures and operators in deep networks have been mainly proposed or designed empirically and trained via back propagation as a black box (Rumelhart et al., 1986). Table 3 left column summarizes the main characteristics of the conventional practice of deep networks,[35] whereas the right column highlights comparison of our new compression based framework to the current practice of deep neural networks, on which we will elaborate a little more below.

White-box versus black-box architectures. This new framework offers a constructive approach to derive deep (convolution) networks entirely as a white box from a principled objective of learning a low-dimensional linear discriminative representation for the given (mixed) data. The goodness of the representation is measured by the intrinsic rate reduction. The resulting network, called the ReduNet, emulates a gradient-based iterative scheme to optimize the rate reduction objective. It shares almost all the main structural characteristics of modern deep networks. Nevertheless, its architectures, linear and nonlinear operators and even their values are all derived from the data and all have precise geometric and statistical interpretation. In particular, we find it is rather intriguing that the linear operator

35. There are exceptions such as ScatteringNets (Bruna and Mallat, 2013; Wiatowski and Bölcskei, 2018), whose operators are pre-designed and fixed, as also indicated in the table.

	Conventional DNNs	Compression (ReduNets)
Objectives	input/output fitting	rate reduction
Deep architectures	trial & error	iterative optimization
Layer operators	empirical	projected gradient
Shift invariance	CNNs + augmentation	invariant ReduNets
Initializations	random/pre-designed	forward computed
Training/fine-tuning	back prop/fixed	forward/back prop
Interpretability	black box	white box
Representations	hidden or latent	incoherent subspaces (LDR)

Table 3: Comparison between conventional deep networks and compression based ReduNets.

of each layer has a non-parameteric "data auto-regression" interpretation. Together with the "forward construction," they give rather basic but universal computing mechanisms that even simple organisms/systems can use to learn good representations from the data that help future classification tasks (e.g. object detection or recognition).

Forward versus backward optimization and refinement. This constructive and white-box approach has demonstrated potential in liberating the practice of deep networks from relying (almost entirely) on empirically designed architectures, random initialization, and back propagation of all network parameters. It offers effective mechanisms to construct (hence initialize) deep networks in a forward fashion. The forward-constructed ReduNet already exhibits descent classification performance. Preliminary experiments given in this paper indicate that the ReduNet architecture is amenable to fine-tuning via back propagation too (say with new training data), and initialization with the forward-constructed network has advantages over random initialization. Furthermore, the constructive nature of ReduNet makes it amenable to other fine-tuning schemes such as forward propagation or incremental learning. Since one no longer has to update all network parameters simultaneously as a black box, one can effectively avoid the so-called "catastrophic forgetting" (McCloskey and Cohen, 1989) in the sequential or *incremental learning* setting, as the recent works of Wu et al. (2021); Tong et al. (2022) suggest.

Modeling invariance and equivariance. In the case of seeking classification invariant to certain transformation groups (say translation), the new framework models both eqvuivariance and invariance in a very natural way: all equivariant instances are mapped into the same subspace and the resulting subspaces are hence invariant to the transformation. As we have shown, in this case, the ReduNet naturally becomes a deep convolution network. Arguably, this work gives a new constructive and explicit justification for the role of multi-channel convolutions in each layer (widely adopted in modern CNNs) as incremental operators to compress or expand all equivariant instances for learning an LDR for the data. Moreover, our derivation reveals the fundamental computational advantage in constructing and learning such multi-channel convolutions in the spectral domain. Simulations and experiments on

synthetic and real data sets clearly verify such forward-constructed ReduNet can be invariant for *all* transformed instances. The computation scales gracefully with the number of classes, channels and sample dimension/size. In particular, the recent work of Baek et al. (2022) indicates that the $\log\det(\cdot)$ terms in the rate reduction objective can be computed much more efficiently through their variational forms.

Invariance and sparsity. The new framework also reveals a *fundamental trade-off* between sparsity and invariance: essentially one cannot expect to separate different classes of signals/data if signals in each class can be both arbitrarily shifted and arbitrarily superimposed. To achieve invariance to all translation, one must impose that signals in each class are sparsely generated so that all shifted samples span a proper submanifold (in the high-dimensional space) and features can be mapped to a proper subspace. Although sparse representation for individual signals have been extensively studied and well understood in the literature (Wright and Ma, 2021), very little is yet known about how to characterize the distribution of sparse codes of a class of (equivariant or locally equivariant) signals and its separability from other classes. This fundamental trade-off between sparsity and invariance certainly merits further theoretical study, as it will lead to more precise characterization of the statistical resource (network width) and computational resource (network depth) needed to provide performance guarantees, say for (multi-manifold) classification (Buchanan et al., 2021).

6.2 Further Improvements and Extensions

We believe the proposed rate reduction provides a principled framework for designing new networks with interpretable architectures and operators that can provide performance guarantees (say invariance) when applied to real-world datasets and problems. Nevertheless, the purposes of this paper are to introduce the basic principles and concepts of this new framework.

Improvements on ReduNet-inspired architectures. In this work, we have chosen arguably the simplest and most basic gradient-based scheme to construct the ReduNet for optimizing the rate reduction. As we have touched upon briefly in the paper, many powerful ideas from optimization can be further applied to improve the efficiency and performance of the network, such as acceleration, precondition/normalization, and regularization. Also there are additional relationships and structures among the channel operators $\boldsymbol{E}$ and $\boldsymbol{C}^j$ that have not been exploited in this work for computational efficiency. The reader may have realized that the basic ReduNet is expected to work when the data have relatively benign nonlinear structures when each class is close to be a linear subspace or Gaussian distribution. Real data (say image classes) have much more complicated structures: each class can have highly nonlinear geometry and topology or even be multi-modal itself. Hence in practice, to learn a better LDR, one may have to resort to more sophisticated strategies to control the compression (linearization) and expansion process. Real data also have additional priors and data structures that can be exploited. For example, one can reduce the dimension of the ambient feature space whenever the intrinsic dimension of the features are low (or sparse) enough. Such dimension-reduction operations (e.g. pooling or striding) are widely practiced in modern deep (convolution) networks for both computational efficiency and even accuracy.

According to the theory of compressive sensing, even a random projection would be rather efficient and effective in preserving the (discriminative) low-dimensional structures (Wright and Ma, 2021).

Rate reduction for structured autoencoding. In this work, we have only considered learning a one-sided embedding of the data via maximizing rate reduction: $\max_\theta \Delta R(\boldsymbol{Z}(\theta))$: $\boldsymbol{X} \xrightarrow{f(\boldsymbol{x},\theta)} \boldsymbol{Z}(\theta)$. As we have contended earlier in the problem formulation, the objective of LDR is to learn the intrinsic distribution of the data $\boldsymbol{X}$. Hence, the learned representation $\boldsymbol{Z}$ can naturally be used to generate the original data $\boldsymbol{X}$, say through a learned generator $g : \boldsymbol{z} \to \boldsymbol{x}$, and leads to an autoencoding:

$$\boldsymbol{X} \xrightarrow{f(\boldsymbol{x},\theta)} \boldsymbol{Z}(\theta) \xrightarrow{g(\boldsymbol{z},\eta)} \hat{\boldsymbol{X}}. \tag{53}$$

The rate reduction measure provides a *closed-form* distance for distributions that are mixtures of subspace-like Gaussians. Other distribution distances (normally used in learning generative models) such as KL-divergence, Jensen-Shannon divergence, and the Wasserstein distances, do not have a closed form even for (mixtures of degenerate) Guassians and are intractable, if not impossible, to compute in high-dimensional spaces. Nevertheless, the recent works of Dai et al. (2021); Tong et al. (2022) suggest that rate reduction can be an efficient and effective choice for learning a structured autoencoding (53) where the learned $\boldsymbol{Z}$ is an LDR.

Rate reduction for unsupervised learning. In this work, we have mainly considered learning a good representation $\boldsymbol{Z}$ for the data $\boldsymbol{X}$ when the class label $\boldsymbol{\Pi}$ is given and fixed. Nevertheless, notice that in its most general form, the maximal rate reduction objective can be optimized against both the representation $\boldsymbol{Z}$ and the membership $\boldsymbol{\Pi}$. In fact, the original work of Ma et al. (2007) precisely studies the complementary problem of learning $\boldsymbol{\Pi}$ by maximizing $\Delta R(\boldsymbol{Z}, \boldsymbol{\Pi}, \epsilon)$ with $\boldsymbol{Z}$ fixed, hence equivalent to minimizing only the compression term: $\min_{\boldsymbol{\Pi}} R_c(\boldsymbol{Z}, \epsilon \mid \boldsymbol{\Pi})$. Therefore, it is obvious that this framework can be naturally extended to *unsupervised* settings if the membership $\boldsymbol{\Pi}$ is partially known or entirely unknown and it is to be optimized together with the representation $\boldsymbol{Z}$. The recent work of Li et al. (2022) has demonstrated that rate reduction can indeed be a very effective objective for unsupervised learning. In a similar vein to the construction of the ReduNet in the supervised setting, the unsupervised learning may entail us to examine the joint dynamics of the gradient of the representation $\boldsymbol{Z}$ and the membership $\boldsymbol{\Pi}$:

$$\dot{\boldsymbol{Z}} = \eta \cdot \frac{\partial \Delta R}{\partial \boldsymbol{Z}}, \quad \dot{\boldsymbol{\Pi}} = \gamma \cdot \frac{\partial \Delta R}{\partial \boldsymbol{\Pi}}. \tag{54}$$

Data with other structures. Last but not the least, in this work, we only considered data that are naturally embedded (as submanifolds) in a vector space (real or complex). There have been many work that study and apply deep networks to data with additional structures or in a non-Euclidean space. For example, in reinforcement learning and optimal control, people often use deep networks to process data with additional dynamical structures, say linearizing the dynamics (Lusch et al., 2018). In computer graphics or many other fields, people deal with data on a non-Euclidean domain such as a mesh or a graph (Bronstein et al., 2017). It remains interesting to see how the principles of *data compression* and *linear*

discriminative representation can be extended to help study or design principled white-box deep networks associated with dynamical or graphical data and problems.

Acknowledgments

Yi would like to thank professor Yann LeCun of New York University for a stimulating discussion in his office back in November 2019 when they contemplated a fundamental question: *what does or should a deep network try to optimize?* At the time, they both believed the low-dimensionality of the data (say sparsity) and discriminativeness of the representation (e.g. contrastive learning) have something to do with the answer. The conversation had inspired Yi to delve into this problem more deeply while self-isolated at home during the pandemic.

Yi would also like to thank Dr. Harry Shum who has had many hours of conversations with Yi about how to understand and interpret deep networks during the past couple of years. In particular, Harry suggested how to better visualize the representations learned by the rate reduction, including the results shown in Figure 13.

The authors would like to thank the anonymous reviewers for their constructive suggestions and comments. Yi acknowledges support from ONR grant N00014-20-1-2002 and the joint Simons Foundation-NSF DMS grant #2031899, as well as support from Berkeley FHL Vive Center for Enhanced Reality and Berkeley Center for Augmented Cognition. Chong and Yi acknowledge support from Tsinghua-Berkeley Shenzhen Institute (TBSI) Research Fund. Yaodong, Haozhi, and Yi acknowledge support from Berkeley AI Research (BAIR). Yaodong acknowledges support from the joint Simons Foundation-NSF DMS grant #2031899. Haozhi acknowledges support from ONR grant N00014-20-1-2002. John acknowledges support from NSF grants #1838061, #1740833, and #1733857. Ryan Chan acknowledges support from the National Science Foundation Graduate Research Fellowship Program under Grant No. DGE2139757. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the National Science Foundation.

Appendix A. Properties of the Rate Reduction Function

This section is organized as follows. We present background and preliminary results for the $\log\det(\cdot)$ function and the coding rate function in Section A.1. Then, Section A.2 and A.3 provide technical lemmas for bounding the coding rate and coding rate reduction functions, respectively. Finally, these lemmas are used to prove our main theoretical results about the properties of the rate reduction function. The main results (given informally as Theorem 1 in the main body) are stated formally in Section A.4 and a proof is given in Section A.5.

Notations Throughout this section, we use $\mathbb{S}^n_{++}$, $\mathbb{R}_+$ and $\mathbb{Z}_{++}$ to denote the set of symmetric positive definite matrices of size $n \times n$, nonnegative real numbers and positive integers, respectively.

A.1 Preliminaries

Properties of the $\log\det(\cdot)$ function.

Lemma 7 *The function $\log\det(\cdot) : \mathbb{S}^n_{++} \to \mathbb{R}$ is strictly concave. That is,*

$$\log\det((1-\beta)\boldsymbol{Z}_1 + \beta\boldsymbol{Z}_2)) \geq (1-\beta)\log\det(\boldsymbol{Z}_1) + \beta\log\det(\boldsymbol{Z}_2)$$

for any $\beta \in (0,1)$ and $\{\boldsymbol{Z}_1, \boldsymbol{Z}_2\} \subseteq \mathbb{S}^n_{++}$, with equality holds if and only if $\boldsymbol{Z}_1 = \boldsymbol{Z}_2$.

Proof *Consider an arbitrary line given by $\boldsymbol{Z} = \boldsymbol{Z}_0 + t\Delta\boldsymbol{Z}$ where $\boldsymbol{Z}_0$ and $\Delta\boldsymbol{Z} \neq \boldsymbol{0}$ are symmetric matrices of size $n \times n$. Let $f(t) \doteq \log\det(\boldsymbol{Z}_0 + t\Delta\boldsymbol{Z})$ be a function defined on an interval of values of t for which $\boldsymbol{Z}_0 + t\Delta\boldsymbol{Z} \in \mathbb{S}^n_{++}$. Following the same argument as in Boyd and Vandenberghe (2004), we may assume $\boldsymbol{Z}_0 \in \mathbb{S}^n_{++}$ and get*

$$f(t) = \log\det\boldsymbol{Z}_0 + \sum_{i=1}^{n}\log(1+t\lambda_i),$$

where $\{\lambda_i\}_{i=1}^n$ are eigenvalues of $\boldsymbol{Z}_0^{-\frac{1}{2}}\Delta\boldsymbol{Z}\boldsymbol{Z}_0^{-\frac{1}{2}}$. The second order derivative of $f(t)$ is given by

$$f''(t) = -\sum_{i=1}^{n}\frac{\lambda_i^2}{(1+t\lambda_i)^2} < 0.$$

Therefore, $f(t)$ is strictly concave along the line $\boldsymbol{Z} = \boldsymbol{Z}_0 + t\Delta\boldsymbol{Z}$. By definition, we conclude that $\log\det(\cdot)$ is strictly concave. ■

Properties of the coding rate function. The following properties, also known as the Sylvester's determinant theorem, for the coding rate function are known in the paper of Ma et al. (2007).

Lemma 8 (Commutative property) *For any $\boldsymbol{Z} \in \mathbb{R}^{n\times m}$ we have*

$$R(\boldsymbol{Z},\epsilon) \doteq \frac{1}{2}\log\det\left(\boldsymbol{I} + \frac{n}{m\epsilon^2}\boldsymbol{Z}\boldsymbol{Z}^*\right) = \frac{1}{2}\log\det\left(\boldsymbol{I} + \frac{n}{m\epsilon^2}\boldsymbol{Z}^*\boldsymbol{Z}\right).$$

Lemma 9 (Invariant property) *For any $\boldsymbol{Z} \in \mathbb{R}^{n\times m}$ and any orthogonal matrices $\boldsymbol{U} \in \mathbb{R}^{n\times n}$ and $\boldsymbol{V} \in \mathbb{R}^{m\times m}$ we have*

$$R(\boldsymbol{Z},\epsilon) = R(\boldsymbol{U}\boldsymbol{Z}\boldsymbol{V}^*,\epsilon).$$

A.2 Lower and Upper Bounds for Coding Rate

The following result provides an upper and a lower bound on the coding rate of $\boldsymbol{Z}$ as a function of the coding rate for its components $\{\boldsymbol{Z}^j\}_{j=1}^k$. The lower bound is tight when all the components $\{\boldsymbol{Z}^j\}_{j=1}^k$ have the same covariance (assuming that they have zero mean). The upper bound is tight when the components $\{\boldsymbol{Z}^j\}_{j=1}^k$ are pair-wise orthogonal.

Lemma 10 *For any $\{\boldsymbol{Z}^j \in \mathbb{R}^{n\times m_j}\}_{j=1}^k$ and any $\epsilon > 0$, let $\boldsymbol{Z} = [\boldsymbol{Z}^1, \cdots, \boldsymbol{Z}^k] \in \mathbb{R}^{n\times m}$ with $m = \sum_{j=1}^k m_j$. We have*

$$\begin{aligned}\sum_{j=1}^k \frac{m_j}{2}\log\det\Big(\boldsymbol{I} + \frac{n}{m_j\epsilon^2}\boldsymbol{Z}^j(\boldsymbol{Z}^j)^*\Big) &\le \frac{m}{2}\log\det\Big(\boldsymbol{I} + \frac{n}{m\epsilon^2}\boldsymbol{Z}\boldsymbol{Z}^*\Big)\\ &\le \sum_{j=1}^k \frac{m}{2}\log\det\Big(\boldsymbol{I} + \frac{n}{m\epsilon^2}\boldsymbol{Z}^j(\boldsymbol{Z}^j)^*\Big),\end{aligned} \tag{55}$$

where the first equality holds if and only if

$$\frac{\boldsymbol{Z}^1(\boldsymbol{Z}^1)^*}{m_1} = \frac{\boldsymbol{Z}^2(\boldsymbol{Z}^2)^*}{m_2} = \cdots = \frac{\boldsymbol{Z}^k(\boldsymbol{Z}^k)^*}{m_k},$$

and the second equality holds if and only if $(\boldsymbol{Z}^{j_1})^*\boldsymbol{Z}^{j_2} = \boldsymbol{0}$ for all $1 \le j_1 < j_2 \le k$.

Proof By Lemma 7, $\log\det(\cdot)$ is strictly concave. Therefore,

$$\log\det\Big(\sum_{j=1}^k \beta_j\boldsymbol{S}^j\Big) \ge \sum_{j=1}^k \beta_j\log\det(\boldsymbol{S}^j),\ \text{for all } \{\beta_j > 0\}_{j=1}^k, \sum_{j=1}^k \beta_j = 1 \text{ and } \{\boldsymbol{S}^j \in \mathbb{S}_{++}^n\}_{j=1}^k,$$

where equality holds if and only if $\boldsymbol{S}^1 = \boldsymbol{S}^2 = \cdots = \boldsymbol{S}^k$. Take $\beta_j = \frac{m_j}{m}$ and $\boldsymbol{S}^j = \boldsymbol{I} + \frac{n}{m_j\epsilon^2}\boldsymbol{Z}^j(\boldsymbol{Z}^j)^*$, we get

$$\frac{m}{2}\log\det\Big(\boldsymbol{I} + \frac{n}{m\epsilon^2}\boldsymbol{Z}\boldsymbol{Z}^*\Big) \ge \sum_{j=1}^k \frac{m_j}{2}\log\det\Big(\boldsymbol{I} + \frac{n}{m_j\epsilon^2}\boldsymbol{Z}^j(\boldsymbol{Z}^j)^*\Big),$$

with equality holds if and only if $\frac{\boldsymbol{Z}^1(\boldsymbol{Z}^1)^*}{m_1} = \cdots = \frac{\boldsymbol{Z}^k(\boldsymbol{Z}^k)^*}{m_k}$. This proves the lower bound in (55).

We now prove the upper bound. By the strict concavity of $\log\det(\cdot)$, we have

$$\log\det(\mathbf{Q}) \le \log\det(\boldsymbol{S}) + \langle\nabla\log\det(\boldsymbol{S}),\, \mathbf{Q} - \boldsymbol{S}\rangle,\ \text{for all } \{\mathbf{Q}, \boldsymbol{S}\} \subseteq \mathbb{S}_{++}^m,$$

where equality holds if and only if $\mathbf{Q} = \boldsymbol{S}$. Plugging in $\nabla\log\det(\boldsymbol{S}) = \boldsymbol{S}^{-1}$ (see e.g., Boyd and Vandenberghe (2004)) and $\boldsymbol{S}^{-1} = (\boldsymbol{S}^{-1})^*$ gives

$$\log\det(\mathbf{Q}) \le \log\det(\boldsymbol{S}) + \mathsf{tr}(\boldsymbol{S}^{-1}\mathbf{Q}) - m. \tag{56}$$

We now take

$$\boldsymbol{Q} = \boldsymbol{I} + \frac{n}{m\epsilon^2}\boldsymbol{Z}^*\boldsymbol{Z} = \boldsymbol{I} + \frac{n}{m\epsilon^2}\begin{bmatrix} (\boldsymbol{Z}^1)^*\boldsymbol{Z}^1 & (\boldsymbol{Z}^1)^*\boldsymbol{Z}^2 & \cdots & (\boldsymbol{Z}^1)^*\boldsymbol{Z}^k \\ (\boldsymbol{Z}^2)^*\boldsymbol{Z}^1 & (\boldsymbol{Z}^2)^*\boldsymbol{Z}^2 & \cdots & (\boldsymbol{Z}^2)^*\boldsymbol{Z}^2 \\ \vdots & \vdots & \ddots & \vdots \\ (\boldsymbol{Z}^k)^*\boldsymbol{Z}^1 & (\boldsymbol{Z}^k)^*\boldsymbol{Z}^2 & \cdots & (\boldsymbol{Z}^k)^*\boldsymbol{Z}^k \end{bmatrix}, \text{ and} \tag{57}$$

$$\boldsymbol{S} = \boldsymbol{I} + \frac{n}{m\epsilon^2}\begin{bmatrix} (\boldsymbol{Z}^1)^*\boldsymbol{Z}^1 & \boldsymbol{0} & \cdots & \boldsymbol{0} \\ \boldsymbol{0} & (\boldsymbol{Z}^2)^*\boldsymbol{Z}^2 & \cdots & \boldsymbol{0} \\ \vdots & \vdots & \ddots & \vdots \\ \boldsymbol{0} & \boldsymbol{0} & \cdots & (\boldsymbol{Z}^k)^*\boldsymbol{Z}^k \end{bmatrix}.$$

From the property of determinant for block diagonal matrix, we have

$$\log\det(\boldsymbol{S}) = \sum_{j=1}^{k} \log\det\left(\boldsymbol{I} + \frac{n}{m\epsilon^2}(\boldsymbol{Z}^j)^*\boldsymbol{Z}^j\right). \tag{58}$$

Also, note that

$$\begin{aligned} &\mathsf{tr}(\boldsymbol{S}^{-1}\boldsymbol{Q}) \\ &= \mathsf{tr}\begin{bmatrix} (\boldsymbol{I} + \frac{n}{m\epsilon^2}(\boldsymbol{Z}^1)^*\boldsymbol{Z}^1)^{-1}(\boldsymbol{I} + \frac{n}{m\epsilon^2}(\boldsymbol{Z}^1)^*\boldsymbol{Z}^1) & \cdots & (\boldsymbol{I} + \frac{n}{m\epsilon^2}(\boldsymbol{Z}^1)^*\boldsymbol{Z}^1)^{-1}(\boldsymbol{I} + \frac{n}{m\epsilon^2}(\boldsymbol{Z}^1)^*\boldsymbol{Z}^k) \\ \vdots & \ddots & \vdots \\ (\boldsymbol{I} + \frac{n}{m\epsilon^2}(\boldsymbol{Z}^k)^*\boldsymbol{Z}^k)^{-1}(\boldsymbol{I} + \frac{n}{m\epsilon^2}(\boldsymbol{Z}^k)^*\boldsymbol{Z}^1) & \cdots & (\boldsymbol{I} + \frac{n}{m\epsilon^2}(\boldsymbol{Z}^k)^*\boldsymbol{Z}^k)^{-1}(\boldsymbol{I} + \frac{n}{m\epsilon^2}(\boldsymbol{Z}^k)^*\boldsymbol{Z}^k) \end{bmatrix} \\ &= \mathsf{tr}\begin{bmatrix} \boldsymbol{I} & \cdots & \bigcirc \\ \vdots & \ddots & \vdots \\ \bigcirc & \cdots & \boldsymbol{I} \end{bmatrix} = m, \end{aligned} \tag{59}$$

where "$\bigcirc$" denotes nonzero quantities that are irrelevant for the purpose of computing the trace. Plugging (58) and (59) back in (56) gives

$$\frac{m}{2}\log\det\left(\boldsymbol{I} + \frac{n}{m\epsilon^2}\boldsymbol{Z}^*\boldsymbol{Z}\right) \le \sum_{j=1}^{k}\frac{m}{2}\log\det\left(\boldsymbol{I} + \frac{n}{m\epsilon^2}(\boldsymbol{Z}^j)^*\boldsymbol{Z}^j\right),$$

where the equality holds if and only if $\boldsymbol{Q} = \boldsymbol{S}$, which by the formulation in (57), holds if and only if $(\boldsymbol{Z}^{j_1})^*\boldsymbol{Z}^{j_2} = \boldsymbol{0}$ for all $1 \le j_1 < j_2 \le k$. Further using the result in Lemma 8 gives

$$\frac{m}{2}\log\det\left(\boldsymbol{I} + \frac{n}{m\epsilon^2}\boldsymbol{Z}\boldsymbol{Z}^*\right) \le \sum_{j=1}^{k}\frac{m}{2}\log\det\left(\boldsymbol{I} + \frac{n}{m\epsilon^2}\boldsymbol{Z}^j(\boldsymbol{Z}^j)^*\right),$$

which produces the upper bound in (55). ■

A.3 An Upper Bound on Coding Rate Reduction

We may now provide an upper bound on the coding rate reduction $\Delta R(\boldsymbol{Z}, \boldsymbol{\Pi}, \epsilon)$ (defined in (11)) in terms of its individual components $\{\boldsymbol{Z}^j\}_{j=1}^k$.

Lemma 11 *For any $\boldsymbol{Z} \in \mathbb{R}^{n\times m}, \boldsymbol{\Pi} \in \Omega$ and $\epsilon > 0$, let $\boldsymbol{Z}^j \in \mathbb{R}^{n\times m_j}$ be $\boldsymbol{Z}\boldsymbol{\Pi}^j$ with zero columns removed. We have*

$$\Delta R(\boldsymbol{Z}, \boldsymbol{\Pi}, \epsilon) \leq \sum_{j=1}^{k} \frac{1}{2m} \log \left(\frac{\det^m \left(\boldsymbol{I} + \frac{n}{m\epsilon^2} \boldsymbol{Z}^j (\boldsymbol{Z}^j)^* \right)}{\det^{m_j} \left(\boldsymbol{I} + \frac{n}{m_j \epsilon^2} \boldsymbol{Z}^j (\boldsymbol{Z}^j)^* \right)} \right), \tag{60}$$

with equality holds if and only if $(\boldsymbol{Z}^{j_1})^* \boldsymbol{Z}^{j_2} = \boldsymbol{0}$ for all $1 \leq j_1 < j_2 \leq k$.

Proof From the definition of $\Delta R(\boldsymbol{Z}, \boldsymbol{\Pi}, \epsilon)$ in Eq. (11), we have

$$\begin{aligned}
&\Delta R(\boldsymbol{Z}, \boldsymbol{\Pi}, \epsilon) \\
&= R(\boldsymbol{Z}, \epsilon) - R_c(\boldsymbol{Z}, \epsilon \mid \boldsymbol{\Pi}) \\
&= \frac{1}{2} \log \left(\det \left(\boldsymbol{I} + \frac{n}{m\epsilon^2} \boldsymbol{Z}\boldsymbol{Z}^* \right) \right) - \sum_{j=1}^{k} \left\{ \frac{\mathsf{tr}(\boldsymbol{\Pi}^j)}{2m} \log \left(\det \left(\boldsymbol{I} + n \frac{\boldsymbol{Z}\boldsymbol{\Pi}^j \boldsymbol{Z}^*}{\mathsf{tr}(\boldsymbol{\Pi}^j)\epsilon^2} \right) \right) \right\} \\
&= \frac{1}{2} \log \left(\det \left(\boldsymbol{I} + \frac{n}{m\epsilon^2} \boldsymbol{Z}\boldsymbol{Z}^* \right) \right) - \sum_{j=1}^{k} \left\{ \frac{m_j}{2m} \log \left(\det \left(\boldsymbol{I} + n \frac{\boldsymbol{Z}^j (\boldsymbol{Z}^j)^*}{m_j \epsilon^2} \right) \right) \right\} \\
&\leq \sum_{j=1}^{k} \frac{1}{2} \log \left(\det \left(\boldsymbol{I} + \frac{n}{m\epsilon^2} \boldsymbol{Z}^j (\boldsymbol{Z}^j)^* \right) \right) - \sum_{j=1}^{k} \left\{ \frac{m_j}{2m} \log \left(\det \left(\boldsymbol{I} + n \frac{\boldsymbol{Z}^j (\boldsymbol{Z}^j)^*}{m_j \epsilon^2} \right) \right) \right\} \\
&= \sum_{j=1}^{k} \frac{1}{2m} \log \left(\det^m \left(\boldsymbol{I} + \frac{n}{m\epsilon^2} \boldsymbol{Z}^j (\boldsymbol{Z}^j)^* \right) \right) - \sum_{j=1}^{k} \left\{ \frac{1}{2m} \log \left(\det^{m_j} \left(\boldsymbol{I} + n \frac{\boldsymbol{Z}^j (\boldsymbol{Z}^j)^*}{m_j \epsilon^2} \right) \right) \right\} \\
&= \sum_{j=1}^{k} \frac{1}{2m} \log \left(\frac{\det^m \left(\boldsymbol{I} + \frac{n}{m\epsilon^2} \boldsymbol{Z}^j (\boldsymbol{Z}^j)^* \right)}{\det^{m_j} \left(\boldsymbol{I} + \frac{n}{m_j \epsilon^2} \boldsymbol{Z}^j (\boldsymbol{Z}^j)^* \right)} \right),
\end{aligned}$$

where the inequality follows from the upper bound in Lemma 10, and that the equality holds if and only if $(\boldsymbol{Z}^{j_1})^* \boldsymbol{Z}^{j_2} = \boldsymbol{0}$ for all $1 \leq j_1 < j_2 \leq k$. ■

A.4 Main Results: Properties of Maximal Coding Rate Reduction

We now present our main theoretical results. The following theorem states that for any fixed encoding of the partition $\boldsymbol{\Pi}$, the coding rate reduction is maximized by data $\boldsymbol{Z}$ that is maximally discriminative between different classes and is diverse within each of the classes. This result holds provided that the sum of rank for different classes is small relative to the ambient dimension, and that ϵ is small.

Theorem 12 *Let $\boldsymbol{\Pi} = \{\boldsymbol{\Pi}^j \in \mathbb{R}^{m\times m}\}_{j=1}^k$ with $\{\boldsymbol{\Pi}^j \geq \boldsymbol{0}\}_{j=1}^k$ and $\boldsymbol{\Pi}_1 + \cdots + \boldsymbol{\Pi}_k = \boldsymbol{I}$ be a given set of diagonal matrices whose diagonal entries encode the membership of the m*

samples in the k classes. Given any $\epsilon > 0$, $n > 0$ and $\{n \geq d_j > 0\}_{j=1}^k$, consider the optimization problem

$$\begin{aligned} \boldsymbol{Z}_\star \in & \underset{\boldsymbol{Z} \in \mathbb{R}^{n \times m}}{\operatorname{argmax}} \Delta R(\boldsymbol{Z}, \boldsymbol{\Pi}, \epsilon) \\ & s.t.\ \|\boldsymbol{Z}\boldsymbol{\Pi}^j\|_F^2 = \mathit{tr}(\boldsymbol{\Pi}^j),\ \mathsf{rank}(\boldsymbol{Z}\boldsymbol{\Pi}^j) \leq d_j,\ \forall j \in \{1, \ldots, k\}. \end{aligned} \tag{61}$$

Under the conditions

- (Large ambient dimension) $n \geq \sum_{j=1}^k d_j$, *and*
- (High coding precision) $\epsilon^4 < \min_{j \in \{1,\ldots,k\}} \left\{ \frac{\mathit{tr}(\boldsymbol{\Pi}^j)}{m} \frac{n^2}{d_j^2} \right\}$,

the optimal solution $\boldsymbol{Z}_\star$ satisfies

- (Between-class discriminative) $(\boldsymbol{Z}_\star^{j_1})^* \boldsymbol{Z}_\star^{j_2} = \boldsymbol{0}$ *for all $1 \leq j_1 < j_2 \leq k$, i.e., $\boldsymbol{Z}_\star^{j_1}$ and $\boldsymbol{Z}_\star^{j_2}$ lie in orthogonal subspaces, and*
- (Within-class diverse) *For each $j \in \{1, \ldots, k\}$, the rank of $\boldsymbol{Z}_\star^j$ is equal to d_j and either all singular values of $\boldsymbol{Z}_\star^j$ are equal to $\frac{\mathit{tr}(\boldsymbol{\Pi}^j)}{d_j}$, or the $d_j - 1$ largest singular values of $\boldsymbol{Z}_\star^j$ are equal and have value larger than $\frac{\mathit{tr}(\boldsymbol{\Pi}^j)}{d_j}$,*

where $\boldsymbol{Z}_\star^j \in \mathbb{R}^{n \times \mathit{tr}(\boldsymbol{\Pi}^j)}$ denotes $\boldsymbol{Z}_\star \boldsymbol{\Pi}^j$ with zero columns removed.

A.5 Proof of Main Results

We start with presenting a lemma that will be used in the proof to Theorem 12.

Lemma 13 *Given any twice differentiable $f : \mathbb{R}_+ \to \mathbb{R}$, integer $r \in \mathbb{Z}_{++}$ and $c \in \mathbb{R}_+$, consider the optimization problem*

$$\begin{aligned} \max_{\boldsymbol{x}} & \sum_{p=1}^r f(x_p) \\ & s.t.\ \boldsymbol{x} = [x_1, \ldots, x_r] \in \mathbb{R}_+^r,\ x_1 \geq x_2 \geq \cdots \geq x_r,\ \text{and} \sum_{p=1}^r x_p = c. \end{aligned} \tag{62}$$

Let $\boldsymbol{x}_\star$ be an arbitrary global solution to (62). If the conditions

- $f'(0) < f'(x)$ *for all $x > 0$,*
- *There exists $x_T > 0$ such that $f'(x)$ is strictly increasing in $[0, x_T]$ and strictly decreasing in $[x_T, \infty)$,*
- $f''(\frac{c}{r}) < 0$ *(equivalently, $\frac{c}{r} > x_T$),*

are satisfied, then we have either

- $\boldsymbol{x}_\star = [\frac{c}{r}, \ldots, \frac{c}{r}]$, *or*

- $\boldsymbol{x}_\star = [x_H, \ldots, x_H, x_L]$ *for some* $x_H \in (\frac{c}{r}, \frac{c}{r-1})$ *and* $x_L > 0$.

Proof The result holds trivially if $r = 1$. Throughout the proof we consider the case where $r > 1$.

We consider the optimization problem with the inequality constraint $x_1 \geq \cdots \geq x_r$ in (62) removed:

$$\max_{\boldsymbol{x}=[x_1,\ldots,x_r]\in\mathbb{R}^r_+} \sum_{p=1}^r f(x_p) \quad \text{s.t.} \sum_{p=1}^r x_p = c. \tag{63}$$

We need to show that any global solution $\boldsymbol{x}_\star = [(x_1)_\star, \ldots, (x_r)_\star]$ to (63) is either $\boldsymbol{x}_\star = [\frac{c}{r}, \ldots, \frac{c}{r}]$ or $\boldsymbol{x}_\star = [x_H, \ldots, x_H, x_L] \cdot \boldsymbol{P}$ for some $x_H > \frac{c}{r}$, $x_L > 0$ and permutation matrix $\boldsymbol{P} \in \mathbb{R}^{r\times r}$. Let

$$\mathcal{L}(\boldsymbol{x}, \boldsymbol{\lambda}) = \sum_{p=1}^r f(x_p) - \lambda_0 \cdot \left(\sum_{p=1}^r x_p - c \right) - \sum_{p=1}^r \lambda_p x_p$$

be the Lagragian function for (63) where $\boldsymbol{\lambda} = [\lambda_0, \lambda_1, \ldots, \lambda_r]$ is the Lagragian multiplier. By the first order optimality conditions (i.e., the Karush–Kuhn–Tucker (KKT) conditions, see, e.g., (Nocedal and Wright, 2006, Theorem 12.1)), there exists $\boldsymbol{\lambda}_\star = [(\lambda_0)_\star, (\lambda_1)_\star, \ldots, (\lambda_r)_\star]$ such that

$$\sum_{p=1}^r (x_q)_\star = c, \tag{64}$$

$$(x_q)_\star \geq 0, \ \forall q \in \{1, \ldots, r\}, \tag{65}$$

$$(\lambda_q)_\star \geq 0, \ \forall q \in \{1, \ldots, r\}, \tag{66}$$

$$(\lambda_q)_\star \cdot (x_q)_\star = 0, \ \forall q \in \{1, \ldots, r\}, \quad \text{and} \tag{67}$$

$$[f'((x_1)_\star), \ldots, f'((x_r)_\star)] = [(\lambda_0)_\star, \ldots, (\lambda_0)_\star] + [(\lambda_1)_\star, \ldots, (\lambda_r)_\star]. \tag{68}$$

By using the KKT conditions, we first show that all entries of $\boldsymbol{x}_\star$ are strictly positive. To prove by contradiction, suppose that $\boldsymbol{x}_\star$ has r_0 nonzero entries and $r - r_0$ zero entries for some $1 \leq r_0 < r$. Note that $r_0 \geq 1$ since an all zero vector $\boldsymbol{x}_\star$ does not satisfy the equality constraint (64).

Without loss of generality, we may assume that $(x_p)_\star > 0$ for $p \leq r_0$ and $(x_p)_\star = 0$ otherwise. By (67), we have

$$(\lambda_1)_\star = \cdots = (\lambda_{r_0})_\star = 0.$$

Plugging it into (68), we get

$$f'((x_1)_\star) = \cdots = f'((x_{r_0})_\star) = (\lambda_0)_\star.$$

From (68) and noting that $x_{r_0+1} = 0$ we get

$$f'(0) = f'(x_{r_0+1}) = (\lambda_0)_\star + (\lambda_{r_0+1})_\star.$$

Finally, from (66), we have

$$(\lambda_{r_0+1})_\star \geq 0.$$

Combining the last three equations above gives $f'(0) - f'((x_1)_\star) \geq 0$, contradicting the assumption that $f'(0) < f'(x)$ for all $x > 0$. This shows that $r_0 = r$, i.e., all entries of $\boldsymbol{x}_\star$ are strictly positive. Using this fact and (67) gives

$$(\lambda_p)_\star = 0 \quad \text{for all } p \in \{1, \ldots, r\}.$$

Combining this with (68) gives

$$f'((x_1)_\star) = \cdots = f'((x_r)_\star) = (\lambda_0)_\star. \tag{69}$$

It follows from the fact that $f'(x)$ is strictly unimodal that

$$\exists\ x_H \geq x_L > 0 \ \text{ s.t. } \ \{(x_p)_\star\}_{p=1}^r \subseteq \{x_L, x_H\}. \tag{70}$$

That is, the set $\{(x_p)_\star\}_{p=1}^r$ may contain no more than two values. To see why this is true, suppose that there exists three distinct values for $\{(x_p)_\star\}_{p=1}^r$. Without loss of generality we may assume that $0 < (x_1)_\star < (x_2)_\star < (x_3)_\star$. If $(x_2)_\star \leq x_T$ (recall $x_T := \arg\max_{x \geq 0} f'(x)$), then by using the fact that $f'(x)$ is strictly increasing in $[0, x_T]$, we must have $f'((x_1)_\star) < f'((x_2)_\star)$ which contradicts (69). A similar contradiction is arrived by considering $f'((x_2)_\star)$ and $f'((x_3)_\star)$ for the case where $(x_2)_\star > x_T$.

There are two possible cases as a consequence of (70). First, if $x_L = x_H$, then we have $(x_1)_\star = \cdots = (x_r)_\star$. By further using (64) we get

$$(x_1)_\star = \cdots = (x_r)_\star = \frac{c}{r}.$$

It remains to consider the case where $x_L < x_H$. First, by the unimodality of $f'(x)$, we must have $x_L < x_T < x_H$, therefore

$$f''(x_L) > 0 \text{ and } f''(x_H) < 0. \tag{71}$$

Let $\ell := |\{p : x_p = x_L\}|$ be the number of entries of $\boldsymbol{x}_\star$ that are equal to x_L and $h := r - \ell$. We show that it is necessary to have $\ell = 1$ and $h = r - 1$. To prove by contradiction, assume that $\ell > 1$ and $h < r - 1$. Without loss of generality we may assume $\{(x_p)_\star = x_H\}_{p=1}^h$ and $\{(x_p)_\star = x_L\}_{p=h+1}^r$. By (71), we have

$$f''((x_p)_\star) > 0 \text{ for all } p > h.$$

In particular, by using $h < r - 1$ we have

$$f''((x_{r-1})_\star) > 0 \text{ and } f''((x_r)_\star) > 0. \tag{72}$$

On the other hand, by using the second order necessary conditions for constraint optimization (see, e.g., (Nocedal and Wright, 2006, Theorem 12.5)), the following result holds

$$\begin{aligned} & \boldsymbol{v}_\star \nabla_{\boldsymbol{x}\boldsymbol{x}} \mathcal{L}(\boldsymbol{x}_\star, \boldsymbol{\lambda}_\star) \boldsymbol{v} \leq 0, \quad \text{for all } \left\{ \boldsymbol{v} : \left\langle \nabla_{\boldsymbol{x}} \left(\sum_{p=1}^r (x_p)_\star - c \right), \boldsymbol{v} \right\rangle = 0 \right\} \\ \iff \quad & \sum_{p=1}^r f''((x_p)_\star) \cdot v_p^2 \leq 0, \quad \text{for all } \left\{ \boldsymbol{v} = [v_1, \ldots, v_r] : \sum_{p=1}^r v_p = 0 \right\}. \end{aligned} \tag{73}$$

Take $\boldsymbol{v}$ to be such that $v_1 = \cdots = v_{r-2} = 0$ and $v_{r-1} = -v_r \neq 0$. Plugging it into (73) gives

$$f''((x_{r-1})_\star) + f''((x_r)_\star) \leq 0,$$

which contradicts (72). Therefore, we may conclude that $\ell = 1$. That is, $\boldsymbol{x}_\star$ is given by

$$\boldsymbol{x}_\star = [x_H, \ldots, x_H, x_L], \text{ where } x_H > x_L > 0.$$

By using the condition in (64), we may further show that

$$\begin{aligned}
&(r-1)x_H + x_L = c \implies x_H = \frac{c}{r-1} - \frac{c}{x_L} < \frac{x_L}{r-1}, \\
&(r-1)x_H + x_L = c \implies (r-1)x_H + x_H > c \implies x_H > \frac{c}{r},
\end{aligned}$$

which completes our proof. ■

Proof [Proof of Theorem 12] Without loss of generality, let $\boldsymbol{Z}_\star = [\boldsymbol{Z}_\star^1, \ldots, \boldsymbol{Z}_\star^k]$ be the optimal solution of problem (61).

To show that $\boldsymbol{Z}_\star^j, j \in \{1, \ldots, k\}$ are pairwise orthogonal, suppose for the purpose of arriving at a contradiction that $(\boldsymbol{Z}_\star^{j_1})^* \boldsymbol{Z}_\star^{j_2} \neq \boldsymbol{0}$ for some $1 \leq j_1 < j_2 \leq k$. By using Lemma 11, the strict inequality in (60) holds for the optimal solution $\boldsymbol{Z}_\star$. That is,

$$\Delta R(\boldsymbol{Z}_\star, \boldsymbol{\Pi}, \epsilon) < \sum_{j=1}^k \frac{1}{2m} \log \left(\frac{\det^m \left(\boldsymbol{I} + \frac{n}{m\epsilon^2} \boldsymbol{Z}_\star^j (\boldsymbol{Z}_\star^j)^* \right)}{\det^{m_j} \left(\boldsymbol{I} + \frac{n}{m_j \epsilon^2} \boldsymbol{Z}_\star^j (\boldsymbol{Z}_\star^j)^* \right)} \right). \tag{74}$$

On the other hand, since $\sum_{j=1}^k d_j \leq n$, there exists $\{\boldsymbol{U}_\diamond^j \in \mathbb{R}^{n \times d_j}\}_{j=1}^k$ such that the columns of the matrix $[\boldsymbol{U}_\diamond^1, \ldots, \boldsymbol{U}_\diamond^k]$ are orthonormal. Denote $\boldsymbol{Z}_\star^j = \boldsymbol{U}_\star^j \boldsymbol{\Sigma}_\star^j (\boldsymbol{V}_\star^j)^*$ the compact SVD of $\boldsymbol{Z}_\star^j$, and let

$$\boldsymbol{Z}_\diamond = [\boldsymbol{Z}_\diamond^1, \ldots, \boldsymbol{Z}_\diamond^k], \quad \text{where } \boldsymbol{Z}_\diamond^j = \boldsymbol{U}_\diamond^j \boldsymbol{\Sigma}_\star^j (\boldsymbol{V}_\star^j)^*.$$

It follows that

$$(\boldsymbol{Z}_\diamond^{j_1})^* \boldsymbol{Z}_\diamond^{j_2} = \boldsymbol{V}_\star^{j_1} \boldsymbol{\Sigma}_\star^{j_1} (\boldsymbol{U}_\diamond^{j_1})^* \boldsymbol{U}_\diamond^{j_2} \boldsymbol{\Sigma}_\star^{j_2} (\boldsymbol{V}_\star^{j_2})^* = \boldsymbol{V}_\star^{j_1} \boldsymbol{\Sigma}_\star^{j_1} \boldsymbol{0} \boldsymbol{\Sigma}_\star^{j_2} (\boldsymbol{V}_\star^{j_2})^* = \boldsymbol{0} \quad \text{for all } 1 \leq j_1 < j_2 \leq k. \tag{75}$$

That is, the matrices $\boldsymbol{Z}_\diamond^1, \ldots, \boldsymbol{Z}_\diamond^k$ are pairwise orthogonal. Applying Lemma 11 for $\boldsymbol{Z}_\diamond$ gives

$$\begin{aligned}
\Delta R(\boldsymbol{Z}_\diamond, \boldsymbol{\Pi}, \epsilon) &= \sum_{j=1}^k \frac{1}{2m} \log \left(\frac{\det^m \left(\boldsymbol{I} + \frac{n}{m\epsilon^2} \boldsymbol{Z}_\diamond^j (\boldsymbol{Z}_\diamond^j)^* \right)}{\det^{m_j} \left(\boldsymbol{I} + \frac{n}{m_j \epsilon^2} \boldsymbol{Z}_\diamond^j (\boldsymbol{Z}_\diamond^j)^* \right)} \right) \\
&= \sum_{j=1}^k \frac{1}{2m} \log \left(\frac{\det^m \left(\boldsymbol{I} + \frac{n}{m\epsilon^2} \boldsymbol{Z}_\star^j (\boldsymbol{Z}_\star^j)^* \right)}{\det^{m_j} \left(\boldsymbol{I} + \frac{n}{m_j \epsilon^2} \boldsymbol{Z}_\star^j (\boldsymbol{Z}_\star^j)^* \right)} \right),
\end{aligned} \tag{76}$$

where the second equality follows from Lemma 9. Comparing (74) and (76) gives $\Delta R(\boldsymbol{Z}_\diamond, \boldsymbol{\Pi}, \epsilon) > \Delta R(\boldsymbol{Z}_\star, \boldsymbol{\Pi}, \epsilon)$, which contradicts the optimality of $\boldsymbol{Z}_\star$. Therefore, we must have

$$(\boldsymbol{Z}_\star^{j_1})^* \boldsymbol{Z}_\star^{j_2} = \boldsymbol{0} \text{ for all } 1 \leq j_1 < j_2 \leq k.$$

Moreover, from Lemma 9 we have

$$\Delta R(\boldsymbol{Z}_\star, \boldsymbol{\Pi}, \epsilon) = \sum_{j=1}^{k} \frac{1}{2m} \log \left(\frac{\det^m \left(\boldsymbol{I} + \frac{n}{m\epsilon^2} \boldsymbol{Z}_\star^j (\boldsymbol{Z}_\star^j)^* \right)}{\det^{m_j} \left(\boldsymbol{I} + \frac{n}{m_j \epsilon^2} \boldsymbol{Z}_\star^j (\boldsymbol{Z}_\star^j)^* \right)} \right). \tag{77}$$

We now prove the result concerning the singular values of $\boldsymbol{Z}_\star^j$. To start with, we claim that the following result holds:

$$\boldsymbol{Z}_\star^j \in \arg\max_{\boldsymbol{Z}^j} \ \log \left(\frac{\det^m \left(\boldsymbol{I} + \frac{n}{m\epsilon^2} \boldsymbol{Z}^j (\boldsymbol{Z}^j)^* \right)}{\det^{m_j} \left(\boldsymbol{I} + \frac{n}{m_j \epsilon^2} \boldsymbol{Z}^j (\boldsymbol{Z}^j)^* \right)} \right) \quad \text{s.t. } \|\boldsymbol{Z}^j\|_F^2 = m_j, \, \mathsf{rank}(\boldsymbol{Z}^j) \le d_j. \tag{78}$$

To see why (78) holds, suppose that there exists $\widetilde{\boldsymbol{Z}}^j$ such that $\|\widetilde{\boldsymbol{Z}}^j\|_F^2 = m_j$, $\mathsf{rank}(\widetilde{\boldsymbol{Z}}^j) \le d_j$ and

$$\log \left(\frac{\det^m \left(\boldsymbol{I} + \frac{n}{m\epsilon^2} \widetilde{\boldsymbol{Z}}^j (\widetilde{\boldsymbol{Z}}^j)^* \right)}{\det^{m_j} \left(\boldsymbol{I} + \frac{n}{m_j \epsilon^2} \widetilde{\boldsymbol{Z}}^j (\widetilde{\boldsymbol{Z}}^j)^* \right)} \right) > \log \left(\frac{\det^m \left(\boldsymbol{I} + \frac{n}{m\epsilon^2} \boldsymbol{Z}_\star^j (\boldsymbol{Z}_\star^j)^* \right)}{\det^{m_j} \left(\boldsymbol{I} + \frac{n}{m_j \epsilon^2} \boldsymbol{Z}_\star^j (\boldsymbol{Z}_\star^j)^* \right)} \right). \tag{79}$$

Denote $\widetilde{\boldsymbol{Z}}^j = \widetilde{\boldsymbol{U}}^j \widetilde{\boldsymbol{\Sigma}}^j (\widetilde{\boldsymbol{V}}^j)^*$ the compact SVD of $\widetilde{\boldsymbol{Z}}^j$ and let

$$\boldsymbol{Z}_\diamond = [\boldsymbol{Z}_\star^1, \ldots, \boldsymbol{Z}_\star^{j-1}, \boldsymbol{Z}_\diamond^j, \boldsymbol{Z}_\star^{j+1}, \ldots, \boldsymbol{Z}_\star^k], \ \text{ where } \boldsymbol{Z}_\diamond^j := \boldsymbol{U}_\star^j \widetilde{\boldsymbol{\Sigma}}^j (\widetilde{\boldsymbol{V}}^j)^*.$$

Note that $\|\boldsymbol{Z}_\diamond^j\|_F^2 = m_j$, $\mathsf{rank}(\boldsymbol{Z}_\diamond^j) \le d_j$ and $(\boldsymbol{Z}_\diamond^j)^* \boldsymbol{Z}_\star^{j'} = \boldsymbol{0}$ for all $j' \neq j$. It follows that $\boldsymbol{Z}_\diamond$ is a feasible solution to (61) and that the components of $\boldsymbol{Z}_\diamond$ are pairwise orthogonal. By using Lemma 11, Lemma 9 and (79) we have

$$\begin{aligned}
&\Delta R(\boldsymbol{Z}_\diamond, \boldsymbol{\Pi}, \epsilon) \\
&= \frac{1}{2m} \log \left(\frac{\det^m \left(\boldsymbol{I} + \frac{n}{m\epsilon^2} \boldsymbol{Z}_\diamond^j (\boldsymbol{Z}_\diamond^j)^* \right)}{\det^{m_j} \left(\boldsymbol{I} + \frac{n}{m_j \epsilon^2} \boldsymbol{Z}_\diamond^j (\boldsymbol{Z}_\diamond^j)^* \right)} \right) + \sum_{j' \neq j} \frac{1}{2m} \log \left(\frac{\det^m \left(\boldsymbol{I} + \frac{n}{m\epsilon^2} \boldsymbol{Z}_\star^{j'} (\boldsymbol{Z}_\star^{j'})^* \right)}{\det^{m_{j'}} \left(\boldsymbol{I} + \frac{n}{m_{j'} \epsilon^2} \boldsymbol{Z}_\star^{j'} (\boldsymbol{Z}_\star^{j'})^* \right)} \right) \\
&= \frac{1}{2m} \log \left(\frac{\det^m \left(\boldsymbol{I} + \frac{n}{m\epsilon^2} \widetilde{\boldsymbol{Z}}^j (\widetilde{\boldsymbol{Z}}^j)^* \right)}{\det^{m_j} \left(\boldsymbol{I} + \frac{n}{m_j \epsilon^2} \widetilde{\boldsymbol{Z}}^j (\widetilde{\boldsymbol{Z}}^j)^* \right)} \right) + \sum_{j' \neq j} \frac{1}{2m} \log \left(\frac{\det^m \left(\boldsymbol{I} + \frac{n}{m\epsilon^2} \boldsymbol{Z}_\star^{j'} (\boldsymbol{Z}_\star^{j'})^* \right)}{\det^{m_{j'}} \left(\boldsymbol{I} + \frac{n}{m_{j'} \epsilon^2} \boldsymbol{Z}_\star^{j'} (\boldsymbol{Z}_\star^{j'})^* \right)} \right) \\
&> \frac{1}{2m} \log \left(\frac{\det^m \left(\boldsymbol{I} + \frac{n}{m\epsilon^2} \boldsymbol{Z}_\star^j (\boldsymbol{Z}_\star^j)^* \right)}{\det^{m_j} \left(\boldsymbol{I} + \frac{n}{m_j \epsilon^2} \boldsymbol{Z}_\star^j (\boldsymbol{Z}_\star^j)^* \right)} \right) + \sum_{j' \neq j} \frac{1}{2m} \log \left(\frac{\det^m \left(\boldsymbol{I} + \frac{n}{m\epsilon^2} \boldsymbol{Z}_\star^{j'} (\boldsymbol{Z}_\star^{j'})^* \right)}{\det^{m_{j'}} \left(\boldsymbol{I} + \frac{n}{m_{j'} \epsilon^2} \boldsymbol{Z}_\star^{j'} (\boldsymbol{Z}_\star^{j'})^* \right)} \right) \\
&= \sum_{j=1}^{k} \frac{1}{2m} \log \left(\frac{\det^m \left(\boldsymbol{I} + \frac{n}{m\epsilon^2} \boldsymbol{Z}_\star^j (\boldsymbol{Z}_\star^j)^* \right)}{\det^{m_j} \left(\boldsymbol{I} + \frac{n}{m_j \epsilon^2} \boldsymbol{Z}_\star^j (\boldsymbol{Z}_\star^j)^* \right)} \right).
\end{aligned}$$

Combining it with (77) shows $\Delta R(\boldsymbol{Z}_\diamond, \boldsymbol{\Pi}, \epsilon) > \Delta R(\boldsymbol{Z}_\star, \boldsymbol{\Pi}, \epsilon)$, contradicting the optimality of $\boldsymbol{Z}_\star$. Therefore, the result in (78) holds.

Observe that the optimization problem in (78) depends on $\boldsymbol{Z}^j$ only through its singular values. That is, by letting $\boldsymbol{\sigma}_j := [\sigma_{1,j}, \ldots, \sigma_{\min(m_j,n),j}]$ be the singular values of $\boldsymbol{Z}^j$, we have

$$\log \left(\frac{\det^m \left(\boldsymbol{I} + \frac{n}{m\epsilon^2} \boldsymbol{Z}^j (\boldsymbol{Z}^j)^* \right)}{\det^{m_j} \left(\boldsymbol{I} + \frac{n}{m_j \epsilon^2} \boldsymbol{Z}^j (\boldsymbol{Z}^j)^* \right)} \right) = \sum_{p=1}^{\min\{m_j, n\}} \log \left(\frac{(1 + \frac{n}{m\epsilon^2} \sigma_{p,j}^2)^m}{(1 + \frac{n}{m_j \epsilon^2} \sigma_{p,j}^2)^{m_j}} \right),$$

also, we have

$$\|\boldsymbol{Z}^j\|_F^2 = \sum_{p=1}^{\min\{m_j,n\}} \sigma_{p,j}^2 \;\text{ and }\; \mathsf{rank}(\boldsymbol{Z}^j) = \|\boldsymbol{\sigma}_j\|_0.$$

Using these relations, (78) is equivalent to

$$\begin{aligned} \max_{\boldsymbol{\sigma}_j \in \mathbb{R}_+^{\min\{m_j,n\}}} & \sum_{p=1}^{\min\{m_j,n\}} \log\left(\frac{(1+\frac{n}{m\epsilon^2}\sigma_{p,j}^2)^m}{(1+\frac{n}{m_j\epsilon^2}\sigma_{p,j}^2)^{m_j}}\right) \\ \text{s.t.} & \sum_{p=1}^{\min\{m_j,n\}} \sigma_{p,j}^2 = m_j, \text{ and } \mathsf{rank}(\boldsymbol{Z}^j) = \|\boldsymbol{\sigma}_j\|_0 \end{aligned} \tag{80}$$

Let $(\boldsymbol{\sigma}_j)_\star = [(\sigma_{1,j})_\star, \ldots, (\sigma_{\min\{m_j,n\},j})_\star]$ be an optimal solution to (80). Without loss of generality we assume that the entries of $(\boldsymbol{\sigma}_j)_\star$ are sorted in descending order. It follows that

$$(\sigma_{p,j})_\star = 0 \;\text{ for all }\; p > d_j,$$

and

$$[(\sigma_{1,j})_\star, \ldots, (\sigma_{d_j,j})_\star] = \underset{\substack{[\sigma_{1,j},\ldots,\sigma_{d_j,j}]\in\mathbb{R}_+^{d_j} \\ \sigma_{1,j}\geq\cdots\geq\sigma_{d_j,j}}}{\operatorname{argmax}} \sum_{p=1}^{d_j} \log\left(\frac{(1+\frac{n}{m\epsilon^2}\sigma_{p,j}^2)^m}{(1+\frac{n}{m_j\epsilon^2}\sigma_{p,j}^2)^{m_j}}\right) \quad \text{s.t. } \sum_{p=1}^{d_j}\sigma_{p,j}^2 = m_j. \tag{81}$$

Then we define

$$f(x; n, \epsilon, m_j, m) = \log\left(\frac{(1+\frac{n}{m\epsilon^2}x)^m}{(1+\frac{n}{m_j\epsilon^2}x)^{m_j}}\right),$$

and rewrite (81) as

$$\max_{\substack{[x_1,\ldots,x_{d_j}]\in\mathbb{R}_+^{d_j} \\ x_1\geq\cdots\geq x_{d_j}}} \sum_{p=1}^{d_j} f(x_p; n, \epsilon, m_j, m) \quad \text{s.t. } \sum_{p=1}^{d_j} x_p = m_j. \tag{82}$$

We compute the first and second derivative for f with respect to x, which are given by

$$\begin{aligned} f'(x; n, \epsilon, m_j, m) &= \frac{n^2 x(m-m_j)}{(nx+m\epsilon^2)(nx+m_j\epsilon^2)}, \\ f''(x; n, \epsilon, m_j, m) &= \frac{n^2(m-m_j)(mm_j\epsilon^4 - n^2x^2)}{(nx+m\epsilon^2)^2(nx+m_j\epsilon^2)^2}. \end{aligned}$$

Note that

- $0 = f'(0) < f'(x)$ for all $x > 0$,
- $f'(x)$ is strictly increasing in $[0, x_T]$ and strictly decreasing in $[x_T, \infty)$, where $x_T = \epsilon^2\sqrt{\frac{m}{n}\frac{m_j}{n}}$, and

- by using the condition $\epsilon^4 < \frac{m_j}{m}\frac{n^2}{d_j^2}$, we have $f''(\frac{m_j}{d_j}) < 0$.

Therefore, we may apply Lemma 13 and conclude that the unique optimal solution to (82) is either

- $\boldsymbol{x}_\star = [\frac{m_j}{d_j}, \ldots, \frac{m_j}{d_j}]$, or
- $\boldsymbol{x}_\star = [x_H, \ldots, x_H, x_L]$ for some $x_H \in (\frac{m_j}{d_j}, \frac{m_j}{d_j - 1})$ and $x_L > 0$.

Equivalently, we have either

- $[(\sigma_{1,j})_\star, \ldots, (\sigma_{d_j,j})_\star] = \left[\sqrt{\frac{m_j}{d_j}}, \ldots, \sqrt{\frac{m_j}{d_j}}\right]$, or
- $[(\sigma_{1,j})_\star, \ldots, (\sigma_{d_j,j})_\star] = [\sigma_H, \ldots, \sigma_H, \sigma_L]$ for some $\sigma_H \in \left(\sqrt{\frac{m_j}{d_j}}, \sqrt{\frac{m_j}{d_j - 1}}\right)$ and $\sigma_L > 0$,

as claimed. ■

Appendix B. ReduNet for 1D Circular Shift Invariance

It has been long known that to implement a convolutional neural network, one can achieve higher computational efficiency by implementing the network in the spectral domain via the fast Fourier transform (Mathieu et al., 2013; Lavin and Gray, 2016; Vasilache et al., 2015). However, our purpose here is different: We want to show that the linear operators $\boldsymbol{E}$ and $\boldsymbol{C}^j$ (or $\bar{\boldsymbol{E}}$ and $\bar{\boldsymbol{C}}^j$) derived from the gradient flow of MCR^2 are naturally convolutions when we enforce shift-invariance rigorously. Their convolution structure is derived from the rate reduction objective, rather than imposed upon the network. Furthermore, the computation involved in constructing these linear operators has a naturally efficient implementation in the spectral domain via fast Fourier transform. Arguably this work is the first to show multi-channel convolutions, together with other convolution-preserving nonlinear operations in the ReduNet, are both necessary and sufficient to ensure shift invariance.

To be somewhat self-contained and self-consistent, in this section, we first introduce our notation and review some of the key properties of circulant matrices which will be used to characterize the properties of the linear operators $\bar{\boldsymbol{E}}$ and $\bar{\boldsymbol{C}}^j$ and to compute them efficiently. The reader may refer to Kra and Simanca (2012) for a more rigorous exposition on circulant matrices.

B.1 Properties of Circulant Matrix and Circular Convolution

Given a vector $\boldsymbol{z} = [z(0), z(1), \ldots, z(n-1)]^* \in \mathbb{R}^n$, we may arrange all its circular shifted versions in a circulant matrix form as

$$\mathsf{circ}(\boldsymbol{z}) \;\doteq\; \begin{bmatrix} z(0) & z(n-1) & \ldots & z(2) & z(1) \\ z(1) & z(0) & z(n-1) & \cdots & z(2) \\ \vdots & z(1) & z(0) & \ddots & \vdots \\ z(n-2) & \vdots & \ddots & \ddots & z(n-1) \\ z(n-1) & z(n-2) & \ldots & z(1) & z(0) \end{bmatrix} \in \mathbb{R}^{n\times n}. \tag{83}$$

Fact 1 (Convolution as matrix multiplication via circulant matrix) *The multiplication of a circulant matrix* $\mathsf{circ}(\boldsymbol{z})$ *with a vector* $\boldsymbol{x} \in \mathbb{R}^n$ *gives a circular (or cyclic) convolution, i.e.,*

$$\mathsf{circ}(\boldsymbol{z}) \cdot \boldsymbol{x} = \boldsymbol{z} \circledast \boldsymbol{x}, \tag{84}$$

where

$$(\boldsymbol{z} \circledast \boldsymbol{x})_i = \sum_{j=0}^{n-1} x(j) z(i + n - j \bmod n). \tag{85}$$

Fact 2 (Properties of circulant matrices) *Circulant matrices have the following properties:*

- *Transpose of a circulant matrix, say* $\mathsf{circ}(\boldsymbol{z})^*$*, is circulant;*
- *Multiplication of two circulant matrices is circulant, for example* $\mathsf{circ}(\boldsymbol{z})\mathsf{circ}(z)^*$*;*

- *For a non-singular circulant matrix, its inverse is also circulant (hence representing a circular convolution).*

These properties of circulant matrices are extensively used in this work as for characterizing the convolution structures of the operators $\boldsymbol{E}$ and $\boldsymbol{C}^j$. Given a set of vectors $[\boldsymbol{z}^1, \ldots, \boldsymbol{z}^m] \in \mathbb{R}^{n \times m}$, let $\mathsf{circ}(\boldsymbol{z}^i) \in \mathbb{R}^{n \times n}$ be the circulant matrix for $\boldsymbol{z}^i$. Then we have the following (Proposition 4 in the main body and here restated for convenience):

Proposition 14 (Convolution structures of $\boldsymbol{E}$ and $\boldsymbol{C}^j$) *Given a set of vectors $\boldsymbol{Z} = [\boldsymbol{z}^1, \ldots, \boldsymbol{z}^m]$, the matrix:*

$$\boldsymbol{E} = \alpha\big(\boldsymbol{I} + \alpha \sum_{i=1}^{m} \mathsf{circ}(\boldsymbol{z}^i)\mathsf{circ}(\boldsymbol{z}^i)^*\big)^{-1}$$

is a circulant matrix and represents a circular convolution:

$$\boldsymbol{E}\boldsymbol{z} = \boldsymbol{e} \circledast \boldsymbol{z},$$

where $\boldsymbol{e}$ is the first column vector of $\boldsymbol{E}$. Similarly, the matrices $\boldsymbol{C}^j$ associated with any subsets of $\boldsymbol{Z}$ are also circular convolutions.

B.2 Circulant Matrix and Circulant Convolution for Multi-channel Signals

In the remainder of this section, we view $\boldsymbol{z}$ as a 1D signal such as an audio signal. Since we will deal with the more general case of multi-channel signals, we will use the traditional notation T to denote the temporal length of the signal and C for the number of channels. Conceptually, the "dimension" n of such a multi-channel signal, if viewed as a vector, should be $n = CT$.[36] As we will also reveal additional interesting structures of the operators $\boldsymbol{E}$ and $\boldsymbol{C}^j$ in the spectral domain, we use t as the index for time, p for the index of frequency, and c for the index of channel.

Given a multi-channel 1D signal $\bar{\boldsymbol{z}} \in \mathbb{R}^{C \times T}$, we denote

$$\bar{\boldsymbol{z}} = \begin{bmatrix} \bar{\boldsymbol{z}}[1]^* \\ \vdots \\ \bar{\boldsymbol{z}}[C]^* \end{bmatrix} = [\bar{\boldsymbol{z}}(0), \bar{\boldsymbol{z}}(1), \ldots, \bar{\boldsymbol{z}}(T-1)] = \{\bar{\boldsymbol{z}}[c](t)\}_{c=1,t=0}^{c=C,t=T-1}. \tag{86}$$

To compute the coding rate reduction for a collection of such multi-channel 1D signals, we may flatten the matrix representation into a vector representation by stacking the multiple channels of $\bar{\boldsymbol{z}}$ as a column vector. In particular, we let

$$\mathsf{vec}(\bar{\boldsymbol{z}}) = [\bar{\boldsymbol{z}}[1](0), \bar{\boldsymbol{z}}[1](1), \ldots, \bar{\boldsymbol{z}}[1](T-1), \bar{\boldsymbol{z}}[2](0), \ldots] \quad \in \mathbb{R}^{(C \times T)}. \tag{87}$$

Furthermore, to obtain shift invariance for the coding rate reduction, we may generate a collection of shifted copies of $\bar{\boldsymbol{z}}$ (along the temporal dimension). Stacking the vector

36. Notice that in the main paper, for simplicity, we have used n to indicate both the 1D "temporal" or 2D "spatial" dimension of a signal, just to be consistent with the vector case, which corresponds to T here. All notation should be clear within the context.

representations for such shifted copies as column vectors, we obtain

$$\mathsf{circ}(\bar{\boldsymbol{z}}) \doteq \begin{bmatrix} \mathsf{circ}(\bar{\boldsymbol{z}}[1]) \\ \vdots \\ \mathsf{circ}(\bar{\boldsymbol{z}}[C]) \end{bmatrix} \quad \in \mathbb{R}^{(C\times T)\times T}. \tag{88}$$

In above, we overload the notation "$\mathsf{circ}(\cdot)$" defined in (83).

We now consider a collection of m multi-channel 1D signals $\{\bar{\boldsymbol{z}}^i \in \mathbb{R}^{C\times T}\}_{i=1}^m$. Compactly representing the data by $\bar{\boldsymbol{Z}} \in \mathbb{R}^{C\times T\times m}$ in which the i-th slice on the last dimension is $\bar{\boldsymbol{z}}^i$, we denote

$$\bar{\boldsymbol{Z}}[c] = [\bar{\boldsymbol{z}}^1[c], \ldots, \bar{\boldsymbol{z}}^m[c]] \in \mathbb{R}^{T\times m}, \qquad \bar{\boldsymbol{Z}}(t) = [\bar{\boldsymbol{z}}^1(t), \ldots, \bar{\boldsymbol{z}}^m(t)] \in \mathbb{R}^{C\times m}. \tag{89}$$

In addition, we denote

$$\begin{aligned} \mathsf{vec}(\bar{\boldsymbol{Z}}) &= [\mathsf{vec}(\bar{\boldsymbol{z}}^1), \ldots, \mathsf{vec}(\bar{\boldsymbol{z}}^m)] \in \mathbb{R}^{(C\times T)\times m}, \\ \mathsf{circ}(\bar{\boldsymbol{Z}}) &= [\mathsf{circ}(\bar{\boldsymbol{z}}^1), \ldots, \mathsf{circ}(\bar{\boldsymbol{z}}^m)] \in \mathbb{R}^{(C\times T)\times (T\times m)}. \end{aligned} \tag{90}$$

Then, we define the *shift invariant coding rate reduction* for $\bar{\boldsymbol{Z}} \in \mathbb{R}^{C\times T\times m}$ as

$$\begin{aligned} &\Delta R_{\mathsf{circ}}(\bar{\boldsymbol{Z}}, \boldsymbol{\Pi}) \doteq \frac{1}{T}\Delta R(\mathsf{circ}(\bar{\boldsymbol{Z}}), \bar{\boldsymbol{\Pi}}) \\ &= \frac{1}{2T}\log\det\left(\boldsymbol{I} + \alpha\cdot\mathsf{circ}(\bar{\boldsymbol{Z}})\cdot\mathsf{circ}(\bar{\boldsymbol{Z}})^*\right) - \sum_{j=1}^{k}\frac{\gamma_j}{2T}\log\det\left(\boldsymbol{I} + \alpha_j\cdot\mathsf{circ}(\bar{\boldsymbol{Z}})\cdot\bar{\boldsymbol{\Pi}}^j\cdot\mathsf{circ}(\bar{\boldsymbol{Z}})^*\right), \end{aligned} \tag{91}$$

where $\alpha = \frac{CT}{mT\epsilon^2} = \frac{C}{m\epsilon^2}$, $\alpha_j = \frac{CT}{\mathsf{tr}(\boldsymbol{\Pi}^j)T\epsilon^2} = \frac{C}{\mathsf{tr}(\boldsymbol{\Pi}^j)\epsilon^2}$, $\gamma_j = \frac{\mathsf{tr}(\boldsymbol{\Pi}^j)}{m}$, and $\bar{\boldsymbol{\Pi}}^j$ is augmented membership matrix in an obvious way. Note that we introduce the normalization factor T in (91) because the circulant matrix $\mathsf{circ}(\bar{\boldsymbol{Z}})$ contains T (shifted) copies of each signal.

By applying (14) and (15), we obtain the derivative of $\Delta R_{\mathsf{circ}}(\bar{\boldsymbol{Z}}, \boldsymbol{\Pi})$ as

$$\begin{aligned} \frac{1}{2T}\frac{\partial\log\det\left(\boldsymbol{I} + \alpha\mathsf{circ}(\bar{\boldsymbol{Z}})\mathsf{circ}(\bar{\boldsymbol{Z}})^*\right)}{\partial\mathsf{vec}(\bar{\boldsymbol{Z}})} &= \frac{1}{2T}\frac{\partial\log\det\left(\boldsymbol{I} + \alpha\mathsf{circ}(\bar{\boldsymbol{Z}})\mathsf{circ}(\bar{\boldsymbol{Z}})^*\right)}{\partial\mathsf{circ}(\bar{\boldsymbol{Z}})}\frac{\partial\mathsf{circ}(\bar{\boldsymbol{Z}})}{\partial\mathsf{vec}(\bar{\boldsymbol{Z}})} \\ &= \underbrace{\alpha\Big(\boldsymbol{I} + \alpha\mathsf{circ}(\bar{\boldsymbol{Z}})\mathsf{circ}(\bar{\boldsymbol{Z}})^*\Big)^{-1}}_{\bar{\boldsymbol{E}}\,\in\mathbb{R}^{(C\times T)\times(C\times T)}}\mathsf{vec}(\bar{\boldsymbol{Z}}), \end{aligned} \tag{92}$$

$$\frac{\gamma_j}{2T}\frac{\partial\log\det\left(\boldsymbol{I} + \alpha_j\mathsf{circ}(\bar{\boldsymbol{Z}})\boldsymbol{\Pi}^j\mathsf{circ}(\bar{\boldsymbol{Z}})^*\right)}{\partial\mathsf{vec}(\bar{\boldsymbol{Z}})} = \gamma_j\underbrace{\alpha_j\Big(\boldsymbol{I} + \alpha_j\mathsf{circ}(\bar{\boldsymbol{Z}})\boldsymbol{\Pi}^j\mathsf{circ}(\bar{\boldsymbol{Z}})^*\Big)^{-1}}_{\bar{\boldsymbol{C}}^j\,\in\mathbb{R}^{(C\times T)\times(C\times T)}}\mathsf{vec}(\bar{\boldsymbol{Z}})\boldsymbol{\Pi}^j. \tag{93}$$

In the following, we show that $\bar{\boldsymbol{E}}\cdot\mathsf{vec}(\bar{\boldsymbol{z}})$ represents a multi-channel circular convolution. Note that

$$\bar{\boldsymbol{E}} = \alpha\begin{bmatrix} \boldsymbol{I} + \alpha\sum_{i=1}^m \mathsf{circ}(\boldsymbol{z}^i[1])\mathsf{circ}(\boldsymbol{z}^i[1])^* & \cdots & \sum_{i=1}^m \mathsf{circ}(\boldsymbol{z}^i[1])\mathsf{circ}(\boldsymbol{z}^i[C])^* \\ \vdots & \ddots & \vdots \\ \sum_{i=1}^m \mathsf{circ}(\boldsymbol{z}^i[C])\mathsf{circ}(\boldsymbol{z}^i[1])^* & \cdots & \boldsymbol{I} + \sum_{i=1}^m \alpha\mathsf{circ}(\boldsymbol{z}^i[C])\mathsf{circ}(\boldsymbol{z}^i[C])^* \end{bmatrix}^{-1}. \tag{94}$$

By using Fact 2, the matrix in the inverse above is a *block circulant matrix*, i.e., a block matrix where each block is a circulant matrix. A useful fact about the inverse of such a matrix is the following.

Fact 3 (Inverse of block circulant matrices) *The inverse of a block circulant matrix is a block circulant matrix (with respect to the same block partition).*

The main result of this subsection is the following (Proposition 5 in the main body and here restated for convenience):

Proposition 15 (Convolution structures of $\bar{\boldsymbol{E}}$ and $\bar{\boldsymbol{C}}^j$) *Given a collection of multi-channel 1D signals $\{\bar{\boldsymbol{z}}^i \in \mathbb{R}^{C\times T}\}_{i=1}^m$, the matrix $\bar{\boldsymbol{E}}$ is a block circulant matrix, i.e.,*

$$\bar{\boldsymbol{E}} \doteq \begin{bmatrix} \bar{\boldsymbol{E}}_{1,1} & \cdots & \bar{\boldsymbol{E}}_{1,C} \\ \vdots & \ddots & \vdots \\ \bar{\boldsymbol{E}}_{C,1} & \cdots & \bar{\boldsymbol{E}}_{C,C} \end{bmatrix}, \tag{95}$$

where each $\bar{\boldsymbol{E}}_{c,c'} \in \mathbb{R}^{T\times T}$ is a circulant matrix. Moreover, $\bar{\boldsymbol{E}}$ represents a multi-channel circular convolution, i.e., for any multi-channel signal $\bar{\boldsymbol{z}} \in \mathbb{R}^{C\times T}$ we have

$$\bar{\boldsymbol{E}} \cdot \mathsf{vec}(\bar{\boldsymbol{z}}) = \mathsf{vec}(\bar{\boldsymbol{e}} \circledast \bar{\boldsymbol{z}}).$$

In above, $\bar{\boldsymbol{e}} \in \mathbb{R}^{C\times C\times T}$ is a multi-channel convolutional kernel with $\bar{\boldsymbol{e}}[c,c'] \in \mathbb{R}^T$ being the first column vector of $\bar{\boldsymbol{E}}_{c,c'}$, and $\bar{\boldsymbol{e}} \circledast \bar{\boldsymbol{z}} \in \mathbb{R}^{C\times T}$ is the multi-channel circular convolution (with "$\circledast$" overloading the notation from Eq. (85)) defined as

$$(\bar{\boldsymbol{e}} \circledast \bar{\boldsymbol{z}})[c] = \sum_{c'=1}^{C} \bar{\boldsymbol{e}}[c,c'] \circledast \bar{\boldsymbol{z}}[c'], \quad \forall c = 1,\ldots,C. \tag{96}$$

Similarly, the matrices $\bar{\boldsymbol{C}}^j$ associated with any subsets of $\bar{\boldsymbol{Z}}$ are also multi-channel circular convolutions.

Note that the calculation of $\bar{\boldsymbol{E}}$ in (94) requires inverting a matrix of size $(C\times T)\times(C\times T)$. In the following, we show that this computation can be accelerated by working in the frequency domain.

B.3 Fast Computation in Spectral Domain

Circulant matrix and Discrete Fourier Transform. A remarkable property of circulant matrices is that *they all share the same set of eigenvectors that form a unitary matrix.* We define the matrix:

$$\boldsymbol{F}_T \doteq \frac{1}{\sqrt{T}} \begin{bmatrix} \omega_T^0 & \omega_T^0 & \cdots & \omega_T^0 & \omega_T^0 \\ \omega_T^0 & \omega_T^1 & \cdots & \omega_T^{T-2} & \omega_T^{T-1} \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ \omega_T^0 & \omega_T^{T-2} & \cdots & \omega_T^{(T-2)^2} & \omega_T^{(T-2)(T-1)} \\ \omega_T^0 & \omega_T^{T-1} & \cdots & \omega_T^{(T-2)(T-1)} & \omega_T^{(T-1)^2} \end{bmatrix} \in \mathbb{C}^{T\times T}, \tag{97}$$

where $\omega_T \doteq \exp(-\frac{2\pi\sqrt{-1}}{T})$ is the roots of unit (as $\omega_T^T = 1$). The matrix $\boldsymbol{F}_T$ is a unitary matrix: $\boldsymbol{F}_T\boldsymbol{F}_T^* = \boldsymbol{I}$ and is the well known *Vandermonde matrix*. Multiplying a vector with $\boldsymbol{F}_T$ is known as the *discrete Fourier transform* (DFT). Be aware that the conventional DFT matrix differs from our definition of $\boldsymbol{F}_T$ here by a scale: it does not have the $\frac{1}{\sqrt{T}}$ in front. Here for simplicity, we scale it so that $\boldsymbol{F}_T$ is a unitary matrix and its inverse is simply its conjugate transpose $\boldsymbol{F}_T^*$, columns of which represent the eigenvectors of a circulant matrix (Abidi et al., 2016).

Fact 4 (DFT as matrix-vector multiplication) *The DFT of a vector $\boldsymbol{z} \in \mathbb{R}^T$ can be computed as*

$$\mathrm{DFT}(\boldsymbol{z}) \doteq \boldsymbol{F}_T \cdot \boldsymbol{z} \quad \in \mathbb{C}^T, \tag{98}$$

where

$$\mathrm{DFT}(\boldsymbol{z})(p) = \frac{1}{\sqrt{T}} \sum_{t=0}^{T-1} z(t) \cdot \omega_T^{p \cdot t}, \quad \forall p = 0, 1, \ldots, T-1. \tag{99}$$

The Inverse Discrete Fourier Transform (IDFT) of a signal $\boldsymbol{v} \in \mathbb{C}^T$ can be computed as

$$\mathrm{IDFT}(\boldsymbol{v}) \doteq \boldsymbol{F}_T^* \cdot \boldsymbol{v} \quad \in \mathbb{C}^T \tag{100}$$

where

$$\mathrm{IDFT}(\boldsymbol{v})(t) = \frac{1}{\sqrt{T}} \sum_{p=0}^{T-1} v(p) \cdot \omega_T^{-p \cdot t}, \quad \forall t = 0, 1, \ldots, T-1. \tag{101}$$

Regarding the relationship between a circulant matrix (convolution) and discrete Fourier transform, we have:

Fact 5 *An $n \times n$ matrix $\boldsymbol{M} \in \mathbb{C}^{n \times n}$ is a circulant matrix if and only if it is diagonalizable by the unitary matrix $\boldsymbol{F}_n$:*

$$\boldsymbol{F}_n \boldsymbol{M} \boldsymbol{F}_n^* = \boldsymbol{D} \quad \textit{or} \quad \boldsymbol{M} = \boldsymbol{F}_n^* \boldsymbol{D} \boldsymbol{F}_n, \tag{102}$$

where $\boldsymbol{D}$ is a diagonal matrix of eigenvalues.

Fact 6 (DFT are eigenvalues of the circulant matrix) *Given a vector $\boldsymbol{z} \in \mathbb{C}^T$, we have*

$$\boldsymbol{F}_T \cdot \mathsf{circ}(\boldsymbol{z}) \cdot \boldsymbol{F}_T^* = \mathsf{diag}(\mathrm{DFT}(\boldsymbol{z})) \quad \textit{or} \quad \mathsf{circ}(\boldsymbol{z}) = \boldsymbol{F}_T^* \cdot \mathsf{diag}(\mathrm{DFT}(\boldsymbol{z})) \cdot \boldsymbol{F}_T. \tag{103}$$

That is, the eigenvalues of the circulant matrix associated with a vector are given by its DFT.

Fact 7 (Parseval's theorem) *Given any $\boldsymbol{z} \in \mathbb{C}^T$, we have $\|\boldsymbol{z}\|_2 = \|\mathrm{DFT}(\boldsymbol{z})\|_2$. More precisely,*

$$\sum_{t=0}^{T-1} |\boldsymbol{z}[t]|^2 = \sum_{p=0}^{T-1} |\mathrm{DFT}(\boldsymbol{z})[p]|^2. \tag{104}$$

This property allows us to easily "normalize" features after each layer onto the sphere $\mathbb{S}^{n-1}$ directly in the spectral domain (see Eq. (26) and (129)).

Circulant matrix and Discrete Fourier Transform for multi-channel signals. We now consider multi-channel 1D signals $\bar{\boldsymbol{z}} \in \mathbb{R}^{C\times T}$. Let $\mathrm{DFT}(\bar{\boldsymbol{z}}) \in \mathbb{C}^{C\times T}$ be a matrix where the c-th row is the DFT of the corresponding signal $\boldsymbol{z}[c]$, i.e.,

$$\mathrm{DFT}(\bar{\boldsymbol{z}}) \doteq \begin{bmatrix} \mathrm{DFT}(\boldsymbol{z}[1])^* \\ \vdots \\ \mathrm{DFT}(\boldsymbol{z}[C])^* \end{bmatrix} \quad \in \mathbb{C}^{C\times T}. \tag{105}$$

Similar to the notation in (86), we denote

$$\begin{aligned} \mathrm{DFT}(\bar{\boldsymbol{z}}) &= \begin{bmatrix} \mathrm{DFT}(\bar{\boldsymbol{z}})[1]^* \\ \vdots \\ \mathrm{DFT}(\bar{\boldsymbol{z}})[C]^* \end{bmatrix} \\ &= [\mathrm{DFT}(\bar{\boldsymbol{z}})(0), \mathrm{DFT}(\bar{\boldsymbol{z}})(1), \ldots, \mathrm{DFT}(\bar{\boldsymbol{z}})(T-1)] \\ &= \{\mathrm{DFT}(\bar{\boldsymbol{z}})[c](t)\}_{c=1,t=0}^{c=C,t=T-1}. \end{aligned} \tag{106}$$

As such, we have $\mathrm{DFT}(\boldsymbol{z}[c]) = \mathrm{DFT}(\bar{\boldsymbol{z}})[c]$.

By using Fact 6, $\mathsf{circ}(\bar{\boldsymbol{z}})$ and $\mathrm{DFT}(\bar{\boldsymbol{z}})$ are related as follows:

$$\mathsf{circ}(\bar{\boldsymbol{z}}) = \begin{bmatrix} \boldsymbol{F}_T^* \cdot \mathsf{diag}(\mathrm{DFT}(\boldsymbol{z}[1])) \cdot \boldsymbol{F}_T \\ \vdots \\ \boldsymbol{F}_T^* \cdot \mathsf{diag}(\mathrm{DFT}(\boldsymbol{z}[C])) \cdot \boldsymbol{F}_T \end{bmatrix} = \begin{bmatrix} \boldsymbol{F}_T^* & \cdots & \boldsymbol{0} \\ \vdots & \ddots & \vdots \\ \boldsymbol{0} & \cdots & \boldsymbol{F}_T^* \end{bmatrix} \cdot \begin{bmatrix} \mathsf{diag}(\mathrm{DFT}(\boldsymbol{z}[1])) \\ \vdots \\ \mathsf{diag}(\mathrm{DFT}(\boldsymbol{z}[C])) \end{bmatrix} \cdot \boldsymbol{F}_T. \tag{107}$$

We now explain how this relationship can be leveraged to produce a fast computation of $\bar{\boldsymbol{E}}$ defined in (92). First, there exists a permutation matrix $\mathbf{P}$ such that

$$\begin{bmatrix} \mathsf{diag}(\mathrm{DFT}(\boldsymbol{z}[1])) \\ \mathsf{diag}(\mathrm{DFT}(\boldsymbol{z}[2])) \\ \vdots \\ \mathsf{diag}(\mathrm{DFT}(\boldsymbol{z}[C])) \end{bmatrix} = \mathbf{P} \cdot \begin{bmatrix} \mathrm{DFT}(\bar{\boldsymbol{z}})(0) & \boldsymbol{0} & \cdots & 0 \\ \boldsymbol{0} & \mathrm{DFT}(\bar{\boldsymbol{z}})(1) & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ \boldsymbol{0} & \boldsymbol{0} & \cdots & \mathrm{DFT}(\bar{\boldsymbol{z}})(T-1) \end{bmatrix}. \tag{108}$$

Combining (107) and (108), we have

$$\mathsf{circ}(\bar{\boldsymbol{z}}) \cdot \mathsf{circ}(\bar{\boldsymbol{z}})^* = \begin{bmatrix} \boldsymbol{F}_T^* & \cdots & \boldsymbol{0} \\ \vdots & \ddots & \vdots \\ \boldsymbol{0} & \cdots & \boldsymbol{F}_T^* \end{bmatrix} \cdot \mathbf{P} \cdot \boldsymbol{D}(\bar{\boldsymbol{z}}) \cdot \mathbf{P}^* \cdot \begin{bmatrix} \boldsymbol{F}_T & \cdots & \boldsymbol{0} \\ \vdots & \ddots & \vdots \\ \boldsymbol{0} & \cdots & \boldsymbol{F}_T \end{bmatrix}, \tag{109}$$

where

$$\boldsymbol{D}(\bar{\boldsymbol{z}}) \doteq \begin{bmatrix} \mathrm{DFT}(\bar{\boldsymbol{z}})(0) \cdot \mathrm{DFT}(\bar{\boldsymbol{z}})(0)^* & \cdots & \boldsymbol{0} \\ \vdots & \ddots & \vdots \\ \boldsymbol{0} & \cdots & \mathrm{DFT}(\bar{\boldsymbol{z}})(T-1) \cdot \mathrm{DFT}(\bar{\boldsymbol{z}})(T-1)^* \end{bmatrix}. \tag{110}$$

Now, consider a collection of m multi-channel 1D signals $\bar{\boldsymbol{Z}} \in \mathbb{R}^{C\times T\times m}$. Similar to the notation in (89), we denote

$$\begin{aligned} \mathrm{DFT}(\bar{\boldsymbol{Z}})[c] &= [\mathrm{DFT}(\bar{\boldsymbol{z}}^1)[c], \ldots, \mathrm{DFT}(\bar{\boldsymbol{z}}^m)[c]] \in \mathbb{R}^{T\times m}, \\ \mathrm{DFT}(\bar{\boldsymbol{Z}})(p) &= [\mathrm{DFT}(\bar{\boldsymbol{z}}^1)(p), \ldots, \mathrm{DFT}(\bar{\boldsymbol{z}}^m)(p)] \in \mathbb{R}^{C\times m}. \end{aligned} \tag{111}$$

By using (109), we have

$$\bar{\boldsymbol{E}} \;=\; \begin{bmatrix} \boldsymbol{F}_T^* & \cdots & \boldsymbol{0} \\ \vdots & \ddots & \vdots \\ \boldsymbol{0} & \cdots & \boldsymbol{F}_T^* \end{bmatrix} \cdot \mathbf{P} \cdot \alpha \cdot \left[\boldsymbol{I} + \alpha \cdot \sum_{i=1}^{m} \boldsymbol{D}(\bar{\boldsymbol{z}}^i)\right]^{-1} \cdot \mathbf{P}^* \cdot \begin{bmatrix} \boldsymbol{F}_T & \cdots & \boldsymbol{0} \\ \vdots & \ddots & \vdots \\ \boldsymbol{0} & \cdots & \boldsymbol{F}_T \end{bmatrix}. \tag{112}$$

Note that $\alpha \cdot \big[\boldsymbol{I} + \alpha \cdot \sum_{i=1}^m \boldsymbol{D}(\bar{\boldsymbol{z}}^i)\big]^{-1}$ is equal to

$$\begin{aligned} &\alpha \begin{bmatrix} \boldsymbol{I}+\alpha \mathrm{DFT}(\bar{\boldsymbol{Z}})(0)\cdot \mathrm{DFT}(\bar{\boldsymbol{Z}}^i)(0)^* & \cdots & \boldsymbol{0} \\ \vdots & \ddots & \vdots \\ \boldsymbol{0} & \cdots & \boldsymbol{I}+\alpha \mathrm{DFT}(\bar{\boldsymbol{Z}})(T-1)\cdot \mathrm{DFT}(\bar{\boldsymbol{Z}})(T-1)^* \end{bmatrix}^{-1} \\ &\qquad = \begin{bmatrix} \alpha\big(\boldsymbol{I}+\alpha \mathrm{DFT}(\bar{\boldsymbol{Z}})(0)\cdot \mathrm{DFT}(\bar{\boldsymbol{Z}})(0)^*\big)^{-1} & \cdots & \boldsymbol{0} \\ \vdots & \ddots & \vdots \\ \boldsymbol{0} & \cdots & \alpha\big(\boldsymbol{I}+\alpha \mathrm{DFT}(\bar{\boldsymbol{Z}})(T-1)\cdot \mathrm{DFT}(\bar{\boldsymbol{Z}})(T-1)^*\big)^{-1} \end{bmatrix}. \end{aligned} \tag{113}$$

Therefore, the calculation of $\bar{\boldsymbol{E}}$ only requires inverting T matrices of size $C \times C$. This motivates us to construct the ReduNet in the spectral domain for the purpose of accelerating the computation, as we explain next.

Shift-invariant ReduNet in the Spectral Domain. Motivated by the result in (113), we introduce the notations $\bar{\mathcal{E}}(p) \in \mathbb{R}^{C\times C\times T}$ and $\bar{\mathcal{C}}^j(p) \in \mathbb{R}^{C\times C\times T}$ given by

$$\bar{\mathcal{E}}(p) \;\doteq\; \alpha \cdot \big[\boldsymbol{I} + \alpha \cdot \mathrm{DFT}(\bar{\boldsymbol{Z}})(p) \cdot \mathrm{DFT}(\bar{\boldsymbol{Z}})(p)^*\big]^{-1} \quad \in \mathbb{C}^{C\times C}, \tag{114}$$

$$\bar{\mathcal{C}}^j(p) \;\doteq\; \alpha_j \cdot \big[\boldsymbol{I} + \alpha_j \cdot \mathrm{DFT}(\bar{\boldsymbol{Z}})(p) \cdot \boldsymbol{\Pi}_j \cdot \mathrm{DFT}(\bar{\boldsymbol{Z}})(p)^*\big]^{-1} \quad \in \mathbb{C}^{C\times C}. \tag{115}$$

In above, $\bar{\mathcal{E}}(p)$ (resp., $\bar{\mathcal{C}}^j(p)$) is the p-th slice of $\bar{\mathcal{E}}$ (resp., $\bar{\mathcal{C}}^j$) on the last dimension. Then, the gradient of $\Delta R_{\mathsf{circ}}(\bar{\boldsymbol{Z}}, \boldsymbol{\Pi})$ with respect to $\bar{\boldsymbol{Z}}$ can be calculated by the following result (Theorem 6 in the main body and here restated for convenience).

Theorem 16 (Computing multi-channel convolutions $\bar{\boldsymbol{E}}$ and $\bar{\boldsymbol{C}}^j$) *Let $\bar{\boldsymbol{U}} \in \mathbb{C}^{C\times T\times m}$ and $\bar{\boldsymbol{W}}^j \in \mathbb{C}^{C\times T\times m}, j = 1, \ldots, k$ be given by*

$$\bar{\boldsymbol{U}}(p) \;\doteq\; \bar{\mathcal{E}}(p) \cdot \mathrm{DFT}(\bar{\boldsymbol{Z}})(p), \tag{116}$$

$$\bar{\boldsymbol{W}}^j(p) \;\doteq\; \bar{\mathcal{C}}^j(p) \cdot \mathrm{DFT}(\bar{\boldsymbol{Z}})(p), \quad j = 1, \ldots, k, \tag{117}$$

for each $p \in \{0, \ldots, T-1\}$. Then, we have

$$\frac{1}{2T}\frac{\partial \log\det(\boldsymbol{I} + \alpha \cdot \mathsf{circ}(\bar{\boldsymbol{Z}})\mathsf{circ}(\bar{\boldsymbol{Z}})^*)}{\partial \bar{\boldsymbol{Z}}} \;=\; \mathrm{IDFT}(\bar{\boldsymbol{U}}), \tag{118}$$

$$\frac{\gamma_j}{2T}\frac{\partial \log\det(\boldsymbol{I} + \alpha_j \cdot \mathsf{circ}(\bar{\boldsymbol{Z}})\bar{\boldsymbol{\Pi}}^j\mathsf{circ}(\bar{\boldsymbol{Z}})^*)}{\partial \bar{\boldsymbol{Z}}} \;=\; \gamma_j \cdot \mathrm{IDFT}(\bar{\boldsymbol{W}}^j\boldsymbol{\Pi}^j). \tag{119}$$

Proof [Also proof to Theorem 6 in the main body]

From (14), (107) and (112), we have

$$\frac{1}{2}\frac{\partial \log\det\Big(\boldsymbol{I}+\alpha\mathsf{circ}(\bar{\boldsymbol{Z}})\mathsf{circ}(\bar{\boldsymbol{Z}})^*\Big)}{\partial \mathsf{circ}(\bar{\boldsymbol{z}}^i)} = \bar{\boldsymbol{E}}\mathsf{circ}(\bar{\boldsymbol{z}}^i) = \bar{\boldsymbol{E}}\begin{bmatrix}\boldsymbol{F}_T^* & \cdots & \boldsymbol{0}\\ \vdots & \ddots & \vdots \\ \boldsymbol{0} & \cdots & \boldsymbol{F}_T^*\end{bmatrix}\begin{bmatrix}\mathsf{diag}(\mathrm{DFT}(\boldsymbol{z}^i[1]))\\ \vdots \\ \mathsf{diag}(\mathrm{DFT}(\boldsymbol{z}^i[C]))\end{bmatrix}\boldsymbol{F}_T \tag{120}$$

$$= \begin{bmatrix}\boldsymbol{F}_T^* & \cdots & \boldsymbol{0}\\ \vdots & \ddots & \vdots \\ \boldsymbol{0} & \cdots & \boldsymbol{F}_T^*\end{bmatrix}\cdot\mathbf{P}\cdot\alpha\cdot\Big[\boldsymbol{I}+\alpha\cdot\sum_i \boldsymbol{D}(\bar{\boldsymbol{z}}^i)\Big]^{-1}\cdot\begin{bmatrix}\mathrm{DFT}(\bar{\boldsymbol{z}}^i)(0) & \cdots & \boldsymbol{0}\\ \vdots & \ddots & \vdots \\ \boldsymbol{0} & \cdots & \mathrm{DFT}(\bar{\boldsymbol{z}}^i)(T-1)\end{bmatrix}\cdot\boldsymbol{F}_T \tag{121}$$

$$= \begin{bmatrix}\boldsymbol{F}_T^* & \cdots & \boldsymbol{0}\\ \vdots & \ddots & \vdots \\ \boldsymbol{0} & \cdots & \boldsymbol{F}_T^*\end{bmatrix}\cdot\mathbf{P}\cdot\begin{bmatrix}\bar{\mathcal{E}}(0)\cdot\mathrm{DFT}(\bar{\boldsymbol{z}}^i)(0) & \cdots & \boldsymbol{0}\\ \vdots & \ddots & \vdots \\ \boldsymbol{0} & \cdots & \bar{\mathcal{E}}(T-1)\cdot\mathrm{DFT}(\bar{\boldsymbol{z}}^i)(T-1)\end{bmatrix}\cdot\boldsymbol{F}_T \tag{122}$$

$$= \begin{bmatrix}\boldsymbol{F}_T^* & \cdots & \boldsymbol{0}\\ \vdots & \ddots & \vdots \\ \boldsymbol{0} & \cdots & \boldsymbol{F}_T^*\end{bmatrix}\cdot\mathbf{P}\cdot\begin{bmatrix}\bar{\boldsymbol{u}}^i(0) & \cdots & \boldsymbol{0}\\ \vdots & \ddots & \vdots \\ \boldsymbol{0} & \cdots & \bar{\boldsymbol{u}}^i(T-1)\end{bmatrix}\cdot\boldsymbol{F}_T = \begin{bmatrix}\boldsymbol{F}_T^* & \cdots & \boldsymbol{0}\\ \vdots & \ddots & \vdots \\ \boldsymbol{0} & \cdots & \boldsymbol{F}_T^*\end{bmatrix}\cdot\begin{bmatrix}\mathsf{diag}(\bar{\boldsymbol{u}}^i[1])\\ \vdots \\ \mathsf{diag}(\bar{\boldsymbol{u}}^i[C])\end{bmatrix}\cdot\boldsymbol{F}_T \tag{123}$$

$$= \mathsf{circ}(\mathrm{IDFT}(\bar{\boldsymbol{u}}^i)). \tag{124}$$

Therefore, we have

$$\begin{aligned}&\frac{1}{2}\frac{\partial \log\det\Big(\boldsymbol{I}+\alpha\cdot\mathsf{circ}(\bar{\boldsymbol{Z}})\cdot\mathsf{circ}(\bar{\boldsymbol{Z}})^*\Big)}{\partial \bar{\boldsymbol{z}}^i}\\ =&\frac{1}{2}\frac{\partial \log\det\Big(\boldsymbol{I}+\alpha\cdot\mathsf{circ}(\bar{\boldsymbol{Z}})\cdot\mathsf{circ}(\bar{\boldsymbol{Z}})^*\Big)}{\partial \mathsf{circ}(\bar{\boldsymbol{z}}^i)}\cdot\frac{\partial \mathsf{circ}(\bar{\boldsymbol{z}}^i)}{\partial \bar{\boldsymbol{z}}^i}\\ =&T\cdot\mathrm{IDFT}(\bar{\boldsymbol{u}}^i).\end{aligned} \tag{125}$$

By collecting the results for all i, we have

$$\frac{\partial \frac{1}{2T}\log\det\Big(\boldsymbol{I}+\alpha\cdot\mathsf{circ}(\bar{\boldsymbol{Z}})\cdot\mathsf{circ}(\bar{\boldsymbol{Z}})^*\Big)}{\partial \bar{\boldsymbol{Z}}} = \mathrm{IDFT}(\bar{\boldsymbol{U}}). \tag{126}$$

In a similar fashion, we get

$$\frac{\partial \frac{\gamma_j}{2T}\log\det\Big(\boldsymbol{I}+\alpha_j\cdot\mathsf{circ}(\bar{\boldsymbol{Z}})\cdot\bar{\boldsymbol{\Pi}}^j\cdot\mathsf{circ}(\bar{\boldsymbol{Z}})^*\Big)}{\partial \bar{\boldsymbol{Z}}} = \gamma_j\cdot\mathrm{IDFT}(\bar{\boldsymbol{W}}^j\cdot\boldsymbol{\Pi}^j). \tag{127}$$

■

By the above theorem, the gradient ascent update in (13) (when applied to $\Delta R_{\mathsf{circ}}(\bar{\boldsymbol{Z}},\boldsymbol{\Pi})$) can be equivalently expressed as an update in frequency domain on $\bar{\boldsymbol{V}}_\ell \doteq \mathrm{DFT}(\bar{\boldsymbol{Z}}_\ell)$ as

$$\bar{\boldsymbol{V}}_{\ell+1}(p) \;\propto\; \bar{\boldsymbol{V}}_\ell(p)+\eta\Big(\bar{\mathcal{E}}_\ell(p)\cdot\bar{\boldsymbol{V}}_\ell(p)-\sum_{j=1}^k\gamma_j\bar{\mathcal{C}}_\ell^j(p)\cdot\bar{\boldsymbol{V}}_\ell(p)\boldsymbol{\Pi}^j\Big),\quad p=0,\ldots,T-1. \tag{128}$$

Similarly, the gradient-guided feature map increment in (26) can be equivalently expressed as an update in frequency domain on $\bar{\boldsymbol{v}}_\ell \doteq \mathrm{DFT}(\bar{\boldsymbol{z}}_\ell)$ as

$$\bar{\boldsymbol{v}}_{\ell+1}(p)\propto\bar{\boldsymbol{v}}_\ell(p)+\eta\cdot\bar{\mathcal{E}}_\ell(p)\bar{\boldsymbol{v}}_\ell(p)-\eta\cdot\boldsymbol{\sigma}\Big([\bar{\mathcal{C}}_\ell^1(p)\bar{\boldsymbol{v}}_\ell(p),\ldots,\bar{\mathcal{C}}_\ell^k(p)\bar{\boldsymbol{v}}_\ell(p)]\Big),\quad p=0,\ldots,T-1, \tag{129}$$

Algorithm 3 Training Algorithm (1D Signal, Shift Invariance, Spectral Domain)

Input: $\bar{\boldsymbol{Z}} \in \mathbb{R}^{C\times T\times m}$, $\boldsymbol{\Pi}$, $\epsilon > 0$, λ, and a learning rate η.

Set $\alpha = \frac{C}{m\epsilon^2}$, $\{\alpha_j = \frac{C}{\mathsf{tr}(\boldsymbol{\Pi}^j)\epsilon^2}\}_{j=1}^k$, $\{\gamma_j = \frac{\mathsf{tr}(\boldsymbol{\Pi}^j)}{m}\}_{j=1}^k$.
Set $\bar{\boldsymbol{V}}_1 = \{\bar{\boldsymbol{v}}_1^i(p) \in \mathbb{C}^C\}_{p=0,i=1}^{T-1,m} \doteq \mathrm{DFT}(\bar{\boldsymbol{Z}}) \in \mathbb{C}^{C\times T\times m}$.
for $\ell = 1, 2, \ldots, L$ **do**
  `# Step 1:Compute network parameters` $\mathcal{E}_\ell$ `and` $\{\mathcal{C}_\ell^j\}_{j=1}^k$.
  for $p = 0, 1, \ldots, T-1$ **do**
    $\bar{\mathcal{E}}_\ell(p) \doteq \alpha \cdot \big[\boldsymbol{I} + \alpha \cdot \bar{\boldsymbol{V}}_\ell(p) \cdot \bar{\boldsymbol{V}}_\ell(p)^*\big]^{-1}$,
    $\bar{\mathcal{C}}_\ell^j(p) \doteq \alpha_j \cdot \big[\boldsymbol{I} + \alpha_j \cdot \bar{\boldsymbol{V}}_\ell(p) \cdot \boldsymbol{\Pi}^j \cdot \bar{\boldsymbol{V}}_\ell(p)^*\big]^{-1}$;
  end for
  `# Step 2:Update feature` $\bar{\boldsymbol{V}}$.
  for $i = 1, \ldots, m$ **do**
    `# Compute (approximately) projection at each frequency` p.
    for $p = 0, 1, \ldots, T-1$ **do**
      Compute $\{\bar{\boldsymbol{p}}_\ell^{ij}(p) \doteq \bar{\mathcal{C}}_\ell^j(p) \cdot \bar{\boldsymbol{v}}_\ell^i(p) \in \mathbb{C}^{C\times 1}\}_{j=1}^k$;
    end for
    `# Compute overall (approximately) projection by aggregating over frequency` p.
    Let $\{\bar{\mathbf{P}}_\ell^{ij} = [\bar{\boldsymbol{p}}_\ell^{ij}(0), \ldots, \bar{\boldsymbol{p}}_\ell^{ij}(T-1)] \in \mathbb{C}^{C\times T}\}_{j=1}^k$;
    `# Compute soft assignment from (approximately) projection.`
    Compute $\left\{\widehat{\boldsymbol{\pi}}_\ell^{ij} = \frac{\exp(-\lambda\|\bar{\mathbf{P}}_\ell^{ij}\|_F)}{\sum_{j=1}^k \exp(-\lambda\|\bar{\mathbf{P}}_\ell^{ij}\|_F)}\right\}_{j=1}^k$;
    `# Compute update at each frequency` p.
    for $p = 0, 1, \ldots, T-1$ **do**
      $\bar{\boldsymbol{v}}_{\ell+1}^i(p) = \mathcal{P}_{\mathbb{S}^{n-1}}\Big(\bar{\boldsymbol{v}}_\ell^i(p) + \eta\Big(\bar{\mathcal{E}}_\ell(p)\bar{\boldsymbol{v}}_\ell^i(p) - \sum_{j=1}^k \gamma_j \cdot \widehat{\boldsymbol{\pi}}_\ell^{ij} \cdot \bar{\mathcal{C}}_\ell^j(p) \cdot \bar{\boldsymbol{v}}_\ell^i(p)\Big)\Big)$;
    end for
  end for
  Set $\bar{\boldsymbol{Z}}_{\ell+1} = \mathrm{IDFT}(\bar{\boldsymbol{V}}_{\ell+1})$ as the feature at the ℓ-th layer;
end for

Output: features $\bar{\boldsymbol{Z}}_{L+1}$, the learned filters $\{\bar{\boldsymbol{E}}_\ell(p)\}_{\ell,p}$ and $\{\bar{\mathcal{C}}_\ell^j(p)\}_{j,\ell,p}$.

subject to the constraint that $\|\bar{\boldsymbol{v}}_{\ell+1}\|_F = \|\bar{\boldsymbol{z}}_{\ell+1}\|_F = 1$ (the first equality follows from Fact 7).

We summarize the training, or construction to be more precise, of ReduNet in the spectral domain in Algorithm 3.

Appendix C. ReduNet for 2D Circular Translation Invariance

To a large degree, both conceptually and technically, the 2D case is very similar to the 1D case that we have studied carefully in the previous Appendix B. For the sake of consistency and completeness, we here give a brief account.

C.1 Doubly Block Circulant Matrix

In this section, we consider $\boldsymbol{z}$ as a 2D signal such as an image, and use H and W to denote its "height" and "width", respectively. It will be convenient to work with both a matrix representation

$$\boldsymbol{z} = \begin{bmatrix} z(0,0) & z(0,1) & \cdots & z(0,W-1) \\ z(1,0) & z(1,1) & \cdots & z(1,W-1) \\ \vdots & \vdots & \ddots & \vdots \\ z(H-1,0) & z(H-1,1) & \cdots & z(H-1,W-1) \end{bmatrix} \in \mathbb{R}^{H \times W}, \tag{130}$$

as well as a vector representation

$$\begin{aligned} \mathsf{vec}(\boldsymbol{z}) \doteq \Big[& z(0,0), \ldots, z(0,W-1), z(1,0), \ldots, \\ & z(1,W-1), \ldots, z(H-1,0), \ldots, z(H-1,W-1) \Big]^* \in \mathbb{R}^{(H \times W)}. \end{aligned} \tag{131}$$

We represent the circular translated version of $\boldsymbol{z}$ as $\mathsf{trans}_{p,q}(\boldsymbol{z}) \in \mathbb{R}^{H \times W}$ by an amount of p and q on the vertical and horizontal directions, respectively. That is, we let

$$\mathsf{trans}_{p,q}(\boldsymbol{z})(h,w) \doteq \boldsymbol{z}(h - p \bmod H, w - q \bmod W), \tag{132}$$

where $\forall (h,w) \in \{0,\ldots,H-1\} \times \{0,\ldots,W-1\}$. It is obvious that $\mathsf{trans}_{0,0}(\boldsymbol{z}) = \boldsymbol{z}$. Moreover, there is a total number of $H \times W$ distinct translations given by $\{\mathsf{trans}_{p,q}(\boldsymbol{z}), (p,q) \in \{0,\ldots,H-1\} \times \{0,\ldots,W-1\}\}$. We may arrange the vector representations of them into a matrix and obtain

$$\begin{aligned} \mathsf{circ}(\boldsymbol{z}) \doteq \Big[& \mathsf{vec}(\mathsf{trans}_{0,0}(\boldsymbol{z})), \ldots, \mathsf{vec}(\mathsf{trans}_{0,W-1}(\boldsymbol{z})), \mathsf{vec}(\mathsf{trans}_{1,0}(\boldsymbol{z})), \ldots, \mathsf{vec}(\mathsf{trans}_{1,W-1}(\boldsymbol{z})), \\ & \ldots, \\ & \mathsf{vec}(\mathsf{trans}_{H-1,0}(\boldsymbol{z})), \ldots, \mathsf{vec}(\mathsf{trans}_{H-1,W-1}(\boldsymbol{z})) \Big] \in \mathbb{R}^{(H \times W) \times (H \times W)}. \end{aligned} \tag{133}$$

The matrix $\mathsf{circ}(\boldsymbol{z})$ is known as the *doubly block circulant matrix* associated with $\boldsymbol{z}$ (see, e.g., Abidi et al. (2016); Sedghi et al. (2019)).

We now consider a multi-channel 2D signal represented as a tensor $\bar{\boldsymbol{z}} \in \mathbb{R}^{C \times H \times W}$, where C is the number of channels. The c-th channel of $\bar{\boldsymbol{z}}$ is represented as $\bar{\boldsymbol{z}}[c] \in \mathbb{R}^{H \times W}$, and the (h,w)-th pixel is represented as $\bar{\boldsymbol{z}}(h,w) \in \mathbb{R}^C$. To compute the coding rate reduction for a collection of such multi-channel 2D signals, we may flatten the tenor representation into a vector representation by concatenating the vector representation of each channel, i.e., we let

$$\mathsf{vec}(\bar{\boldsymbol{z}}) = [\mathsf{vec}(\bar{\boldsymbol{z}}[1])^*, \ldots, \mathsf{vec}(\bar{\boldsymbol{z}}[C])^*]^* \quad \in \mathbb{R}^{(C \times H \times W)} \tag{134}$$

Furthermore, to obtain shift invariance for coding rate reduction, we may generate a collection of translated versions of $\bar{\boldsymbol{z}}$ (along two spatial dimensions). Stacking the vector representation for such translated copies as column vectors, we obtain

$$\mathsf{circ}(\bar{\boldsymbol{z}}) \doteq \begin{bmatrix} \mathsf{circ}(\bar{\boldsymbol{z}}[1]) \\ \vdots \\ \mathsf{circ}(\bar{\boldsymbol{z}}[C]) \end{bmatrix} \quad \in \mathbb{R}^{(C \times H \times W) \times (H \times W)}. \tag{135}$$

We can now define a *translation invariant coding rate reduction* for multi-channel 2D signals. Consider a collection of m multi-channel 2D signals $\{\bar{\boldsymbol{z}}^i \in \mathbb{R}^{C\times H\times W}\}_{i=1}^m$. Compactly representing the data by $\bar{\boldsymbol{Z}} \in \mathbb{R}^{C\times H\times W\times m}$ where the i-th slice on the last dimension is $\bar{\boldsymbol{z}}^i$, we denote

$$\mathsf{circ}(\bar{\boldsymbol{Z}}) = [\mathsf{circ}(\bar{\boldsymbol{z}}^1), \ldots, \mathsf{circ}(\bar{\boldsymbol{z}}^m)] \quad \in \mathbb{R}^{(C\times H\times W)\times(H\times W\times m)}. \tag{136}$$

Then, we define

$$\Delta R_{\mathsf{circ}}(\bar{\boldsymbol{Z}}, \boldsymbol{\Pi}) \doteq \frac{1}{HW}\Delta R(\mathsf{circ}(\bar{\boldsymbol{Z}}), \bar{\boldsymbol{\Pi}}) = \frac{1}{2HW}\log\det\left(\boldsymbol{I} + \alpha\cdot\mathsf{circ}(\bar{\boldsymbol{Z}})\cdot\mathsf{circ}(\bar{\boldsymbol{Z}})^*\right) - \sum_{j=1}^{k}\frac{\gamma_j}{2HW}\log\det\left(\boldsymbol{I} + \alpha_j\cdot\mathsf{circ}(\bar{\boldsymbol{Z}})\cdot\bar{\boldsymbol{\Pi}}^j\cdot\mathsf{circ}(\bar{\boldsymbol{Z}})^*\right), \tag{137}$$

where $\alpha = \frac{CHW}{mHW\epsilon^2} = \frac{C}{m\epsilon^2}$, $\alpha_j = \frac{CHW}{\mathsf{tr}(\boldsymbol{\Pi}^j)HW\epsilon^2} = \frac{C}{\mathsf{tr}(\boldsymbol{\Pi}^j)\epsilon^2}$, $\gamma_j = \frac{\mathsf{tr}(\boldsymbol{\Pi}^j)}{m}$, and $\bar{\boldsymbol{\Pi}}^j$ is augmented membership matrix in an obvious way.

By following an analogous argument as in the 1D case, one can show that ReduNet for multi-channel 2D signals naturally gives rise to the multi-channel 2D circulant convolution operations. We omit the details, and focus on the construction of ReduNet in the frequency domain.

C.2 Fast Computation in Spectral Domain

Doubly block circulant matrix and 2D-DFT. Similar to the case of circulant matrices for 1D signals, all doubly block circulant matrices share the same set of eigenvectors, and these eigenvectors form a unitary matrix given by

$$\boldsymbol{F} \doteq \boldsymbol{F}_H \otimes \boldsymbol{F}_W \quad \in \mathbb{C}^{(H\times W)\times(H\times W)}, \tag{138}$$

where $\otimes$ denotes the Kronecker product and $\boldsymbol{F}_H, \boldsymbol{F}_W$ are defined as in (97).

Analogous to Fact 4, $\boldsymbol{F}$ defines 2D-DFT as follows.

Fact 8 (2D-DFT as matrix-vector multiplication) *The 2D-DFT of a signal $\boldsymbol{z} \in \mathbb{R}^{H\times W}$ can be computed as*

$$\mathsf{vec}(\mathrm{DFT}(\boldsymbol{z})) \doteq \boldsymbol{F}\cdot\mathsf{vec}(\boldsymbol{z}) \quad \in \mathbb{C}^{(H\times W)}, \tag{139}$$

where $\forall (p,q) \in \{0,\ldots,H-1\}\times\{0,\ldots,W-1\}$,

$$\mathrm{DFT}(\boldsymbol{z})(p,q) = \frac{1}{\sqrt{H\cdot W}}\sum_{h=0}^{H-1}\sum_{w=0}^{W-1}\boldsymbol{z}(h,w)\cdot\omega_H^{p\cdot h}\omega_W^{q\cdot w}. \tag{140}$$

The 2D-IDFT of a signal $\boldsymbol{v} \in \mathbb{C}^{H\times W}$ can be computed as

$$\mathsf{vec}(\mathrm{IDFT}(\boldsymbol{v})) \doteq \boldsymbol{F}_T^*\cdot\mathsf{vec}(\boldsymbol{v}) \quad \in \mathbb{C}^{(H\times W)}, \tag{141}$$

where $\forall (h,w) \in \{0,\ldots,H-1\}\times\{0,\ldots,W-1\}$,

$$\mathrm{IDFT}(\boldsymbol{v})(h,w) = \frac{1}{\sqrt{H\cdot W}}\sum_{p=0}^{H-1}\sum_{q=0}^{W-1}v(p,q)\cdot\omega_H^{-p\cdot h}\omega_W^{-q\cdot w}. \tag{142}$$

Analogous to Fact 9, $\boldsymbol{F}$ relates $\mathrm{DFT}(\boldsymbol{z})$ and $\mathsf{circ}(\boldsymbol{z})$ as follows.

Fact 9 (2D-DFT are eigenvalues of the doubly block circulant matrix) *Given a signal $\boldsymbol{z} \in \mathbb{C}^{H\times W}$, we have*

$$\boldsymbol{F}\cdot\mathsf{circ}(\boldsymbol{z})\cdot\boldsymbol{F}^* = \mathsf{diag}(\mathsf{vec}(\mathrm{DFT}(\boldsymbol{z}))) \quad \textit{or} \quad \mathsf{circ}(\boldsymbol{z}) = \boldsymbol{F}^*\cdot\mathsf{diag}(\mathsf{vec}(\mathrm{DFT}(\boldsymbol{z})))\cdot\boldsymbol{F}. \tag{143}$$

Doubly block circulant matrix and 2D-DFT for multi-channel signals. We now consider multi-channel 2D signals $\bar{\boldsymbol{z}} \in \mathbb{R}^{C\times H\times W}$. Let $\mathrm{DFT}(\bar{\boldsymbol{z}}) \in \mathbb{C}^{C\times H\times W}$ be a matrix where the c-th slice on the first dimension is the DFT of the corresponding signal $\boldsymbol{z}[c]$. That is, $\mathrm{DFT}(\bar{\boldsymbol{z}})[c] = \mathrm{DFT}(\boldsymbol{z}[c]) \in \mathbb{C}^{H\times W}$. We use $\mathrm{DFT}(\bar{\boldsymbol{z}})(p,q) \in \mathbb{C}^C$ to denote slicing of $\bar{\boldsymbol{z}}$ on the frequency dimensions.

By using Fact 9, $\mathsf{circ}(\bar{\boldsymbol{z}})$ and $\mathrm{DFT}(\bar{\boldsymbol{z}})$ are related as follows:

$$\begin{aligned}\mathsf{circ}(\bar{\boldsymbol{z}}) &= \begin{bmatrix}\boldsymbol{F}^*\cdot\mathsf{diag}(\mathsf{vec}(\mathrm{DFT}(\boldsymbol{z}[1])))\cdot\boldsymbol{F}\\ \vdots \\ \boldsymbol{F}^*\cdot\mathsf{diag}(\mathsf{vec}(\mathrm{DFT}(\boldsymbol{z}[C])))\cdot\boldsymbol{F}\end{bmatrix}\\ &= \begin{bmatrix}\boldsymbol{F}^* & \cdots & \boldsymbol{0}\\ \boldsymbol{0} & \cdots & \boldsymbol{0}\\ \vdots & \ddots & \vdots\\ \boldsymbol{0} & \cdots & \boldsymbol{F}^*\end{bmatrix}\cdot\begin{bmatrix}\mathsf{diag}(\mathsf{vec}(\mathrm{DFT}(\boldsymbol{z}[1])))\\ \mathsf{diag}(\mathsf{vec}(\mathrm{DFT}(\boldsymbol{z}[2])))\\ \vdots\\ \mathsf{diag}(\mathsf{vec}(\mathrm{DFT}(\boldsymbol{z}[C])))\end{bmatrix}\cdot\boldsymbol{F}.\end{aligned} \tag{144}$$

Similar to the 1D case, this relation can be leveraged to produce a fast implementation of ReduNet in the spectral domain.

Translation-invariant ReduNet in the Spectral Domain. Given a collection of multi-channel 2D signals $\bar{\boldsymbol{Z}} \in \mathbb{R}^{C\times H\times W\times m}$, we denote

$$\mathrm{DFT}(\bar{\boldsymbol{Z}})(p,q) \doteq [\mathrm{DFT}(\bar{\boldsymbol{z}}^1)(p,q),\ldots,\mathrm{DFT}(\bar{\boldsymbol{z}}^m)(p,q)] \quad \in \mathbb{R}^{C\times m}. \tag{145}$$

We introduce the notations $\bar{\mathcal{E}}(p,q) \in \mathbb{R}^{C\times C\times H\times W}$ and $\bar{\mathcal{C}}^j(p,q) \in \mathbb{R}^{C\times C\times H\times W}$ given by

$$\bar{\mathcal{E}}(p,q) \doteq \alpha\cdot\big[\boldsymbol{I}+\alpha\cdot\mathrm{DFT}(\bar{\boldsymbol{Z}})(p,q)\cdot\mathrm{DFT}(\bar{\boldsymbol{Z}})(p,q)^*\big]^{-1} \quad \in \mathbb{C}^{C\times C}, \tag{146}$$

$$\bar{\mathcal{C}}^j(p,q) \doteq \alpha_j\cdot\big[\boldsymbol{I}+\alpha_j\cdot\mathrm{DFT}(\bar{\boldsymbol{Z}})(p,q)\cdot\boldsymbol{\Pi}_j\cdot\mathrm{DFT}(\bar{\boldsymbol{Z}})(p,q)^*\big]^{-1} \quad \in \mathbb{C}^{C\times C}. \tag{147}$$

In above, $\bar{\mathcal{E}}(p,q)$ (resp., $\bar{\mathcal{C}}^j(p,q)$) is the (p,q)-th slice of $\bar{\mathcal{E}}$ (resp., $\bar{\mathcal{C}}^j$) on the last two dimensions. Then, the gradient of $\Delta R_{\mathsf{circ}}(\bar{\boldsymbol{Z}},\boldsymbol{\Pi})$ with respect to $\bar{\boldsymbol{Z}}$ can be calculated by the following result.

Theorem 17 (Computing multi-channel 2D convolutions $\bar{\boldsymbol{E}}$ and $\bar{\boldsymbol{C}}^j$) *Suppose $\bar{\boldsymbol{U}} \in \mathbb{C}^{C\times H\times W\times m}$ and $\bar{\boldsymbol{W}}^j \in \mathbb{C}^{C\times H\times W\times m}, j=1,\ldots,k$ are given by*

$$\bar{\boldsymbol{U}}(p,q) \doteq \bar{\mathcal{E}}(p,q)\cdot\mathrm{DFT}(\bar{\boldsymbol{Z}})(p,q), \tag{148}$$

$$\bar{\boldsymbol{W}}^j(p,q) \doteq \bar{\mathcal{C}}^j(p,q)\cdot\mathrm{DFT}(\bar{\boldsymbol{Z}})(p,q), \quad j=1,\ldots,k, \tag{149}$$

for each $(p, q) \in \{0, \ldots, H-1\} \times \{0, \ldots, W-1\}$. *Then, we have*

$$\frac{1}{2HW}\frac{\partial \log\det(\boldsymbol{I} + \alpha \cdot \mathsf{circ}(\bar{\boldsymbol{Z}})\mathsf{circ}(\bar{\boldsymbol{Z}})^*)}{\partial \bar{\boldsymbol{Z}}} = \mathrm{IDFT}(\bar{\boldsymbol{U}}), \tag{150}$$

$$\frac{1}{2HW}\frac{\partial \left(\gamma_j \log\det(\boldsymbol{I} + \alpha_j \cdot \mathsf{circ}(\bar{\boldsymbol{Z}})\bar{\boldsymbol{\Pi}}^j\mathsf{circ}(\bar{\boldsymbol{Z}})^*)\right)}{\partial \bar{\boldsymbol{Z}}} = \gamma_j \cdot \mathrm{IDFT}(\bar{\boldsymbol{W}}^j\boldsymbol{\Pi}^j). \tag{151}$$

This result shows that the calculation of the derivatives for the 2D case is analogous to that of the 1D case. Therefore, the construction of the ReduNet for 2D translation invariance can be performed using Algorithm 3 with straightforward extensions.

Appendix D. Additional Simulations and Experiments for MCR^2

D.1 Simulations - Verifying Diversity Promoting Properties of MCR^2

As proved in Theorem 12, the proposed MCR^2 objective promotes within-class diversity. In this section, we use simulated data to verify the diversity promoting property of MCR^2. As shown in Table 4, we calculate our proposed MCR^2 objective on simulated data. We observe that orthogonal subspaces with *higher* dimension achieve higher MCR^2 value, which is consistent with our theoretical analysis in Theorem 12.

D.2 Implementation Details

Training Setting. We mainly use ResNet-18 (He et al., 2016) in our experiments, where we use 4 residual blocks with layer widths $\{64, 128, 256, 512\}$. The implementation of network architectures used in this paper are mainly based on this github repo.[37] For data augmentation in the supervised setting, we apply the `RandomCrop` and `RandomHorizontalFlip`. For the supervised setting, we train the models for 500 epochs and use stage-wise learning rate decay every 200 epochs (decay by a factor of 10). For the supervised setting, we train the models for 100 epochs and use stage-wise learning rate decay at 20-th epoch and 40-th epoch (decay by a factor of 10).

Evaluation Details. For the supervised setting, we set the number of principal components for nearest subspace classifier $r_j = 30$. We also study the effect of r_j in Section D.3.2. For the CIFAR100 dataset, we consider 20 superclasses and set the cluster number as 20, which is the same setting as in Chang et al. (2017); Wu et al. (2018).

Datasets. We apply the default datasets in PyTorch, including CIFAR10, CIFAR100, and STL10.

Augmentations $\mathcal{T}$ used for the self-supervised setting. We apply the same data augmentation for CIFAR10 dataset and CIFAR100 dataset and the pseudo-code is as follows.

```
import torchvision.transforms as transforms
TRANSFORM = transforms.Compose([
    transforms.RandomResizedCrop(32),
    transforms.RandomHorizontalFlip(),
    transforms.RandomApply([transforms.ColorJitter(0.4, 0.4, 0.4, 0.1)], p=0.8),
    transforms.RandomGrayscale(p=0.2),
    transforms.ToTensor()])
```

The augmentations we use for STL10 dataset and the pseudo-code is as follows.

37. https://github.com/kuangliu/pytorch-cifar

	R	R_c	ΔR	Orthogonal?	Output Dimension
Random Gaussian	552.70	193.29	360.41	✓	512
Subspace ($d_j = 50$)	545.63	108.46	**437.17**	✓	512
Subspace ($d_j = 40$)	487.07	92.71	394.36	✓	512
Subspace ($d_j = 30$)	413.08	74.84	338.24	✓	512
Subspace ($d_j = 20$)	318.52	54.48	264.04	✓	512
Subspace ($d_j = 10$)	195.46	30.97	164.49	✓	512
Subspace ($d_j = 1$)	31.18	4.27	26.91	✓	512
Random Gaussian	292.71	154.13	138.57	✓	256
Subspace ($d_j = 25$)	288.65	56.34	**232.31**	✓	256
Subspace ($d_j = 20$)	253.51	47.58	205.92	✓	256
Subspace ($d_j = 15$)	211.97	38.04	173.93	✓	256
Subspace ($d_j = 10$)	161.87	27.52	134.35	✓	256
Subspace ($d_j = 5$)	98.35	15.55	82.79	✓	256
Subspace ($d_j = 1$)	27.73	3.92	23.80	✓	256
Random Gaussian	150.05	110.85	39.19	✓	128
Subspace ($d_j = 12$)	144.36	27.72	**116.63**	✓	128
Subspace ($d_j = 10$)	129.12	24.06	105.05	✓	128
Subspace ($d_j = 8$)	112.01	20.18	91.83	✓	128
Subspace ($d_j = 6$)	92.55	16.04	76.51	✓	128
Subspace ($d_j = 4$)	69.57	11.51	58.06	✓	128
Subspace ($d_j = 2$)	41.68	6.45	35.23	✓	128
Subspace ($d_j = 1$)	24.28	3.57	20.70	✓	128
Subspace ($d_j = 50$)	145.60	75.31	70.29	✗	128
Subspace ($d_j = 40$)	142.69	65.68	77.01	✗	128
Subspace ($d_j = 30$)	135.42	54.27	81.15	✗	128
Subspace ($d_j = 20$)	120.98	40.71	80.27	✗	128
Subspace ($d_j = 15$)	111.10	32.89	78.21	✗	128
Subspace ($d_j = 12$)	101.94	27.73	74.21	✗	128

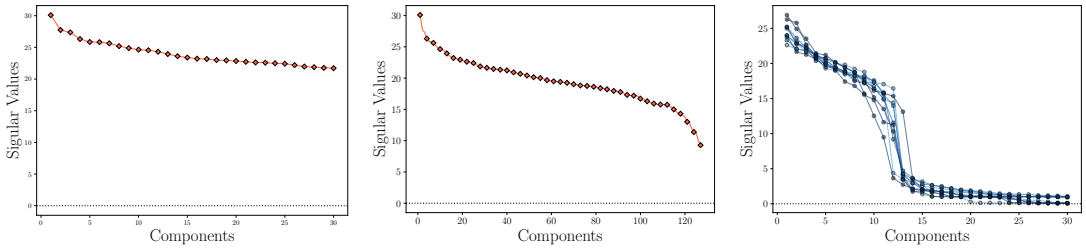
Table 4: **MCR^2 objective on simulated data.** We evaluate the proposed MCR^2 objective defined in (11), including R, R_c, and ΔR, on simulated data. The output dimension d is set as 512, 256, and 128. We set the batch size as $m = 1000$ and random assign the label of each sample from 0 to 9, i.e., 10 classes. We generate two types of data: 1) (Random Gaussian) For comparison with data without structures, for each class we generate random vectors sampled from Gaussian distribution (the dimension is set as the output dimension d) and normalize each vector to be on the unit sphere. 2) (Subspace) For each class, we generate vectors sampled from its corresponding subspace with dimension d_j and normalize each vector to be on the unit sphere. We consider the subspaces from different classes are orthogonal/nonorthogonal to each other.

```
import torchvision.transforms as transforms
TRANSFORM = transforms.Compose([
    transforms.RandomResizedCrop(96),
    transforms.RandomHorizontalFlip(),
    transforms.RandomApply([transforms.ColorJitter(0.8, 0.8, 0.8, 0.2)], p=0.8),
    transforms.RandomGrayscale(p=0.2),
    GaussianBlur(kernel_size=9),
    transforms.ToTensor()])
```

Cross-entropy training details. For CE models presented in Table 1, Figure 18(d)-18(f), and Figure 19, we use the same network architecture, ResNet-18 (He et al., 2016), for cross-entropy training on CIFAR10, and set the output dimension as 10 for the last layer. We apply SGD, and set learning rate `lr=0.1`, momentum `momentum=0.9`, and weight decay `wd=5e-4`. We set the total number of training epoch as 400, and use stage-wise learning rate decay every 150 epochs (decay by a factor of 10).

D.3 Additional Experimental Results

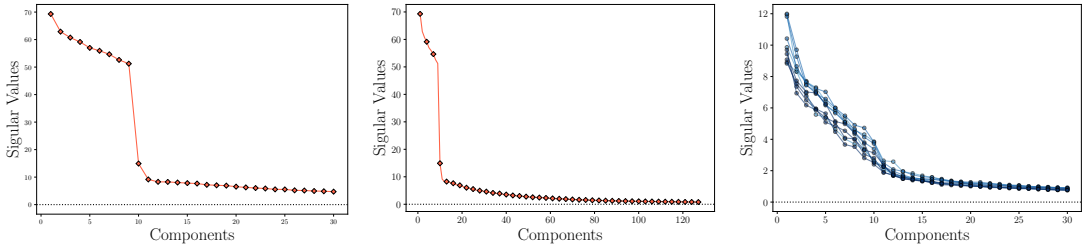
D.3.1 PCA Results of MCR^2 Training versus Cross-Entropy Training



(a) PCA: MCR^2 training learned features for overall data (first 30 components).

(b) PCA: MCR^2 training learned features for overall data.

(c) PCA: MCR^2 training learned features for every class.



(d) PCA: cross-entropy training learned features for overall data (first 30 components).

(e) PCA: cross-entropy training learned features for overall data.

(f) PCA: cross-entropy training learned features for every class.

Figure 18: Principal component analysis (PCA) of learned representations for the MCR^2 trained model (**first row**) and the cross-entropy trained model (**second row**).

For comparison, similar to Figure 12(c), we calculate the principle components of representations learned by MCR^2 training and cross-entropy training. For cross-entropy training, we take the output of the second last layer as the learned representation. The results are summarized in Figure 18. We also compare the cosine similarity between learned representations for both MCR^2 training and cross-entropy training, and the results are presented in Figure 19.

As shown in Figure 18, we observe that representations learned by MCR^2 are much more diverse, the dimension of learned features (each class) is around a dozen, and the

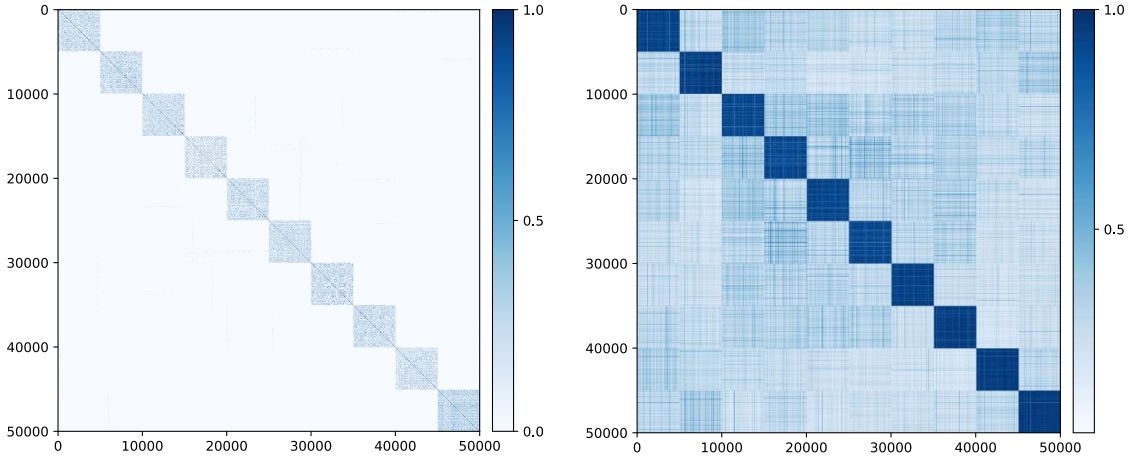
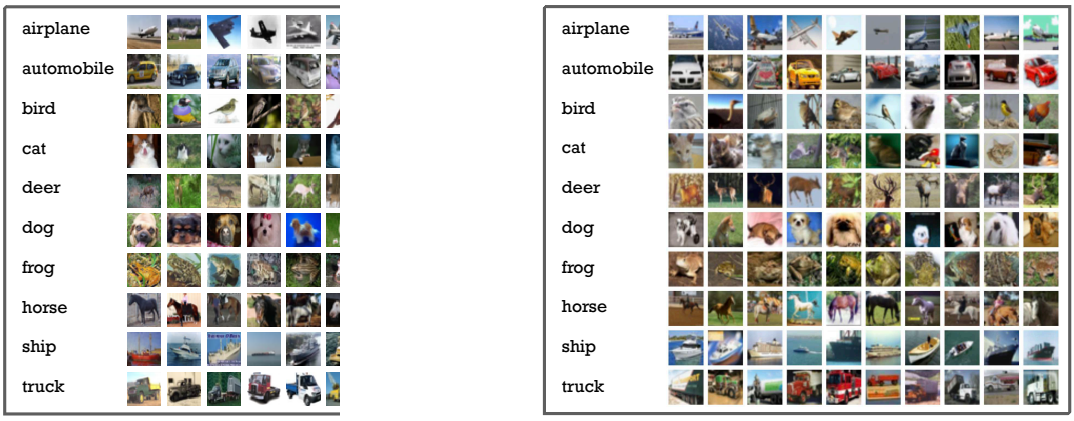


Figure 19: Cosine similarity between learned features by using the MCR^2 objective (**left**) and CE loss (**right**).



(a) 10 representative images fro
on top-10 principal componen
learned representations by MCR^2.

(b) Randomly selected 10 images from each class.

Figure 20: Visualization of top-10 "principal" images for each class in the CIFAR10 dataset. **(a)** For each class-j, we first compute the top-10 singular vectors of the SVD of the learned features $\boldsymbol{Z}^j$. Then for the l-th singular vector of class j, $\boldsymbol{u}_j^l$, and for the feature of the i-th image of class j, $\boldsymbol{z}_j^i$, we calculate the absolute value of inner product, $|\langle \boldsymbol{z}_j^i, \boldsymbol{u}_j^l \rangle|$, then we select the largest one for each singular vector within class j. Each row corresponds to one class, and each image corresponds to one singular vector, ordered by the value of the associated singular value. **(b)** For each class, 10 images are randomly selected in the dataset. These images are the ones displayed in the CIFAR dataset website (Krizhevsky, 2009).

dimension of the overall features is nearly 120, and the output dimension is 128. In contrast, the dimension of the overall features learned using entropy is slightly greater than 10, which

is much smaller than that learned by MCR^2. From Figure 19, for MCR^2 training, we find that the features of different class are almost orthogonal.

Visualize representative images selected from CIFAR10 dataset by using MCR^2. As mentioned in Section 1.3, obtaining the properties of desired representation in the proposed MCR^2 principle is equivalent to performing *nonlinear generalized principle components* on the given dataset. As shown in Figure 18(a)-18(c), MCR^2 can indeed learn such diverse and discriminative representations. In order to better interpret the representations learned by MCR^2, we select images according to their "principal" components (singular vectors using SVD) of the learned features. In Figure 13, we visualize images selected from class-'Bird' and class-'Ship'. For each class, we first compute top-10 singular vectors of the SVD of the learned features and then for each of the top singular vectors, we display in each row the top-10 images whose corresponding features are closest to the singular vector. As shown in Figure 13, we observe that images in the same row share many common characteristics such as shapes, textures, patterns, and styles, whereas images in different rows are significantly different from each other—suggesting our method captures all the different "modes" of the data even within the same class. Notice that top rows are associated with components with larger singular values, hence they are images that show up more frequently in the dataset.

In Figure 20(a), we visualize the 10 "principal" images selected from CIFAR10 for each of the 10 classes. That is, for each class, we display the 10 images whose corresponding features are most coherent with the top-10 singular vectors. We observe that the selected images are much more diverse and representative than those selected randomly from the dataset (displayed on the CIFAR official website), indicating such principal images can be used as a good "summary" of the dataset.

D.3.2 Experimental Results of MCR^2 in the Supervised Learning Setting.

Training details for mainline experiment. For the model presented in Figure 1 (**Right**) and Figure 12, we use ResNet-18 to parameterize $f(\cdot, \boldsymbol{\theta})$, and we set the output dimension $d = 128$, precision $\epsilon^2 = 0.5$, mini-batch size $m = 1,000$. We use SGD in Pytorch (Paszke et al., 2019) as the optimizer, and set the learning rate `lr=0.01`, weight decay `wd=5e-4`, and `momentum=0.9`.

Experiments for studying the effect of hyperparameters and architectures. We present the experimental results of MCR^2 training in the supervised setting by using various training hyperparameters and different network architectures. The results are summarized in Table 5. Besides the ResNet architecture, we also consider VGG architecture (Simonyan and Zisserman, 2015) and ResNext achitecture (Xie et al., 2017). From Table 5, we find that larger batch size m can lead to better performance. Also, models with higher output dimension d require larger training batch size m.

Effect of r_j on classification. Unless otherwise stated, we set the number of components $r_j = 30$ for nearest subspace classification. We study the effect of r_j when used for classification, and the results are summarized in Table 6. We observe that the nearest subspace classification works for a wide range of r_j.

Effect of ϵ^2 on learning from corrupted labels. To further study the proposed MCR^2 on learning from corrupted labels, we use different precision parameters, $\epsilon^2 = 0.75, 1.0$, in

Arch	Dim n	Precision ϵ^2	BatchSize m	lr	ACC	Comment
ResNet-18	128	0.5	1,000	0.01	0.922	Mainline, Fig 12
ResNext-29	128	0.5	1,000	0.01	0.925	Different
VGG-11	128	0.5	1,000	0.01	0.907	Architecture
ResNet-18	512	0.5	1,000	0.01	0.886	Effect of
ResNet-18	256	0.5	1,000	0.01	0.921	Output
ResNet-18	64	0.5	1,000	0.01	0.922	Dimension
ResNet-18	128	1.0	1,000	0.01	0.930	Effect of
ResNet-18	128	0.4	1,000	0.01	0.919	precision
ResNet-18	128	0.2	1,000	0.01	0.900	
ResNet-18	128	0.5	500	0.01	0.823	
ResNet-18	128	0.5	2,000	0.01	0.930	Effect of
ResNet-18	128	0.5	4,000	0.01	0.925	Batch Size
ResNet-18	512	0.5	2,000	0.01	0.924	
ResNet-18	512	0.5	4,000	0.01	0.921	
ResNet-18	128	0.5	1,000	0.05	0.860	
ResNet-18	128	0.5	1,000	0.005	0.923	Effect of lr
ResNet-18	128	0.5	1,000	0.001	0.922	

Table 5: Experiments of MCR^2 in the supervised setting on the CIFAR10 dataset.

Number of components	$r_j = 10$	$r_j = 20$	$r_j = 30$	$r_j = 40$	$r_j = 50$
Mainline (Label Noise Ratio=0.0)	0.926	0.925	0.922	0.923	0.921
Label Noise Ratio=0.1	0.917	0.917	0.911	0.918	0.917
Label Noise Ratio=0.2	0.906	0.906	0.897	0.906	0.905
Label Noise Ratio=0.3	0.882	0.879	0.881	0.881	0.881
Label Noise Ratio=0.4	0.864	0.866	0.866	0.867	0.864
Label Noise Ratio=0.5	0.839	0.841	0.843	0.841	0.837

Table 6: Effect of number of components r_j for nearest subspace classification in the supervised setting.

addition to the one shown in Table 1. Except for the precision parameter ϵ^2, all the other parameters are the same as the mainline experiment (the first row in Table 5). The first row ($\epsilon^2 = 0.5$) in Table 7 is identical to the MCR^2 training in Table 10. Notice that with slightly different choices in ϵ^2, one might even see slightly improved performance over the ones reported in the main body.

D.4 Comparison with Related Work on Label Noise

We compare the proposed MCR^2 with OLE (Lezama et al., 2018), Large Margin Deep Networks (Elsayed et al., 2018), and ITLM (Shen and Sanghavi, 2019) in label noise robustness experiments on CIFAR10 dataset. In Table 8, we compare MCR^2 with OLE (Lezama et al.,

Precision	Ratio=0.1	Ratio=0.2	Ratio=0.3	Ratio=0.4	Ratio=0.5
$\epsilon^2 = 0.5$	0.911	0.897	0.881	0.866	0.843
$\epsilon^2 = 0.75$	**0.923**	0.908	**0.899**	**0.876**	0.836
$\epsilon^2 = 1.0$	0.919	**0.911**	0.896	0.870	**0.845**

Table 7: Effect of Precision ϵ^2 on classification results with features learned with labels corrupted at different levels by using MCR^2 training.

2018) and Large Margin Deep Networks (Elsayed et al., 2018) on the corrupted label task using the same network, MCR^2 achieves significant better performance. We compare MCR^2 with ITLM (Shen and Sanghavi, 2019) using the same network. MCR2 achieves better performance without any noise ratio dependent hyperparameters as required by Shen and Sanghavi (2019).

ResNet18	Ratio=0.1	Ratio=0.2	Ratio=0.3	Ratio=0.4	Ratio=0.5
OLE	0.910	0.860	0.806	0.717	0.610
LargeMargin	0.901	0.874	0.837	0.785	0.724
MCR^2	**0.911**	**0.897**	**0.881**	**0.866**	**0.843**
WRN16	Ratio=0.1	Ratio=0.3	Ratio=0.5	Ratio=0.7	
ITLM	0.903	0.882	0.825	0.647	
MCR^2	**0.915**	**0.888**	**0.842**	**0.670**	

Table 8: Comparison with related work (OLE (Lezama et al., 2018), LargeMargin (Elsayed et al., 2018), ITLM (Shen and Sanghavi, 2019)) on learning from noisy labels.

D.5 Learning from Gaussian noise corrupted data.

We investigate the performance of MCR^2 training with corrupted data by adding varying levels of Gaussian noise. For each corruption level, we add $\mathcal{N}(0, \sigma^2)$ to the input images with different standard deviations $\sigma \in \{0.04, 0.06, 0.08, 0.09, 0.1\}$ as in Hendrycks and Dietterich (2018). We train the same architecture ResNet-18 as the previous experiments for 500 epochs, set mini-batch size to $m = 1000$ and optimize using SGD with learning rate `lr=0.01`, momentum `momentum=0.9` and weight decay `wd=5e-4`. We also decrease the learning rate to 0.001 at epoch 200 and to 0.0001 at epoch 400. In our objective, we set precision $\epsilon^2 = 0.5$. To compare the performance of MCR^2 versus cross-entropy (CE), we train the same architecture using the cross-entropy loss for 200 epochs and optimize using SGD with learning rate `lr=0.1`, momentum `momentum=0.9` and weight decay `wd=5e-4`. We also use Cosine Annealing learning rate scheduler during training. We show the respective testing accuracy in Table 9. Although the classification result of using MCR^2 slightly lags behind that of using CE, when the noise level is small, their performances are comparable with each other. Similar sensitivity to the noise indicates that the reason might be because of the

choice of the same network architecture. We reserve the study on how to improve robustness to input noise for future work.

Noise level	$\sigma = 0.04$	$\sigma = 0.06$	$\sigma = 0.08$	$\sigma = 0.09$	$\sigma = 0.1$
CE Training	0.912	0.897	0.876	0.867	0.857
MCR^2 Training	0.909	0.882	0.869	0.855	0.829

Table 9: Classification results of features learned with inputs corrupted by Gaussian noise at different levels.

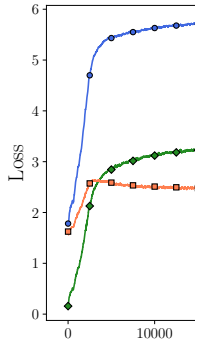
D.6 Experimental Results of MCR^2 in the Self-supervised Learning Setting

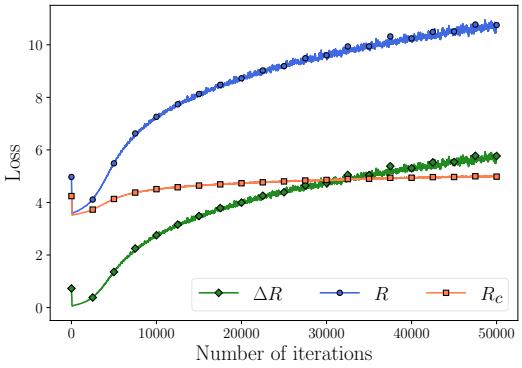
D.6.1 Self-supervised Learning of Invariant Features

Learning invariant features via rate reduction. Motivated by self-supervised learning algorithms (LeCun et al., 2004; Kavukcuoglu et al., 2009; Oord et al., 2018; He et al., 2020; Wu et al., 2018), we use the MCR^2 principle to learn representations that are *invariant* to certain class of transformations/augmentations, say $\mathcal{T}$ with a distribution $P_{\mathcal{T}}$. Given a mini-batch of data $\{\boldsymbol{x}^j\}_{j=1}^k$ with mini-batch size equals to k, we augment each sample $\boldsymbol{x}^j$ with n transformations/augmentations $\{\tau^i(\cdot)\}_{i=1}^n$ randomly drawn from $P_{\mathcal{T}}$. We simply label all the augmented samples $\boldsymbol{X}^j = [\tau_1(\boldsymbol{x}^j), \ldots, \tau_n(\boldsymbol{x}^j)]$ of $\boldsymbol{x}^j$ as the j-th class, and $\boldsymbol{Z}^j$ the corresponding learned features. Using this self-labeled data, we train our feature mapping $f(\cdot, \boldsymbol{\theta})$ the same way as the supervised setting above. For every mini-batch, the total number of samples for training is $m = kn$.

Evaluation via clustering. To learn invariant features, our formulation itself does *not* require the original samples $\boldsymbol{x}^j$ come from a fixed number of classes. For evaluation, we may train on a few classes and observe how the learned features facilitate classification or clustering of the data. A common method to evaluate learned features is to train an additional linear classifier (Oord et al., 2018; He et al., 2020), with ground truth labels. But for our purpose, because we explicitly verify whether the so-learned invariant features have good subspace structures when the samples come from k classes, we use an off-the-shelf subspace clustering algorithm EnSC (You et al., 2016), which is computationally efficient and is provably correct for data with well-structured subspaces. We also use K-Means on the original data $\boldsymbol{X}$ as our baseline for comparison. We use normalized mutual information (NMI), clustering accuracy (ACC), and adjusted rand index (ARI) for our evaluation metrics, see Appendix D.6.2 for their detailed definitions.

Controlling dynamics of expansion and compression. By directly optimizing the rate reduction $\Delta R = R - R_c$, we achieve 0.570 clustering accuracy on CIFAR10 dataset, which is the second best result compared with previous methods. Empirically, we observe that, without class labels, the overall *coding rate* R expands quickly and the MCR^2 loss saturates (at a local maximum), see Fig 21(a). Our experience suggests that learning a good representation from unlabeled data might be too ambitious when directly optimizing the original ΔR. Nonetheless, from the geometric meaning of R and R_c, one can design a different learning strategy by controlling the dynamics of expansion and compression





(b) MCR^2-CTRL.

Figure 21: Evolution of the rates of (**left**) MCR^2 and (**right**) MCR^2-CTRL in the training process in the self-supervised setting on CIFAR10 dataset.

differently during training. For instance, we may re-scale the rate by replacing $R(\boldsymbol{Z}, \epsilon)$ with

$$\widetilde{R}(\boldsymbol{Z}, \epsilon) \doteq \frac{1}{2\gamma_1} \log\det(\boldsymbol{I} + \frac{\gamma_2 d}{m\epsilon^2} \boldsymbol{Z}\boldsymbol{Z}^*). \tag{152}$$

With $\gamma_1 = \gamma_2 = k$, the learning dynamics change from Figure 21(a) to Figure 21(b): All features are first compressed then gradually expand. We denote the controlled MCR^2 training by MCR^2-CTRL.

Experiments on real data. Similar to the supervised learning setting, we train *exactly the same* ResNet-18 network on the CIFAR10, CIFAR100, and STL10 (Coates et al., 2011) datasets. We set the mini-batch size as $k = 20$, number of augmentations for each sample as $n = 50$ and the precision parameter as $\epsilon^2 = 0.5$. Table 10 shows the results of the proposed MCR^2-CTRL in comparison with methods JULE (Yang et al., 2016), RTM (Nina et al., 2019), DEC (Xie et al., 2016), DAC (Chang et al., 2017), and DCCM (Wu et al., 2019) that have achieved the best results on these datasets. Surprisingly, without utilizing any inter-class or inter-sample information and heuristics on the data, the invariant features learned by our method with augmentations alone achieves a better performance over other highly engineered clustering methods. More comparisons and ablation studies can be found in Appendix D.6.2.

Nevertheless, compared to the representations learned in the supervised setting where the optimal partition $\boldsymbol{\Pi}$ in (11) is initialized by correct class information, the representations here learned with self-supervised classes are far from being optimal. It remains wide open how to design better optimization strategies and dynamics to learn from unlabelled or partially-labelled data better representations (and the associated partitions) close to the global maxima of the MCR^2 objective (11).

Training details of MCR^2-CTRL. For three datasets (CIFAR10, CIFAR100, and STL10), we use ResNet-18 as in the supervised setting, and we set the output dimension $d = 128$, precision $\epsilon^2 = 0.5$, mini-batch size $k = 20$, number of augmentations $n = 50$, $\gamma_1 = \gamma_2 = 20$. We observe that MCR^2-CTRL can achieve better clustering performance by using smaller γ_2,

Dataset	Metric	K-Means	JULE	RTM	DEC	DAC	DCCM	MCR^2-Ctrl
CIFAR10	NMI	0.087	0.192	0.197	0.257	0.395	0.496	**0.630**
	ACC	0.229	0.272	0.309	0.301	0.521	0.623	**0.684**
	ARI	0.049	0.138	0.115	0.161	0.305	0.408	**0.508**
CIFAR100	NMI	0.084	0.103	-	0.136	0.185	0.285	**0.387**
	ACC	0.130	0.137	-	0.185	0.237	0.327	**0.375**
	ARI	0.028	0.033	-	0.050	0.087	0.173	**0.178**
STL10	NMI	0.124	0.182	-	0.276	0.365	0.376	**0.446**
	ACC	0.192	0.182	-	0.359	0.470	0.482	**0.491**
	ARI	0.061	0.164	-	0.186	0.256	0.262	**0.290**

Table 10: Clustering results on CIFAR10, CIFAR100, and STL10 datasets.

i.e., $\gamma_2 = 15$, on CIFAR10 and CIFAR100 datasets. We use SGD as the optimizer, and set the learning rate `lr=0.1`, weight decay `wd=5e-4`, and `momentum=0.9`.

Training dynamic comparison between MCR^2 and MCR^2-ctrl. In the self-supervised setting, we compare the training process for MCR^2 and MCR^2-ctrl in terms of $R, \widetilde{R}, R_c$, and ΔR. For MCR^2 training shown in Figure 21(a), the features first expand (for both R and R_c) then compress (for R_c). For MCR^2-ctrl, both $\widetilde{R}$ and R_c first compress then $\widetilde{R}$ expands quickly and R_c remains small, as we have seen in Figure 21(b).

Clustering results comparison. We compare the clustering performance between MCR^2 and MCR^2-ctrl in terms of NMI, ACC, and ARI. The clustering results are summarized in Table 11. We find that MCR^2-ctrl can achieve better performance for clustering.

	NMI	ACC	ARI
MCR^2	0.544	0.570	0.399
MCR^2-Ctrl	0.630	0.684	0.508

Table 11: Clustering comparison between MCR^2 and MCR^2-ctrl on CIFAR10 dataset.

D.6.2 Clustering Metrics and More Results

We first introduce the definitions of normalized mutual information (NMI) (Strehl and Ghosh, 2002), clustering accuracy (ACC), and adjusted rand index (ARI) (Hubert and Arabie, 1985).

Normalized mutual information (NMI). Suppose Y is the ground truth partition and C is the prediction partition. The NMI metric is defined as

$$\mathrm{NMI}(Y, C) = \frac{\sum_{i=1}^{k}\sum_{j=1}^{s} |Y_i \cap C_j| \log\left(\frac{m|Y_i \cap C_j|}{|Y_i||C_j|}\right)}{\sqrt{\left(\sum_{i=1}^{k} |Y_i| \log\left(\frac{|Y_i|}{m}\right)\right)\left(\sum_{j=1}^{s} |C_j| \log\left(\frac{|C_j|}{m}\right)\right)}},$$

where Y_i is the i-th cluster in Y and C_j is the j-th cluster in C, and m is the total number of samples.

Clustering accuracy (ACC). Given m samples, $\{(\boldsymbol{x}^i, \boldsymbol{y}^i)\}_{i=1}^m$. For the i-th sample $\boldsymbol{x}^i$, let $\boldsymbol{y}^i$ be its ground truth label, and let $\boldsymbol{c}^i$ be its cluster label. The ACC metric is defined as

$$\mathrm{ACC}(\boldsymbol{Y}, \boldsymbol{C}) = \max_{\sigma \in S} \frac{\sum_{i=1}^m \mathbf{1}\{\boldsymbol{y}^i = \sigma(\boldsymbol{c}^i)\}}{m},$$

where S is the set includes all the one-to-one mappings from cluster to label, and $\boldsymbol{Y} = [\boldsymbol{y}^1, \ldots, \boldsymbol{y}^m]$, $\boldsymbol{C} = [\boldsymbol{c}^1, \ldots, \boldsymbol{c}^m]$.

Adjusted rand index (ARI). Suppose there are m samples, and let Y and C be two clustering of these samples, where $Y = \{Y_1, \ldots, Y_r\}$ and $C = \{C_1, \ldots, C_s\}$. Let m_{ij} denote the number of the intersection between Y_i and C_j, i.e., $m_{ij} = |Y_i \cap C_j|$. The ARI metric is defined as

$$\mathrm{ARI} = \frac{\sum_{ij} \binom{m_{ij}}{2} - \left(\sum_i \binom{a_i}{2} \sum_j \binom{b_j}{2}\right) / \binom{m}{2}}{\frac{1}{2}\left(\sum_i \binom{a_i}{2} + \sum_j \binom{b_j}{2}\right) - \left(\sum_i \binom{a_i}{2} \sum_j \binom{b_j}{2}\right) / \binom{m}{2}},$$

where $a_i = \sum_j m_{ij}$ and $b_j = \sum_i m_{ij}$.

Comparison with Ji et al. (2019); Hu et al. (2017). We compare MCR^2 with IIC (Ji et al., 2019) and IMSAT (Hu et al., 2017) in Table 12. We find that MCR^2 outperforms IIC (Ji et al., 2019) and IMSAT (Hu et al., 2017) on both CIFAR10 and CIFAR100 by a large margin. For STL10, Hu et al. (2017) applied pretrained ImageNet models and Ji et al. (2019) used more data for training.

Dataset	Metric	IIC	IMSAT	MCR^2-Ctrl
CIFAR10	NMI	-	-	**0.630**
	ACC	0.617	0.456	**0.684**
	ARI	-	-	**0.508**
CIFAR100	NMI	-	-	**0.387**
	ACC	0.257	0.275	**0.375**
	ARI	-	-	**0.178**

Table 12: Compare with Ji et al. (2019); Hu et al. (2017) on clustering.

More experiments on the effect of hyperparameters of MCR^2-CTRL. We provide more experimental results of MCR^2-CTRL training in the self-supervised setting by varying training hyperparameters on the STL10 dataset. The results are summarized in Table 13. Notice that the choice of hyperparameters only has small effect on the performance with the MCR^2-CTRL objective. We may hypothesize that, in order to further improve the performance, one has to seek other, potentially better, control of optimization dynamics or strategies. We leave those for future investigation.

Arch	Precision ϵ^2	Learning Rate lr	NMI	ACC	ARI
ResNet-18	0.5	0.1	0.446	0.491	0.290
ResNet-18	0.75	0.1	0.450	0.484	0.288
ResNet-18	0.25	0.1	0.447	0.489	0.293
ResNet-18	0.5	0.2	0.477	0.473	0.295
ResNet-18	0.5	0.05	0.444	0.496	0.293
ResNet-18	0.25	0.05	0.454	0.489	0.294

Table 13: Experiments of MCR^2-ctrl in the self-supervised setting on STL10 dataset.

Appendix E. Implementation Details and Additional Experiments for ReduNets

In this section, we provide additional experimental results related to ReduNet in Section 5. Obviously, in this work we have chosen a simplest design of the ReduNet and do not particularly optimize any of the hyper parameters, such as the number of initial channels, kernel sizes, normalization, and learning rate etc., for the best performance or scalability. The choices are mostly for convenience and just minimally adequate to verify the concept. We leave all such practical matters for us and others to investigate in the future.

We first present the visualization of rotated and translated images of the MNIST dataset in Section E.1. We provide additional experimental results on rotational invariance in Section E.2 and translational invariance in Section E.3. In Section E.4, we provide more results on learning mixture of Gaussians.

E.1 Visualization of Rotation and Translation on MNIST

In this subsection, we present the visualization of rotation and translation images on the MNIST dataset. The rotation examples are shown in Figure 22 and the translation examples are shown in Figure 23.

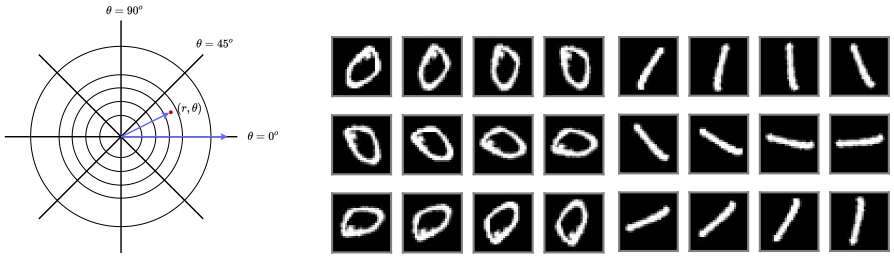


Figure 22: Examples of rotated images of MNIST digits, each by 18°. (**Left**) Diagram for polar coordinate representation; (**Right**) Rotated images of digit '0' and '1'.

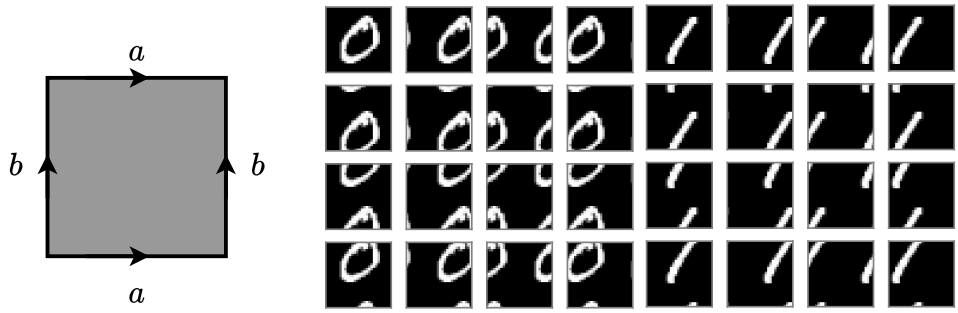


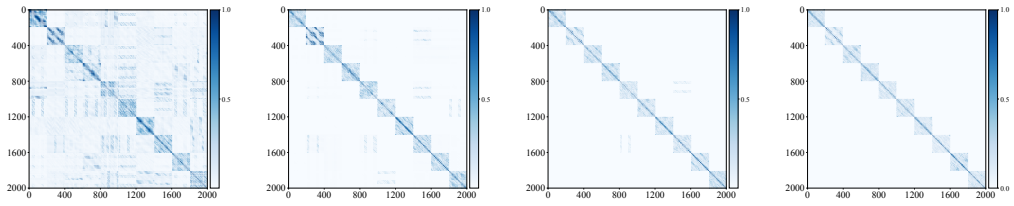
Figure 23: (**Left**) A torus on which 2D cyclic translation is defined; (**Right**) Cyclic translated images of MNIST digits '0' and '1' (with stride=7).

E.2 Additional Experiments on Learning Rotational Invariance on MNIST

Effect of channel size. We study the effect of the number of channels on the rotation invariant task on the MNIST dataset. We apply the same parameters as in Figure 16(b) except that we vary the channel size from 5 to 20, where the number of channels used in Figure 16(b) is 20. We summarize the results in Table 14, Figure 24 and Figure 25. From Table 14, we find that the invariance training accuracy increases when we increase the channel size. As we can see from Figure 24, the shifted learned features become more orthogonal across different classes when the channel size is large. Also, we observe that with more channel size, the training loss converges faster (in Figure 25).

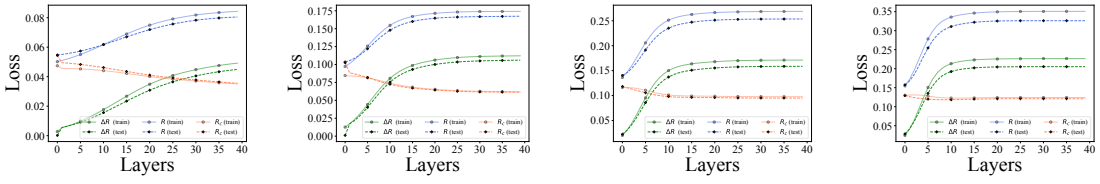
Channel	5	10	15	20
Training Acc	1.000	1.000	1.000	1.000
Test Acc	0.560	0.610	0.610	0.610
Invariance Training Acc	0.838	0.972	0.990	1.000
Invariance Test Acc	0.559	0.609	0.610	0.610

Table 14: Training accuracy of 1D rotation-invariant ReduNet with different number of channels on the MNIST dataset.



(a) 5 Channels (b) 10 Channels (c) 15 Channels (d) 20 Channels

Figure 24: Heatmaps of cosine similarity among shifted learned features (with different channel sizes) $\bar{\boldsymbol{Z}}_{\text{shift}}$ for rotation invariance on the MNIST dataset (RI-MNIST).



(a) 5 Channels (b) 10 Channels (c) 15 Channels (d) 20 Channels

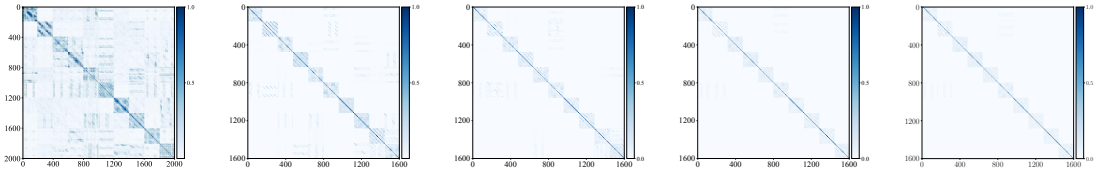
Figure 25: Training and test losses of rotational invariant ReduNet for rotation invariance on the MNIST dataset (RI-MNIST).

E.3 Additional Experiments on Learning 2D Translation Invariance on MNIST

Effect of channel size. We study the effect of the number of channels on the translation invariant task on the MNIST dataset. We use the same parameters as in Figure 16(f) except that we vary the channel size from 5 to 75, where the number of channels used in Figure 16(f) is 75. Similar to the observations in Section E.2, in Table 15, we observe that both the invariance training accuracy and invariance test accuracy increase when we increase the channel size. From Figure 26 and Figure 27, we find that when we increase the number of channels, the shifted learned features become more orthogonal across different classes and the training loss converges faster.

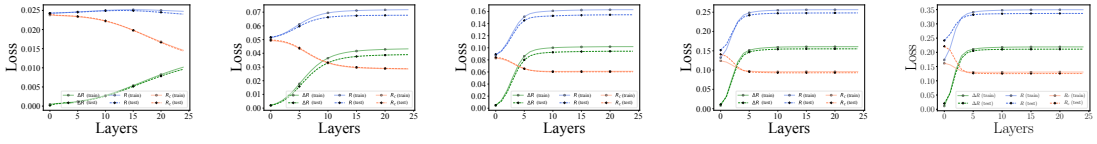
Channel	5	15	35	55	75
Training Acc	1.000	1.000	1.000	1.000	1.000
Test Acc	0.680	0.610	0.670	0.770	0.840
Invariance Training Acc	0.879	0.901	0.933	0.933	0.976
Invariance Test Acc	0.619	0.599	0.648	0.767	0.838

Table 15: Training/test accuracy of 2D translation-invariant ReduNet with different number of channels on the MNIST dataset.



(a) 5 Channels (b) 15 Channels (c) 35 Channels (d) 55 Channels (e) 75 Channels

Figure 26: Heatmaps of cosine similarity among shifted learned features (with different channel sizes) $\bar{\boldsymbol{Z}}_{\text{shift}}$ for translation invariance on the MNIST dataset (TI-MNIST).

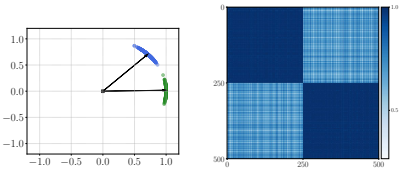


(a) 5 Channels (b) 15 Channels (c) 35 Channels (d) 55 Channels (e) 75 Channels

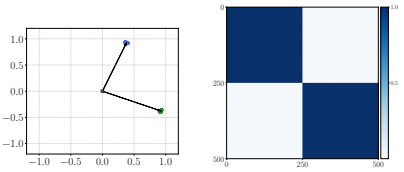
Figure 27: Training and test losses of translation invariant ReduNet for translation invariance on the MNIST dataset (TI-MNIST).

E.4 Additional Experiments on Learning Mixture of Gaussians in $\mathbb{S}^1$ and $\mathbb{S}^2$

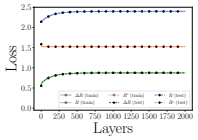
Additional experiments on $\mathbb{S}^1$ and $\mathbb{S}^2$. We also provide additional experiments on learning mixture of Gaussians in $\mathbb{S}^1$ and $\mathbb{S}^2$ in Figure 28. We can observe similar behavior of the proposed ReduNet: the network can map data points from different classes to orthogonal subspaces.



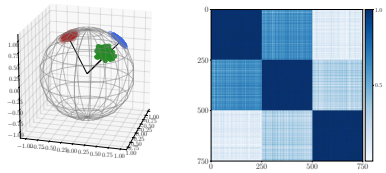
(a) $\boldsymbol{X}(2D)$ (**left:** scatter plot; **right:** cosine similarity visualization)



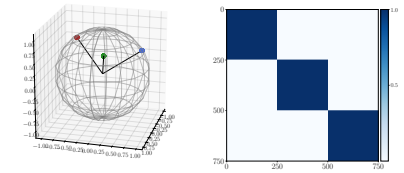
(b) $\boldsymbol{Z}(2D)$ (**left:** scatter plot; **right:** cosine similarity visualization)



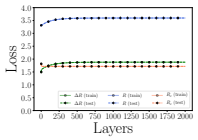
(c) Loss



(d) $\boldsymbol{X}(3D)$ (**left:** scatter plot; **right:** cosine similarity visualization)



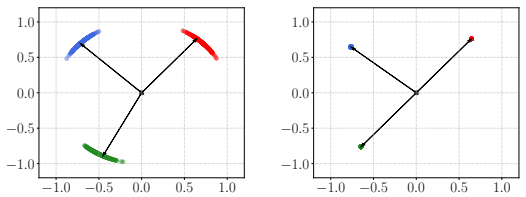
(e) $\boldsymbol{Z}(3D)$ (**left:** scatter plot; **right:** cosine similarity visualization)



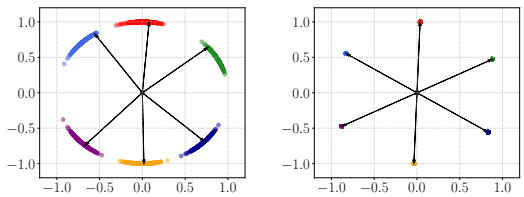
(f) Loss

Figure 28: Learning mixture of Gaussians in $\mathbb{S}^1$ and $\mathbb{S}^2$. (**Top**) For $\mathbb{S}^1$, we set $\sigma_1 = \sigma_2 = 0.1$; (**Bottom**) For $\mathbb{S}^2$, we set $\sigma_1 = \sigma_2 = \sigma_3 = 0.1$.

Additional experiments on $\mathbb{S}^1$ with more than 2 classes. We try to apply ReduNet to learn mixture of Gaussian distributions on $\mathbb{S}^1$ with the number of class is larger than 2. Notice that these are the cases to which Theorem 1 no longer applies. These experiments suggest that the MCR^2 still promotes between-class discriminativeness with so constructed ReduNet. In particular, the case on the left of Figure 29 indicates that the ReduNet has "merged" two linearly correlated clusters into one on the same line. This is consistent with the objective of rate reduction to group data as linear subspaces.



(a) 3 classes. (**Left**) $\boldsymbol{X}$; (**Right**) $\boldsymbol{Z}$



(b) 6 classes. (**Left**) $\boldsymbol{X}$; (**Right**) $\boldsymbol{Z}$

Figure 29: Learning mixture of Gaussian distributions with more than 2 classes. For both cases, we use step size $\eta = 0.5$ and precision $\epsilon = 0.1$. For (a), we set iteration $L = 2,500$; for (b), we set iteration $L = 4,000$.

E.5 Additional Experimental Results of ReduNet and Scattering Transform

Comparing ResNet and ReduNet on maximizing ΔR using CIFAR10. We compare the objective value reached by training a ResNet-18 versus constructing a ReduNet that maximizes the MCR^2 objective in an iterative fashion. For training ResNet-18, we use the same setting as Section 5.1, with mini-batch size $m = 1000$ and precision $\epsilon^2 = 0.5$. For training ReduNet, we first apply scattering transform to the training samples, then flatten the scatter-transformed features and project them to a 128-dimensional feature vector using a random linear projection. Then we construct a 4000-layer ReduNet with precision $\epsilon^2 = 0.1$, step size $\eta = 0.5$ and $\lambda = 500$. We vary the scales $\{2^3, 2^4\}$ and angles $\{4, 8, 16\}$ in scattering transforms, but set the phase to 4 for all sets of experiments. In both cases, we use all 50000 training samples. Our results are listed in Table 16. Empirically, features learned by training a ResNet-18 with MCR^2 achieves a higher ΔR of 48.65 than those learned by constructing a ReduNet, achieving a ΔR of 46.14.

Scales	Angles	ΔR	R	R_c
2^3	4	46.1418	66.0627	19.9208
2^3	8	46.3207	66.1358	19.8151
2^3	16	46.4162	66.1474	19.7312
2^4	4	40.5377	58.8128	18.2751
2^4	8	41.3374	60.1895	18.8521
2^4	16	42.5311	61.5164	18.9854
ResNet-18		48.6497	69.2653	20.6155

Table 16: Objective values under scattering transform with varying scales and angles. Each model is evaluated on the 50,000 training samples from the CIFAR10 dataset.

Comparing ReduNet with random filters and scattering transform. Here we provide comparisons of using ReduNet on MNIST digits that are lifted by scattering transform versus by random filters. We select $m = 5000$ training samples (500 from each class), and set precision $\epsilon^2 = 0.1$ and step size $\eta = 0.1$. We vary the number of layers for each comparison, as the numbers of layers needed for the rate reduction objective to converge also varies. For each comparison, we construct two networks: 1). We apply scattering transform to each training sample, then flatten the feature vector and construct a ReduNet, and 2). We apply a number random filters, such that the resulting dimensions of the feature are the same. We ablate across different scales $\{2^3, 2^4\}$ and different angles $\{4, 8, 16\}$, but keep the number of phases to 4 the same. To compute the testing accuracy of each architecture, we evaluate all 10000 testing samples in the MNIST dataset using a nearest subspace classifier. We provide details about the feature dimensions and test accuracies in Table 17. For comparison, we also provide the test accuracy of linear SVM classifiers learned on scatter-transformed features (without using ReduNet) in Table 18. From these empirical results, we have shown that using scattering transform outperforms using random filters in all scenarios. This implies that scattering transform is better at making the original data samples more separable than the random filters. With a more suitable lifting operator, ReduNet is able to linearize

low-dimensional structure into incoherent subspaces. This also showcases that scattering transform and ReduNet complement each other: scatter transform learns separable features and ReduNet learns diverse and discrminative features. In comparing performance of using scattering-transformed features with a ReduNet versus a linear SVM classifier, we observe that the benefit of ReduNet in terms of test accuracy is prominent when the dimension of the features is relatively small (e.g., with $n = 65$, 129, and 257). This can be attributed to the fact that different classes are not well separated in the relatively low-dimensional feature space, hence ReduNet can help to learn better discriminative features and improve the classification performance. On the other hand, the difference between ReduNet and linear SVM is very small when the dimension of the features is relatively large (e.g., with $n = 441$, 873, and 1737). This could because the features are already discriminative in the high-dimensional space. Therefore, a linear SVM model is sufficient for separating different classes.

Lifting + Classifier	n	L	Acc
Random filters + ReduNet	441	150	0.9225
Scattering (2^3 scales, 4 angles) + Linear SVM	441	-	**0.9795**
Scattering (2^3 scales, 4 angles) + ReduNet	441	150	0.9734
Random filters + ReduNet	873	100	0.9153
Scattering (2^3 scales, 8 angles) + Linear SVM	873	-	**0.9839**
Scattering (2^3 scales, 8 angles) + ReduNet	873	100	0.9781
Random filters + ReduNet	1737	50	0.9061
Scattering (2^3 scales, 16 angles) + Linear SVM	1737	-	**0.9859**
Scattering (2^3 scales, 16 angles) + ReduNet	1737	50	0.9795
Random filters + ReduNet	65	400	0.8794
Scattering (2^4 scales, 4 angles) + Linear SVM	65	-	0.8829
Scattering (2^4 scales, 4 angles) + ReduNet	65	400	**0.8952**
Random filters + ReduNet	129	300	0.9172
Scattering (2^4 scales, 8 angles) + Linear SVM	129	-	0.9205
Scattering (2^4 scales, 8 angles) + ReduNet	129	300	**0.9392**
Random filters + ReduNet	257	200	0.8794
Scattering (2^4 scales, 16 angles) + Linear SVM	257	-	0.9337
Scattering (2^4 scales, 16 angles) + ReduNet	257	200	**0.9569**

Table 17: Test accuracy of ReduNet with lifting by scattering transform versus by random filters on the MNIST dataset. The dimension of the feature vector n and the number of layers used in the ReduNet L are also stated. For comparison, we also report the classification performance of scattering-transformed features with a linear SVM classifier.

(a) Class 3 (b) Class 7

Figure 30: Verification of Equivariance in ReduNet. For each class of samples, and for all training samples (and their augmentations) $\boldsymbol{z}^i$ and a test sample (and its augmentations) $\hat{\boldsymbol{z}}$, we compute their inner product $|\langle \hat{\boldsymbol{z}}, \boldsymbol{z}^i \rangle|$, $\forall i$. We select the top-9 largest inner products (sorted from left to right), and visualize their respective image. The test sample and its augmented version are highlighted in red.

E.6 Equivariance of learned features using Translation-Invariant ReduNet

We investigate how features learned using Translational-Invariant ReduNet possess equivariant properties. More specifically, we construct a ReduNet using 500 training samples of MNIST digits (50 from each class) with number of layers $L = 30$, precision $\epsilon^2 = 0.1$, step size $\eta = 0.5$, and $\lambda = 500$. We also apply 2D circulant convolution to the (1-channel) inputs with 16 random Gaussian kernels of size 7×7.

To evaluate its equivariant properties, we augment each training and test sample by shifting 7 pixels in each canonical direction, resulting in 9 augmented images for each original image. As shown in Figure 30, we compute the distance between the representation of a (shifted) test sample and representations of training samples (including all translation augmentations) using cosine similarity. By computing the top-9 largest inner products, we observe that each augmented test sample is closest to a training sample with the similar translation.

E.7 Effect of hyperparameters on ReduNet on MNIST

We perform an ablation study on precision ϵ^2 and step size η parameter of ReduNet on MNIST. We flatten the image from MNIST into a feature vector, and directly apply ReduNet with varying precision ϵ^2 and step size η, and fixed $\lambda = 500$ and number of layers $L = 100$. For each ReduNet, we use the same set of $m = 5000$ training samples (500 from each

class), and evaluate its generalization performance on 10,000 test samples using the nearest subspace classifier. As a baseline, we train a linear SVM classifier with the same training samples and evaluate its performance on the same testing samples. As shown in Table 18, while both methods are able to achieve near-perfect training accuracy, ReduNet achieves better performance than linear SVM. We also find that ReduNet is stable to the choice of step size η, while precision ϵ^2 plays an important role in the generalization performance of ReduNet.

Layers (L)	Precision (ϵ^2)	Step size (η)	Train Acc.	Test Acc.
100	0.1	0.1	0.9988	0.9186
100	0.1	0.5	0.9998	0.9184
100	0.1	1.0	0.9999	0.9184
100	0.5	0.1	0.9986	0.9392
100	0.5	0.5	0.9986	0.9393
100	0.5	1.0	0.9986	0.9393
100	1.0	0.1	0.9874	0.9500
100	1.0	0.5	0.9968	0.9476
100	1.0	1.0	0.9968	0.9476
Linear SVM	-	-	0.9986	0.8463

Table 18: Ablation of precision ϵ^2 and step size η of ReduNet on MNIST. Each ReduNet is constructed with 5,000 MNIST flattened training samples. For comparison, we evaluate the performance of the linear SVM classifier trained on the same training samples.

References

Mongi A Abidi, Andrei V Gribok, and Joonki Paik. *Optimization Techniques in Computer Vision*. Springer, 2016.

Zeyuan Allen-Zhu, Yuanzhi Li, and Zhao Song. A convergence theory for deep learning via over-parameterization. In *ICML*, 2019.

Marcin Andrychowicz, Misha Denil, Sergio Gomez, Matthew W Hoffman, David Pfau, Tom Schaul, Brendan Shillingford, and Nando De Freitas. Learning to learn by gradient descent by gradient descent. In *NeurIPS*, 2016.

Sanjeev Arora, Nadav Cohen, Wei Hu, and Yuping Luo. Implicit regularization in deep matrix factorization. In *NeurIPS*, 2019.

Aharon Azulay and Yair Weiss. Why do deep convolutional networks generalize so poorly to small image transformations? *JMLR*, 2019.

Jimmy Lei Ba, Jamie Ryan Kiros, and Geoffrey E Hinton. Layer normalization. *arXiv:1607.06450*, 2016.

Christina Baek, Ziyang Wu, Tianjiao Ding Ryan Chan, and Yi Maand Benjamin Haeffele. Efficient maximal coding rate reduction by variational form. In *CVPR*, 2022.

Bowen Baker, Otkrist Gupta, N. Naik, and R. Raskar. Designing neural network architectures using reinforcement learning. In *ICLR*, 2017.

Pierre Baldi and Kurt Hornik. Neural networks and principal component analysis: Learning from examples without local minima. *Neural networks*, 1989.

Andrew R Barron. Approximation and estimation bounds for artificial neural networks. *Machine Learning*, 1991.

Peter L. Bartlett, Dylan J. Foster, and Matus J. Telgarsky. Spectrally-normalized margin bounds for neural networks. In *NeurIPS*, 2017.

Amir Beck and Marc Teboulle. A fast iterative shrinkage-thresholding algorithm for linear inverse problems. *SIAM Journal on Imaging Sciences*, 2009.

Andrei Belitski, Arthur Gretton, Cesare Magri, Yusuke Murayama, Marcelo A. Montemurro, Nikos K. Logothetis, and Stefano Panzeri. Low-frequency local field potentials and spikes in primary visual cortex convey independent visual information. *Journal of Neuroscience*, 2008.

Mikhail Belkin, Daniel Hsu, Siyuan Ma, and Soumik Mandal. Reconciling modern machine-learning practice and the classical bias–variance trade-off. *PNAS*, 2019.

Stephen P. Boyd and Lieven Vandenberghe. *Convex optimization*. Cambridge university press, 2004.

Michael M. Bronstein, Joan Bruna, Yann LeCun, Arthur Szlam, and Pierre Vandergheynst. Geometric deep learning: Going beyond euclidean data. *IEEE Signal Processing Magazine*, 2017.

Joan Bruna and Stéphane Mallat. Invariant scattering convolution networks. *PAMI*, 2013.

Sam Buchanan, Dar Gilboa, and John Wright. Deep networks and the multiple manifold problem. In *ICLR*, 2021.

Jacopo Cavazza, Pietro Morerio, Benjamin Haeffele, Connor Lane, Vittorio Murino, and Rene Vidal. Dropout as a low-rank regularizer for matrix factorization. In *AISTATS*, 2018.

Tsung-Han Chan, Kui Jia, Shenghua Gao, Jiwen Lu, Zinan Zeng, and Yi Ma. PCANet: A simple deep learning baseline for image classification? *TIP*, 2015.

Jianlong Chang, Lingfeng Wang, Gaofeng Meng, Shiming Xiang, and Chunhong Pan. Deep adaptive image clustering. In *ICCV*, 2017.

Le Chang and Doris Tsao. The code for facial identity in the primate brain. *Cell*, 2017.

Tian Qi Chen, Yulia Rubanova, Jesse Bettencourt, and David K Duvenaud. Neural ordinary differential equations. In *NeurIPS*, 2018a.

Yubei Chen, Dylan Paiton, and Bruno Olshausen. The sparse manifold transform. In *NeurIPS*, 2018b.

Francois Chollet. Xception: Deep learning with depthwise separable convolutions. In *CVPR*, 2017.

Adam Coates, Andrew Ng, and Honglak Lee. An analysis of single-layer networks in unsupervised feature learning. In *AISTATS*, 2011.

Taco Cohen and Max Welling. Group equivariant convolutional networks. In *ICML*, 2016.

Taco Cohen, Mario Geiger, and Maurice Weiler. A general theory of equivariant CNNs on homogeneous spaces. In *NeurIPS*, 2019.

Thomas M. Cover and Joy A. Thomas. *Elements of Information Theory*. Wiley-Interscience, 2006.

Xili Dai, Shengbang Tong, Mingyang Li, Ziyang Wu, Michael Psenka, Kwan Ho Ryan Chan, Pengyuan Zhai, Yaodong Yu, Xiaojun Yuan, Heung Yeung Shum, and Yi Ma. Closed-loop data transcription to an ldr via minimaxing rate reduction. *arXiv:2111.06636*, 2021.

Matthias Delange, Rahaf Aljundi, Marc Masana, Sarah Parisot, Xu Jia, Ales Leonardis, Greg Slabaugh, and Tinne Tuytelaars. A continual learning survey: Defying forgetting in classification tasks. *PAMI*, 2021.

Simon S. Du, Xiyu Zhai, Barnabas Poczos, and Aarti Singh. Gradient descent provably optimizes over-parameterized neural networks. In *ICLR*, 2019.

Chris Eliasmith and Charles Anderson. *Neural Engineering: Computation, Representation and Dynamics in Neurobiological Systems*. MIT press, 2003.

Gamaleldin Elsayed, Dilip Krishnan, Hossein Mobahi, Kevin Regan, and Samy Bengio. Large margin deep networks for classification. In *NeurIPS*, 2018.

Logan Engstrom, Brandon Tran, Dimitris Tsipras, Ludwig Schmidt, and Aleksander Madry. Exploring the landscape of spatial robustness. In *ICML*, 2019.

Cong Fang, Hangfeng He, Qi Long, and Weijie J Su. Layer-peeled model: Toward understanding well-trained deep neural networks. *PNAS*, 2021.

Maryam Fazel, Haitham Hindi, and Stephen P. Boyd. Log-det heuristic for matrix rank minimization with applications to Hankel and Euclidean distance matrices. In *ACC*, 2003.

William Fedus, Barret Zoph, and Noam Shazeer. Switch transformers: Scaling to trillion parameter models with simple and efficient sparsity. *arXiv:2101.03961*, 2021.

Gauthier Gidel, Francis Bach, and Simon Lacoste-Julien. Implicit regularization of discrete gradient dynamics in linear neural networks. In *NeurIPS*, 2019.

Raja Giryes, Yonina C Eldar, Alex M Bronstein, and Guillermo Sapiro. Tradeoffs between convergence speed and reconstruction accuracy in inverse problems. *IEEE Transactions on Signal Processing*, 2018.

Noah Golowich, Alexander Rakhlin, and Ohad Shamir. Size-independent sample complexity of neural networks. In *COLT*, 2018.

Ian Goodfellow, Yoshua Bengio, Aaron Courville, and Yoshua Bengio. *Deep learning*. MIT Press, 2016.

Priya Goyal, Piotr Dollár, Ross Girshick, Pieter Noordhuis, Lukasz Wesolowski, Aapo Kyrola, Andrew Tulloch, Yangqing Jia, and Kaiming He. Accurate, large minibatch sgd: Training imagenet in 1 hour. *arXiv:1706.02677*, 2017.

Karol Gregor and Yann LeCun. Learning fast approximations of sparse coding. In *ICML*, 2010.

Suriya Gunasekar, Jason D. Lee, Daniel Soudry, and Nati Srebro. Implicit bias of gradient descent on linear convolutional networks. In *NeurIPS*, 2018.

Raia Hadsell, Sumit Chopra, and Yann LeCun. Dimensionality reduction by learning an invariant mapping. In *CVPR*, 2006.

Benjamin David Haeffele, Chong You, and Rene Vidal. A critique of self-expressive deep subspace clustering. In *ICLR*, 2021.

X.Y. Han, Vardan Papyan, and David L Donoho. Neural collapse under mse loss: Proximity to and dynamics on the central path. *arXiv:2106.02073*, 2021.

Boris Hanin. Universal function approximation by deep neural nets with bounded width and ReLU activations. *arXiv:1708.02691*, 2017.

Trevor Hastie, Robert Tibshirani, and Jerome Friedman. *The Elements of Statistical Learning*. Springer, 2009.

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *CVPR*, 2016.

Kaiming He, Haoqi Fan, Yuxin Wu, Saining Xie, and Ross Girshick. Momentum contrast for unsupervised visual representation learning. In *CVPR*, 2020.

Matthias Hein and Jean-Yves Audibert. Intrinsic dimensionality estimation of submanifolds in Rd. In *ICML*, 2005.

Dan Hendrycks and Thomas Dietterich. Benchmarking neural network robustness to common corruptions and perturbations. In *ICLR*, 2018.

Dan Hendrycks and Kevin Gimpel. Gaussian error linear units (GELUs). *arXiv:1606.08415*, 2016.

Geoffrey E. Hinton and Ruslan R. Salakhutdinov. Reducing the dimensionality of data with neural networks. *Science*, 2006.

R Devon Hjelm, Alex Fedorov, Samuel Lavoie-Marchildon, Karan Grewal, Phil Bachman, Adam Trischler, and Yoshua Bengio. Learning deep representations by mutual information estimation and maximization. In *ICLR*, 2019.

Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural computation*, 1997.

David Hong, Yue Sheng, and Edgar Dobriban. Selecting the number of components in PCA via random signflips. *arXiv:2012.02985*, 2020.

Weihua Hu, Takeru Miyato, Seiya Tokui, Eiichi Matsumoto, and Masashi Sugiyama. Learning discrete representations via information maximizing self-augmented training. In *ICML*, 2017.

Gao Huang, Zhuang Liu, Laurens Van Der Maaten, and Kilian Q Weinberger. Densely connected convolutional networks. In *CVPR*, 2017.

Lawrence Hubert and Phipps Arabie. Comparing partitions. *Journal of Classification*, 1985.

Like Hui, Mikhail Belkin, and Preetum Nakkiran. Limitations of neural collapse for understanding generalization in deep learning. *arXiv:2202.08384*, 2022.

Frank Hutter, Lars Kotthoff, and Joaquin Vanschoren. *Automatic Machine Learning: Methods, Systems, Challenges*. Springer, 2019.

Aapo Hyvärinen and Erkki Oja. Independent component analysis: algorithms and applications. *Neural Networks*, 2000.

Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *ICML*, 2015.

Arthur Jacot, Franck Gabriel, and Clément Hongler. Neural tangent kernel: Convergence and generalization in neural networks. In *NeurIPS*, 2018.

Pan Ji, Tong Zhang, Hongdong Li, Mathieu Salzmann, and Ian Reid. Deep subspace clustering networks. In *NeurIPS*, 2017.

Xu Ji, João F Henriques, and Andrea Vedaldi. Invariant information clustering for unsupervised image classification and segmentation. In *ICCV*, 2019.

Chi Jin, Rong Ge, Praneeth Netrapalli, Sham M. Kakade, and Michael I. Jordan. How to escape saddle points efficiently. In *ICML*, 2017.

Chi Jin, Praneeth Netrapalli, and Michael I Jordan. Accelerated gradient descent escapes saddle points faster than gradient descent. In *COLT*, 2018.

Ian T. Jolliffe. *Principal Component Analysis*. Springer-Verlag, 2002.

Zhao Kang, Chong Peng, Jie Cheng, and Qiang Cheng. Logdet rank minimization with application to subspace clustering. *Computational intelligence and neuroscience*, 2015.

Koray Kavukcuoglu, Marc'Aurelio Ranzato, Rob Fergus, and Yann LeCun. Learning invariant features through topographic filter maps. In *CVPR*, 2009.

Günter Klambauer, Thomas Unterthiner, Andreas Mayr, and Sepp Hochreiter. Self-normalizing neural networks. In *NeurIPS*, 2017.

Artemy Kolchinsky, Brendan D Tracey, and Steven Van Kuyk. Caveats for information bottleneck in deterministic scenarios. In *ICLR*, 2019.

Simon Kornblith, Ting Chen, Honglak Lee, and Mohammad Norouzi. Why do better loss functions lead to less transferable features? In *NeurIPS*, 2021.

Irwin Kra and Santiago R. Simanca. On circulant matrices. *Notices of the American Mathematical Society*, 2012.

Mark A. Kramer. Nonlinear principal component analysis using autoassociative neural networks. *AIChE Journal*, 1991.

Alex Krizhevsky. Learning multiple layers of features from tiny images, 2009. URL `https://www.cs.toronto.edu/~kriz/learning-features-2009-TR.pdf`.

Alex Krizhevsky, Ilya Sutskever, and Geoffrey E. Hinton. Imagenet classification with deep convolutional neural networks. In *NeurIPS*, 2012.

Andrew Lavin and Scott Gray. Fast algorithms for convolutional neural networks. In *CVPR*, 2016.

Yann LeCun. The MNIST database of handwritten digits, 1998. URL `http://yann.lecun.com/exdb/mnist/`.

Yann LeCun and Yoshua Bengio. Convolutional networks for images, speech, and time series. *The handbook of brain theory and neural networks*, 1995.

Yann LeCun, Lawrence D. Jackel, Léon Bottou, Corinna Cortes, John S. Denker, Harris Drucker, Isabelle Guyon, Urs A. Muller, Eduard Sackinger, Patrice Simard, and Vladimir Vapnik. Learning algorithms for classification: A comparison on handwritten digit recognition. *Neural networks: the statistical mechanics perspective*, 1995.

Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 1998.

Yann LeCun, Fu Jie Huang, and Léon Bottou. Learning methods for generic object recognition with invariance to pose and lighting. In *CVPR*, 2004.

Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. Deep learning. *Nature*, 2015.

José Lezama, Qiang Qiu, Pablo Musé, and Guillermo Sapiro. OLE: Orthogonal low-rank embedding-a plug and play geometric loss for deep learning. In *CVPR*, 2018.

Ke Li, Shichong Peng, Tianhao Zhang, and Jitendra Malik. Multimodal image synthesis with conditional implicit maximum likelihood estimation. *IJCV*, 2020.

Yanjun Li and Yoram Bresler. Multichannel sparse blind deconvolution on the sphere. *IEEE Transactions on Information Theory*, 2019.

Zengyi Li, Yubei Chen, Yann LeCun, and Friedrich T. Sommer. Neural manifold clustering and embedding. *arXiv:2201.10000*, 2022.

Sheng Liu, Xiao Li, Yuexiang Zhai, Chong You, Zhihui Zhu, Carlos Fernandez-Granda, and Qing Qu. Convolutional normalization: Improving deep convolutional network robustness and training. In *NeurIPS*, 2021.

Bethany Lusch, J. Nathan Kutz, and Steven Brunton. Deep learning for universal linear embeddings of nonlinear dynamics. *Nature Communications*, 2018.

Yi Ma, Harm Derksen, Wei Hong, and John Wright. Segmentation of multivariate mixed data via lossy data coding and compression. *PAMI*, 2007.

Andrew L Maas, Awni Y Hannun, and Andrew Y Ng. Rectifier nonlinearities improve neural network acoustic models. In *ICML*, 2013.

Jan MacDonald, Stephan Wäldchen, Sascha Hauch, and Gitta Kutyniok. A rate-distortion framework for explaining neural network decisions. *arXiv:1905.11092*, 2019.

Najib J. Majaj, Ha Hong, Ethan A. Solomon, and James J. DiCarlo. Simple learned weighted sums of inferior temporal neuronal firing rates accurately predict human core object recognition performance. *Journal of Neuroscience*, 2015.

James Martens, Andy Ballard, Guillaume Desjardins, Grzegorz Swirszcz, Valentin Dalibard, Jascha Sohl-Dickstein, and Samuel S. Schoenholz. Rapid training of deep neural networks without skip connections or normalization layers using deep kernel shaping. *arXiv:2110.01765*, 2021.

Michael Mathieu, Mikael Henaff, and Yann LeCun. Fast training of convolutional networks through FFTs. *arXiv:1312.5851*, 2013.

Michael McCloskey and Neal J Cohen. Catastrophic interference in connectionist networks: The sequential learning problem. *Psychology of learning and motivation*, 1989.

Warren McCulloch and Walter Pitts. A logical calculus of the ideas immanent in nervous activity. *The Bulletin of Mathematical Biology*, 1943.

Song Mei and Andrea Montanari. The generalization error of random features regression: Precise asymptotics and double descent curve. *Communications on Pure and Applied Mathematics*, 2021.

Poorya Mianjy, Raman Arora, and Rene Vidal. On the implicit bias of dropout. In *ICML*, 2018.

Dustin G. Mixon, Hans Parshall, and Jianzong Pi. Neural collapse with unconstrained features. *arXiv:2011.11619*, 2020.

Takeru Miyato, Toshiki Kataoka, Masanori Koyama, and Yuichi Yoshida. Spectral normalization for generative adversarial networks. In *ICLR*, 2018.

Vishal Monga, Yuelong Li, and Yonina C Eldar. Algorithm unrolling: Interpretable, efficient deep learning for signal and image processing. *arXiv:1912.10557*, 2019.

Sangnam Nam, Mike E. Davies, Michael Elad, and Réni. Gribonval. The cosparse analysis model and algorithms. *Applied and Computational Harmonic Analysis*, 2013.

Yurii Nesterov. A method for unconstrained convex minimization problem with the rate of convergence o (1/kˆ2). In *Doklady AN USSR*, 1983.

Oliver Nina, Jamison Moody, and Clarissa Milligan. A decoder-free approach for unsupervised clustering and manifold learning with random triplet mining. In *ICCV Workshops*, 2019.

Jorge Nocedal and Stephen J. Wright. *Numerical Optimization*. Springer, 2006.

Chigozie Nwankpa, Winifred Ijomah, Anthony Gachagan, and Stephen Marshall. Activation functions: Comparison of trends in practice and research for deep learning. *arXiv:1811.03378*, 2018.

Bruno A. Olshausen and David J. Field. Emergence of simple-cell receptive field properties by learning a sparse code for natural images. *Nature*, 1996.

Aaron van den Oord, Yazhe Li, and Oriol Vinyals. Representation learning with contrastive predictive coding. *arXiv:1807.03748*, 2018.

Vardan Papyan, Yaniv Romano, and Michael Elad. Convolutional neural networks analyzed via convolutional sparse coding. *JMLR*, 2017.

Vardan Papyan, X.Y. Han, and David L. Donoho. Prevalence of neural collapse during the terminal phase of deep learning training. *PNAS*, 2020.

Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Köpf, Edward Yang, Zach DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. Pytorch: An imperative style, high-performance deep learning library. In *NeurIPS*, 2019.

Xi Peng, Jiashi Feng, Shijie Xiao, Jiwen Lu, Zhang Yi, and Shuicheng Yan. Deep sparse subspace clustering. *arXiv:1709.08374*, 2017.

Phil Pope, Chen Zhu, Ahmed Abdelkader, Micah Goldblum, and Tom Goldstein. The intrinsic dimension of images and its impact on learning. In *ICLR*, 2021.

Haozhi Qi, Chong You, Xiaolong Wang, Yi Ma, and Jitendra Malik. Deep isometric learning for visual recognition. In *ICML*, 2020.

Qing Qu, Xiao Li, and Zhihui Zhu. A nonconvex approach for exact and efficient multichannel sparse blind deconvolution. In *NeurIPS*, 2019.

Qing Qu, Xiao Li, and Zhihui Zhu. Exact recovery of multichannel sparse blind deconvolution via gradient descent. *SIAM Journal on Imaging Sciences*, 2020a.

Qing Qu, Yuexiang Zhai, Xiao Li, Yuqian Zhang, and Zhihui Zhu. Geometric analysis of nonconvex optimization landscapes for overcomplete learning. In *ICLR*, 2020b.

J. R. Quinlan. Induction of decision trees. *Machine Learning*, 1986.

Salah Rifai, Pascal Vincent, Xavier Muller, Xavier Glorot, and Yoshua Bengio. Contractive auto-encoders: Explicit invariance during feature extraction. In *ICML*, 2011.

David Rolnick and Max Tegmark. The power of deeper networks for expressing natural functions. In *ICLR*, 2018.

Olaf Ronneberger, Philipp Fischer, and Thomas Brox. U-net: Convolutional networks for biomedical image segmentation. In *MICCAI*, 2015.

Frank Rosenblatt. The perceptron: a probabilistic model for information storage and organization in the brain. *Psychological Review*, 1958.

Ron Rubinstein and Michael Elad. Dictionary learning for analysis-synthesis thresholding. *IEEE Transactions on Signal Processing*, 2014.

David E. Rumelhart, Geoffrey E. Hinton, and Ronald J. Williams. Learning representations by back-propagating errors. *Nature*, 1986.

Sara Sabour, Nicholas Frosst, and Geoffrey E. Hinton. Dynamic routing between capsules. In *NeurIPS*, 2017.

Dominik Scherer, Andreas Müller, and Sven Behnke. Evaluation of pooling operations in convolutional architectures for object recognition. In *International conference on artificial neural networks*, 2010.

Hanie Sedghi, Vineet Gupta, and Philip M. Long. The singular values of convolutional layers. In *ICLR*, 2019.

Noam Shazeer, Azalia Mirhoseini, Krzysztof Maziarz, Andy Davis, Quoc Le, Geoffrey Hinton, and Jeff Dean. Outrageously large neural networks: The sparsely-gated mixture-of-experts layer. In *ICLR*, 2017.

Yanyao Shen and Sujay Sanghavi. Learning with bad training data via iterative trimmed loss minimization. In *ICML*, 2019.

Jonathan R Shewchuk. An introduction to the conjugate gradient method without the agonizing pain. Technical report, Carnegie Mellon University, 1994.

Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. In *ICLR*, 2015.

William R. Softky and Christof Koch. The highly irregular firing of cortical cells is inconsistent with temporal integration of random EPSPs. *Journal of Neuroscience*, 1993.

Mahdi Soltanolkotabi, Adel Javanmard, and Jason D. Lee. Theoretical insights into the optimization landscape of over-parameterized shallow neural networks. *IEEE Transactions on Information Theory*, 2018.

Stefano Spigler, Mario Geiger, and Matthieu Wyart. Asymptotic learning curves of kernel methods: empirical data vs teacher-student paradigm. *arXiv:1905.10843*, 2019.

Rupesh Kumar Srivastava, Klaus Greff, and Jürgen Schmidhuber. Highway networks. *arXiv:1505.00387*, 2015.

Alexander Strehl and Joydeep Ghosh. Cluster ensembles—a knowledge reuse framework for combining multiple partitions. *JMLR*, 2002.

Jeremias Sulam, Vardan Papyan, Yaniv Romano, and Michael Elad. Multilayer convolutional sparse modeling: Pursuit and dictionary learning. *IEEE Transactions on Signal Processing*, 2018.

Xiaoxia Sun, Nasser M Nasrabadi, and Trac D Tran. Supervised deep sparse coding networks for image classification. *TIP*, 2020.

Christian Szegedy, Wojciech Zaremba, Ilya Sutskever, Joan Bruna, Dumitru Erhan, Ian Goodfellow, and Rob Fergus. Intriguing properties of neural networks. *arXiv:1312.6199*, 2013.

Yi Tay, Mostafa Dehghani, Jai Gupta, Dara Bahri, Vamsi Aribandi, Zhen Qin, and Donald Metzler. Are pre-trained convolutions better than pre-trained transformers? In *ACL*, 2021.

Tom Tirer and Joan Bruna. Extended unconstrained features model for exploring deep neural collapse. *arXiv:2202.08087*, 2022.

Naftali Tishby and Noga Zaslavsky. Deep learning and the information bottleneck principle. In *IEEE Information Theory Workshop*, 2015.

Shengbang Tong, Xili Dai, Ziyang Wu, Mingyang Li, Brent Yi, and Yi Ma. Incremental learning of structured memory via closed-loop transcription. *arXiv:2202.05411*, 2022.

Dmitry Ulyanov, Andrea Vedaldi, and Victor Lempitsky. Instance normalization: The missing ingredient for fast stylization. *arXiv:1607.08022*, 2016.

Nicolas Vasilache, Jeff Johnson, Michaël Mathieu, Soumith Chintala, Serkan Piantino, and Yann LeCun. Fast convolutional nets with fbfft: A GPU performance evaluation. In *ICLR*, 2015.

Rene Vidal, Yi Ma, and S. Shankar Sastry. *Generalized Principal Component Analysis*. Springer, 2016.

Oriol Vinyals, Yangqing Jia, Li Deng, and Trevor Darrell. Learning with recursive perceptual representations. In *NeurIPS*, 2012.

Andrew Wagner, John Wright, Arvind Ganesh, Zihan Zhou, Hossein Mobahi, and Yi Ma. Toward a practical face recognition system: Robust alignment and illumination by sparse representation. *PAMI*, 2012.

Michael B. Wakin, David L. Donoho, Hyeokho Choi, and Richard G. Baraniuk. The multiscale structure of non-differentiable image manifolds. In *Proceedings of SPIE*, 2005.

Colin Wei, Sham Kakade, and Tengyu Ma. The implicit and explicit regularization effects of dropout. In *ICML*, 2020.

Thomas Wiatowski and Helmut Bölcskei. A mathematical theory of deep convolutional neural networks for feature extraction. *IEEE Transactions on Information Theory*, 2018.

Scott Wisdom, Thomas Powers, James Pitton, and Les Atlas. Interpretable recurrent neural networks using sequential sparse recovery. *arXiv:1611.07252*, 2016.

John Wright and Yi Ma. *High-Dimensional Data Analysis with Low-Dimensional Models: Principles, Computation, and Applications*. Cambridge University Press, 2021.

John Wright, Yangyu Tao, Zhouchen Lin, Yi Ma, and Heung-Yeung Shum. Classification via minimum incremental coding length (MICL). In *NeurIPS*, 2008.

John Wright, Allen Y. Yang, Arvind Ganesh, S. Shankar Sastry, and Yi Ma. Robust face recognition via sparse representation. *PAMI*, 2009.

Denny Wu and Ji Xu. On the optimal weighted ℓ_2 regularization in overparameterized linear regression. *arXiv:2006.05800*, 2020.

Jianlong Wu, Keyu Long, Fei Wang, Chen Qian, Cheng Li, Zhouchen Lin, and Hongbin Zha. Deep comprehensive correlation mining for image clustering. In *ICCV*, 2019.

Yuxin Wu and Kaiming He. Group normalization. In *ECCV*, 2018.

Zhirong Wu, Yuanjun Xiong, Stella X. Yu, and Dahua Lin. Unsupervised feature learning via non-parametric instance discrimination. In *CVPR*, 2018.

Ziyang Wu, Christina Baek, Chong You, and Yi Ma. Incremental learning via rate reduction. In *CVPR*, 2021.

Junyuan Xie, Ross Girshick, and Ali Farhadi. Unsupervised deep embedding for clustering analysis. In *ICML*, 2016.

Saining Xie, Ross Girshick, Piotr Dollár, Zhuowen Tu, and Kaiming He. Aggregated residual transformations for deep neural networks. In *CVPR*, 2017.

Bing Xu, Naiyan Wang, Tianqi Chen, and Mu Li. Empirical evaluation of rectified activations in convolutional network. *arXiv:1505.00853*, 2015.

Jianwei Yang, Devi Parikh, and Dhruv Batra. Joint unsupervised learning of deep representations and image clusters. In *CVPR*, 2016.

Zitong Yang, Yaodong Yu, Chong You, Jacob Steinhardt, and Yi Ma. Rethinking bias-variance trade-off for generalization of neural networks. In *ICML*, 2020.

Chong You, Chun-Guang Li, Daniel P Robinson, and René Vidal. Oracle based active set algorithm for scalable elastic net subspace clustering. In *CVPR*, 2016.

Yaodong Yu, Kwan Ho Ryan Chan, Chong You, Chaobing Song, and Yi Ma. Learning diverse and discriminative representations via the principle of maximal coding rate reduction. *NeurIPS*, 2020.

Manzil Zaheer, Satwik Kottur, Siamak Ravanbakhsh, Barnabas Poczos, Russ R. Salakhutdinov, and Alexander J. Smola. Deep sets. In *NeurIPS*, 2017.

John Zarka, Louis Thiry, Tomás Angles, and Stéphane Mallat. Deep network classification by scattering and homotopy dictionary learning. In *ICLR*, 2020.

John Zarka, Florentin Guth, and Stéphane Mallat. Separation and concentration in deep networks. In *ICLR*, 2021.

Matthew D. Zeiler and Rob Fergus. Visualizing and understanding convolutional networks. In *ECCV*, 2014.

Chiyuan Zhang, Samy Bengio, Moritz Hardt, Benjamin Recht, and Oriol Vinyals. Understanding deep learning requires rethinking generalization. In *ICLR*, 2017.

Hongyi Zhang, Moustapha Cisse, Yann N Dauphin, and David Lopez-Paz. Mixup: Beyond empirical risk minimization. In *ICLR*, 2018a.

Junjian Zhang, Chun-Guang Li, Chong You, Xianbiao Qi, Honggang Zhang, Jun Guo, and Zhouchen Lin. Self-supervised convolutional subspace clustering network. In *CVPR*, 2019a.

Tong Zhang, Pan Ji, Mehrtash Harandi, Richard Hartley, and Ian Reid. Scalable deep k-subspace clustering. In *ACCV*, 2018b.

Tong Zhang, Pan Ji, Mehrtash Harandi, Wenbing Huang, and Hongdong Li. Neural collaborative subspace clustering. *arXiv:1904.10596*, 2019b.

Xiao Zhang, Yaodong Yu, Lingxiao Wang, and Quanquan Gu. Learning one-hidden-layer relu networks via gradient descent. In *AISTATS*, 2019c.

Kai Zhong, Zhao Song, Prateek Jain, Peter L Bartlett, and Inderjit S Dhillon. Recovery guarantees for one-hidden-layer neural networks. In *ICML*, 2017.

Jinxing Zhou, Xiao Li, Tian Ding, Chong You, Qing Qu, and Zhihui Zhu. On the optimization landscape of neural collapse under mse loss: Global optimality with unconstrained features. *arXiv:2203.01238*, 2022.

Pan Zhou, Yunqing Hou, and Jiashi Feng. Deep adversarial subspace clustering. In *CVPR*, 2018.

Zhihui Zhu, Tianyu Ding, Jinxin Zhou, Xiao Li, Chong You, Jeremias Sulam, and Qing Qu. A geometric analysis of neural collapse with unconstrained features. *arXiv:2105.02375*, 2021.

Barret Zoph and Quoc V Le. Neural architecture search with reinforcement learning. *arXiv:1611.01578*, 2016.