DEEP LEARNING ON MANIFOLD-VALUED DATA WITH APPLICATIONS TO NEUROIMAGING AND VISION

by

Xingjian Zhen

A dissertation submitted in partial fulfillment of
the requirements for the degree of

Doctor of Philosophy

(Computer Sciences)

at the

UNIVERSITY OF WISCONSIN–MADISON

2023

Date of final oral examination: 03/27/23

The dissertation is approved by the following members of the Final Oral Committee:
Frederic Sala, Assistant Professor, Computer Sciences
Yingyu Liang, Assistant Professor, Computer Sciences
Barbara B. Bendlin, Professor, Department of Medicine
Vikas Singh (Adviser), Professor, Biostatistics and Medical Informatics, Computer Sciences

To all my family.

ACKNOWLEDGMENTS

I would like to express my gratitude to my advisor, Vikas, for his continuous support, invaluable guidance, and endless patience during my PhD journey. His insightful feedback encouraged me to conduct meaningful and impactful research in this rapidly growing field. I learned so much from Vikas, and under his supervision, we were fortunate enough to win the Best Paper Award at ECCV 2022. His enthusiasm for research will continue to guide me throughout my career journey. I also want to thank the rest of my thesis committee: Barb, Fred, and Yingyu, for their valuable feedback and comments. I also want to thank all of my collaborators, particularly Rudrasis. It's my pleasure to collaborate with all of them.

I am also grateful to my family for their constant encouragement and support. Without their help, I would not have had the opportunity to study at UW-Madison from China. Their unwavering faith in me has been a source of inspiration and motivation throughout this journey.

Finally, I want to thank my loving boyfriend, Xiaoxiang, for his unconditional love and understanding. During my PhD, whenever I felt down, he was always there to support me. Although we are in a long-distance relationship, we believe that our future can be better with our mutual efforts. I also want to thank my two cats, Puppy and Bunny, for providing me with mental support during the most difficult period, the pandemic.

Without the help and support of these wonderful people, completing my PhD would not have been possible.

CONTENTS

LIST OF TABLES

LIST OF FIGURES

ABSTRACT

Deep learning has achieved significant success in various applications, including image classification, image segmentation, natural language processing, and more. However, traditional deep learning models assume that the input data can be represented as Euclidean vectors. This assumption limits their applicability to structured data that do not conform to Euclidean space, such as symmetric positive definite matrices and distribution functions. When dealing with such structured data, omitting the geometric information and applying traditional deep learning models often leads to suboptimal performance. In this thesis, our focus is to bridge the gap between available deep learning models and structured data of the form described above by incorporating the inherent geometric structures of the data. First, we study how to model fiber bundles in brain images when each voxel along the trajectory is manifold-valued. We show that doing so allows statistical analysis with improved power. We then describe a method to transform one manifold to another, allowing the generation of Orientation Distribution Functions (ODF) images with higher angular information based on a given Diffusion Tensor Imaging (DTI) while preserving meaningful group-wise differences. We also study a problem setting which allows effectively tracking the covariance matrix along a neural network, which is useful when training a certified robust network. Finally, we discuss the use of distance correlation to evaluate the correlation between two different random vectors. This approach offers multiple benefits such as robustness against transferred attack, disentanglement of a generative model, and evaluation of similarity between two neural networks. In summary, our modifications to traditional deep learning models allow for effective utilization of manifold information, resulting in improved performance in terms of speed, efficiency, and robustness across various applications.

1 INTRODUCTION

Machine learning is a field of study that focuses on developing algorithms that can learn patterns and relationships in data. The goal of these algorithms is to make classifications or predictions on unseen data. To achieve this goal, one often simplifies the setting by first deciding on a feasible set of models (also known as the hypothesis class), and then estimating the "best" model, which is specified by the estimated parameters, for the given data. Thus, the design and implementation of machine learning algorithms are heavily influenced by the selection of a feasible model class (or type) $\mathbf{\Theta}$, denoted by $\mathbf{\Theta}$, and the data types, denoted by $\mathbf{X}$. These two fundamental components play a crucial role in determining the type of machine learning model that is best suited for a particular application.

The model types $\mathbf{\Theta}$ can consist of a variety of different models, including non-parametric models such as kernel density estimation Terrell and Scott (1992), kernel SVM Amari and Wu (1999), and KNN Peterson (2009), as well as parametric models, such as distribution families like Poisson and normal distributions DeGroot and Schervish (2002), and the increasingly popular deep neural networks (DNN). Similarly, the data $\mathbf{X}$ can also consist of different types of data, including documents Devlin et al. (2019), images Deng et al. (2009), videos Mao et al. (2018), and more. Based on the specific data type $\mathcal{X} \in \mathbf{X}$ and the assumption of specific feasible model class $\Theta_j \in \mathbf{\Theta}$, when given the data samples $\{x_i\}_{i=1}^n \subset \mathcal{X}$, one normally tries to find the optimal model $\theta \in \Theta_j$. This process is referred to as the learning problem Bottou and Bousquet (2007); Kingma and Ba (2015). For instance, in a regression problem, the goal is to minimize the mean square error between the model's prediction and the actual labels. Given the data samples $(x_i, y_i)_{i=1}^n \subset \mathcal{X}$, where y_i represents the labels, the learning algorithm attempts to find the solution to the optimization problem $\arg\min \theta \in \Theta_j \sum_{i=1}^n (f_\theta(x_i) - y_i)^2$. In other words, it tries to find the optimal model $\theta \in \Theta_j$ that minimizes the difference between the prediction and the actual labels.

In order to make the design and analysis of machine learning algorithms more manageable, it is common to represent the data as high-dimensional feature vectors

and to collect all parameters of the model as a vector. By doing so, an assumption is made that both the model and data exist in high-dimensional Euclidean spaces. This simplifies the mathematical formulation of the learning problem, making it easier to analyze various models.

If the model spaces are known to be Euclidean (e.g., weights and biases in Alexnet Krizhevsky et al. (2012), Resnet He et al. (2016a), etc.), we can use standard learning algorithms (rather optimization solvers), such as gradient descent or stochastic gradient descent (SGD), to learn the model. However, in many cases, the model spaces are not Euclidean, which means that the standard schemes are not directly applicable. For instance, the exponential family and other parametric families do not correspond to Euclidean spaces, so a different approach must be taken in these spaces.

Example 1.1. *The exponential family distribution does not correspond to an Euclidean space, for example, a standard normal distribution for high dimensional data,* $f(x) = \frac{1}{\sqrt{(2\pi)^k|\Sigma|}} e^{-\frac{1}{2}(x-\mu)^T\Sigma^{-1}(x-\mu)}$*, where* μ *is the mean of the distribution and* Σ *is the covariance matrix. One can easily notice that the parameters of the model,* μ, Σ*, do not lie in the Euclidean space, due to the fact that* Σ *should be a symmetric positive definite matrix.*

In neural networks, we can assume that the weight w (of one specific layer) is in $\mathbb{R}^p$. Based on this assumption, we can apply SGD to update the weight w. However, if there is a constraint on the weight, such as $\|w\| = 1$, the space is no longer Euclidean, and other learning algorithms are required to update the weight.

Example 1.2. *For a simple one-layer neural network without the bias term, we have* $f(x) = w^Tx$*. But there is another constraint on* w *such that* $\|w\| = 1$*. After one iteration of gradient decent, we have* $w' = w - \eta\nabla_w f(x)$*, where* η *is the predefined step size. We have no information on* w'*, that is, whether* $\|w'\|$ *is still* 1*.*

In such cases, it is necessary to introduce the geometric information of the models (regarding what is permissible versus not) during the training process. This can be achieved through the use of Riemannian geometry and other mathematical tools that capture the geometry of the model spaces. This is not a new

idea. Lebanon (2005) introduced the idea of geodesic distances between different models in spaces of probability models, while Lebanon and Lafferty (2001) showed how the AdaBoost optimizer can be used in the dual space of exponential models. Other works, such as Lafferty (1988); Lee (2003), have discussed incorporating Riemannian geometry and other tools to ensure that the geometry of the models is properly accounted for during the training process, which can lead to improved model learning.

On the other hand, for data spaces Θ, there are many different data types like images, videos, documents and so on. Although we can usually pretend that the data lies in the Euclidean space, for example, we can treat different channels of color images to be independent so that we can assume that one pixel of a given image lies in $\mathbb{R}^3$, it is obvious that there are many other types of data objects which cannot be expressed as Euclidean vectors easily. Let us take the same color image. If we are not looking at the RGB channels, but instead we use the HSV representation, we can immediately notice that $H \in [0, 2\pi], S \in [0, 1], V \in [0, 1]$ is not a Euclidean space. There are some other examples of data types that are common in machine learning such as graphs David et al. (2020), trees Ahmed et al. (2019), covariance matrices Louizos and Welling (2016), spheres Cohen et al. (2018), and distribution functions Carrazza and Cruz-Martinez (2019), etc. Those data naturally come with constraints or structures. For example, all the covariance matrices must satisfy the constraint that they should be symmetric positive semidefinite matrices. This means that the eigenvalues must be positive or equal to zero and that the matrix should be symmetrical. Moreover, for all the distribution functions, every component in the distribution function should be $\geqslant 0$, and the summation of all the components should be 1. Such structured data are becoming more and more common in several problem settings, including medical imaging Yang et al. (2020), social sciences Lazer et al. (2009), and physics Toth et al. (2020). For example, graphs and trees are commonly used in social science Fan et al. (2019); Guo and Wang (2021), particle physics Shlomi et al. (2021), molecular fingerprints Duvenaud et al. (2015), and medical imaging Wu et al. (2019); Yang et al. (2020). Covariance matrices are used in state estimation Liu et al. (2018), stock market analysis Lin et al. (2021),

molecular physics Anderson et al. (2019), and independence testing Terranova et al. (2015). In neuroimaging, diffusion-weighted imaging (DWI) measures the diffusion ability of the water molecules within tissues Basser et al. (1994); Leow et al. (2008); Hua et al. (2008), while the most widely used technique to quantify the diffusion signal, called diffusion tensor imaging (DTI), models the direction and intensity with an $\mathrm{SPD}(3)$ matrix at each voxel Le Bihan et al. (2001); Alger (2012). Some more examples are described in Section 1.1.

We can notice that the data in the examples above are not in the Euclidean space (or cannot be trivially expressed by Euclidean vectors). Therefore, the core mathematical operations used by the traditional machine learning algorithms such as addition, subtraction, and multiplication cannot be easily carried out for structured data. For example, adding two points in the unit disk might cause the results to go outside of the unit disk (discussion in the Example 1.3), and multiplying a given unit vector by any scalar (except the trivial case "1") might cause it to leave the unit sphere. There are other cases where defining addition and multiplication are non-trivial, such as addition of two graphs with different number of nodes and edges (as shown in the Figure 1.2).

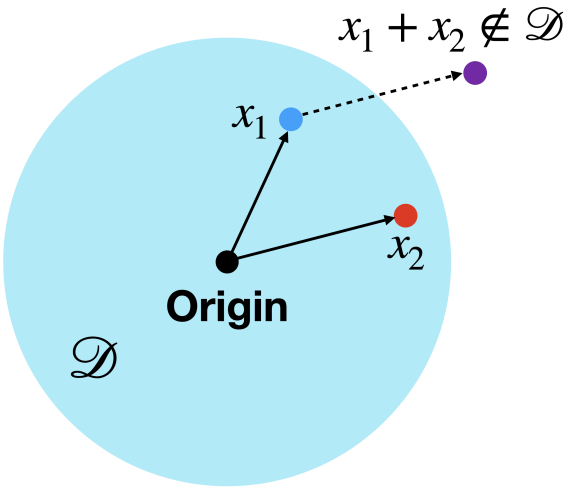

Figure 1.1: Addition in the unit disk space might end up outside of the space.

Example 1.3. *As shown in the Figure 1.1, we have a unit disk space defined as* $\mathcal{D} = \{Q : \|Q - O\| \leqslant 1\}$ *where* O *is the origin of the unit disk. For a given two points* x_1, x_2*, the addition* $x_1 + x_2$ *might go outside of the unit disk* (*in some cases,* $\|x_1 + x_2 - O\| > 1$).

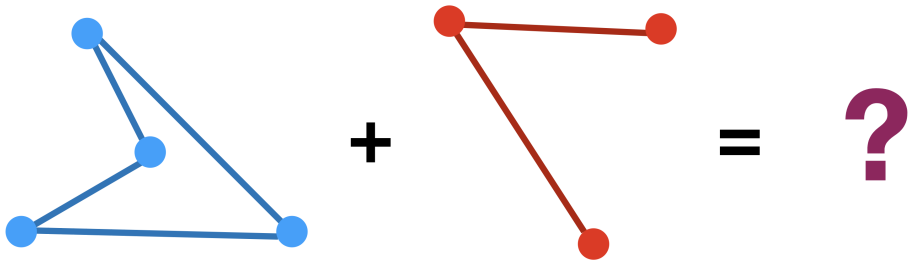

Figure 1.2: For two graphs, the definition of addition is non-trival.

Figure 1.2 illustrates two graphs, $G_1 = (V_1, E_1)$ and $G_2 = (V_2, E_2)$. Depending on the application, different options may exist for defining the addition of $G_1 + G_2$. However, this definition can be difficult to express using Euclidean operators.

Since many traditional machine learning methods heavily rely on these operations (fully connected layers are defined with multiplication and addition $y = Wx + b$), it can be problematic to apply them to such structured data directly. One possible solution is to project the structured data from the original non-Euclidean space into the Euclidean space and perform the operations in the new Euclidean space Huang et al. (2019); Brehmer and Cranmer (2020). However, this approach is not a universal solution. The computational burden of mapping the structured data into a new space can be significant, and it can also lead to decreased accuracy due to the loss of information during the mapping process. Additionally, some projection methods may simply not work, as they may fail to capture the underlying structure of the data. For example, if we are dealing with spaces with non-zero curvature like spheres, there is no simple one-to-one map from a such space to the Euclidean space that can preserve the distances between all pairs of points.

Inspired by the need to explore the inherent structure of data spaces, there is a growing body of research that broadens conventional techniques from Euclidean

spaces to manifolds. This thesis provides several approaches to tackle the inherent structure by utilizing chart maps, redefining required operations such as group operators, etc. We shortly highlight why the geometry of the data space plays a crucial role in constructing more effective and accurate neural networks.

1.1 Some Examples of Structured Data

As the discussion above suggests, we have introduced the intuition that simply using Euclidean structure to represent the data space might lead to some inaccuracies. Here, we would like to give some examples of real-world use cases where the types of structures described above play an important role in the data space.

Graphs

One of the most commonly used data structure is a graph. A graph can be defined as a collection of vertices and edges where vertices represent objects and edges represent relationships between those objects. Graphs are good representations to describe the interconnected structures within the data space.

Example 1.4. *Molecules. Reiser et al.* (2022); *Jørgensen and Bhowmik* (2022) *In chemistry, a molecule is made up of atoms that are connected by chemical bonds, such as ionic bonds and covalent bonds etc. These bonds can be represented as edges in a graph, where the atoms are vertices, as shown in Figure 1.3* (*a*).

Example 1.5. *Social networks. Fan* (2012); *Akhtar and Ahamad* (2021) *Similarly, social networks can also be represented as graphs, where the individuals in the network are the vertices and the relationship between individuals are the edges. The graph representation of social networks can be used to analyze the dynamics of a network and provide a straight forward way to cluster individuals inside.*

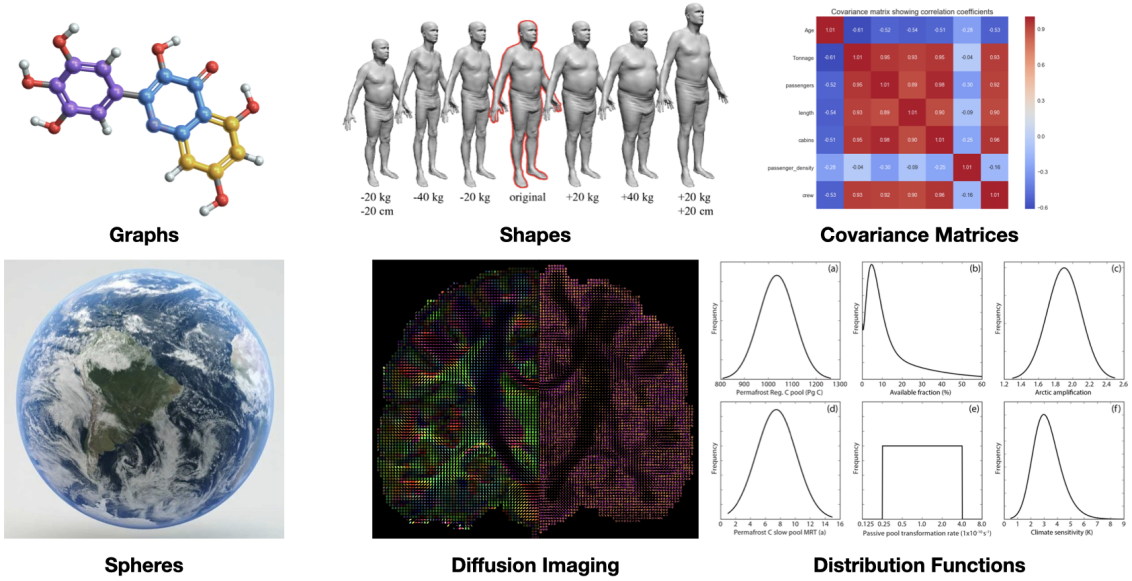

Figure 1.3: Some examples of the structured data.

Symmetric Positive Definite (SPD) Matrices

Matrices (or tensors in high-dimensional space) are another common data type. We can always convert a matrix into a vector and apply operators within this vector space if there are no inherent constraints in the matrix. However, most matrices do have inherent constraints. The most prevalent constraint is that the matrix A should be a symmetric positive definite matrix, as defined mathematically by $A = A^T; \forall x \neq 0, x^T Ax > 0$. SPD matrices appear as covariance matrices Ma et al. (2010); Xu et al. (2021), in diffusion tensor imaging, and so on.

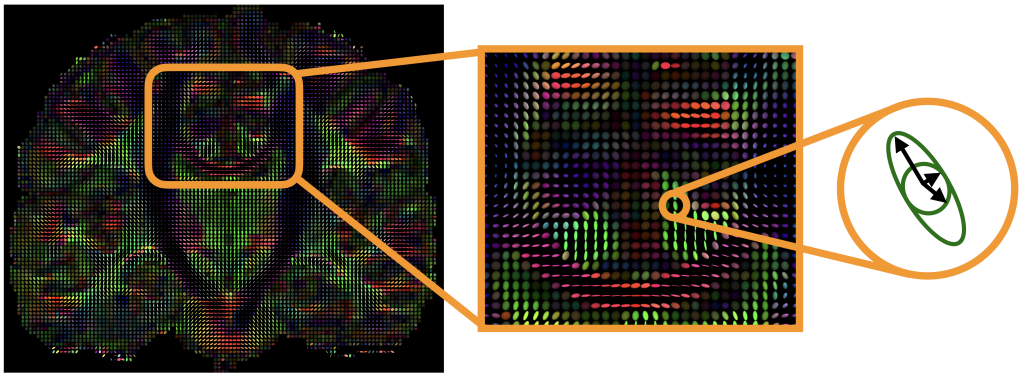

Figure 1.4: A visualization of the diffusion tensor imaging (DTI), where each voxel is a 3×3 SPD matrix.

Example 1.6. *Diffusion tensor imaging (DTI). Alexander et al. (2007) With diffusion MRI, one can measure the ability of the diffusion ability of water molecules in a specific location. If we use the Gaussian model to approximate the distribution, it will turn out to be a 3×3 SPD matrix at that location, as shown in the Figure 1.4. We can use these SPD matrices of the entire brain to analysis the connectivity of the brain or the structure changes of the white matters.*

Hyperbolic Spaces

The hyperbolic space, also known as negatively curved space, is a non-Euclidean geometry. There are several models that are used to describe a hyperbolic space, such as Poincaré half plane model $\{(x_1, ..., x_n) \in \mathbb{R}^n : x_n > 0\}$, Poincaré disc model $\{(x_1, ..., x_n) \in \mathbb{R}^n : \sum_{i=1}^n x_i^2 \leqslant 1\}$, and so on. The key difference between hyperbolic space and Euclidean space is the negative curvature, which can offer significant advantages in many use cases, such as efficient embedding of hierarchical structures Chami et al. (2020b), better clustering, and improved scalability in machine learning and data analysis Fan et al. (2022).

Example 1.7. *Embedding of trees. The embedding of trees can be done efficiently using hyperbolic space, as shown by Sonthalia and Gilbert (2020) and Chami et al. (2020a).*

These studies demonstrate that hyperbolic embeddings can better capture the hierarchical structure of trees, and that clustering in hyperbolic space is much simpler than in the original tree structure or in high-dimensional Euclidean space.

1.2 Why the Geometry of Data Space Is Important

As previously discussed, many types of data cannot be accurately represented as Euclidean vectors. While one can treat them as such, ignoring their inherent structure can lead to suboptimal results. On the other hand, if we take the structure of the data into consideration and adjust the model architecture based on the geometry and specifics of the data, improvements in the performance of the learning algorithms can be achieved. This has been demonstrated in studies such as Cohen et al. (2018); Kondor and Trivedi (2018); Huang et al. (2019); Chakraborty et al. (2022). An example of this can be seen in in computer vision where we try to detect the local patterns regardless of the actual position in the 2D image. Thus, due to the nature of the convolution, convolutional networks can offer equivariance to translation Kauderer-Abrams (2018). However, suppose we extend the 2D planar image to the sphere. In this case, the invariance to translation in the sphere should be updated to seek invariance to the 3D rotation. In Spherical CNNs by Cohen et al. (2018), the authors deploy $SO(3)$ group correlation of the object within the layers of Spherical CNN, with techniques from Chirikjian et al. (2001) and a generalization of the Fourier transform. Such a method considers the geometry of the sphere and thus shows improvement.

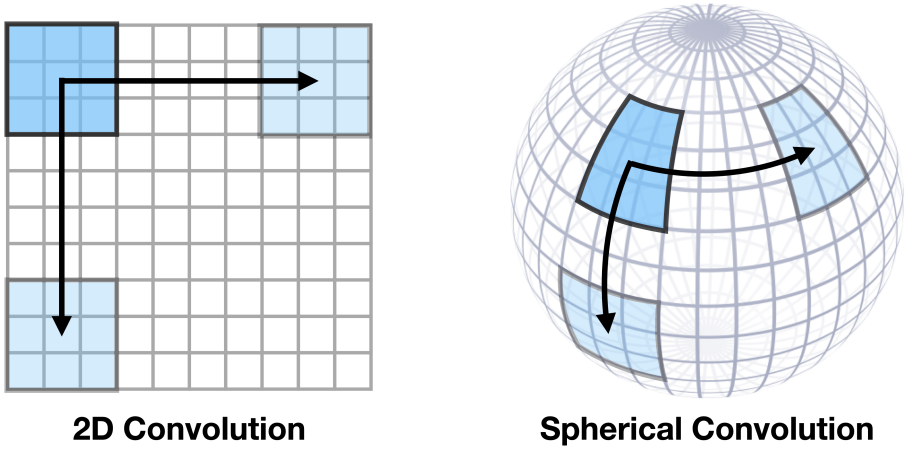

Figure 1.5: A demonstration of the difference between 2D CNN and Spherical CNN.

Example 1.8. *As shown in the Figure 1.5, kernels in 2D convolution are equivariant to translation. However, when dealing with spherical data, translation needs to be replaced with rotation. Therefore, Spherical CNNs are designed to achieve invariance to 3D rotation.*

Now if we consider scientific applications beyond the simple synthetic computer vision tasks mentioned above, in neuroimaging, we encounter manifold-valued data often. For example, an imaging procedure known as diffusion-weighted imaging (DWI), as discussed above, focuses on microstructure detection. The DWI acquisition attempts to measure the ability of the water molecules to diffuse at a specific location , e.g., 3D location/ voxel in the brain. While there are many varieties of diffusion weighted imaging methods, most try to measure the microstructural architecture of the brain and use the manifold-valued data to represent such information. In recent years, there has been a growing body of research exploring the use of deep learning in neuroimaging with explicit or implicit consideration of the structure of this manifold-valued data. Elsaid and Wu (2019); Son et al. (2019); Chakraborty et al. (2018c); Huang et al. (2019) are just a few examples of studies that have made important contributions in this field exploring the benefits and the limitations of manifold-valued data in deep learning algorithms. These studies show that by considering the structure of the data and developing algorithms that

are tailored to its geometry, we can significantly enhance the performance of deep learning models in terms of runtime, number of parameters, and even accuracy.

In this thesis, we aim to bridge the gap between the currently available deep learning models and the various types of structured data described above. We will focus on several special cases of structured data, and will modify the required operations accordingly. We will start by studying the geometric distance corresponding to the field of $\mathsf{SPD}(3)$ and S^{n-1}. Then, we will apply deep neural networks on the fiber bundles of the brain, which are sequences of $\mathsf{SPD}(3)$ or S^{n-1} manifold-valued voxels. Additionally, we will derive the tangent space and the chart maps to enable manifold-to-manifold modality transformation using invertible deep generative models. Later, we will explore covariance matrices and their advantages in training robust neural networks. We will then delve into a more general property shared between manifolds and Euclidean space, namely, the distance. We will demonstrate that the distance, whether defined on smooth manifolds with geometry or the traditional Euclidean distances, can enable evaluating statistical independence within deep learning models and offer various benefits. Finally, we will explore the hyperbolic space, where hierarchical structures of images can be well-embedded. We will show that hyperbolic space can provide benefits that Euclidean space is unable to provide.

1.3 Contribution and Scope of Thesis

This thesis explores the interface between differential geometry and deep learning for both manifold-valued data and Euclidean data and applies the ideas to neuroimaging and natural images commonly in vision. First, we show the overall scope of the thesis along two axes, as shown in the Figure 1.6.

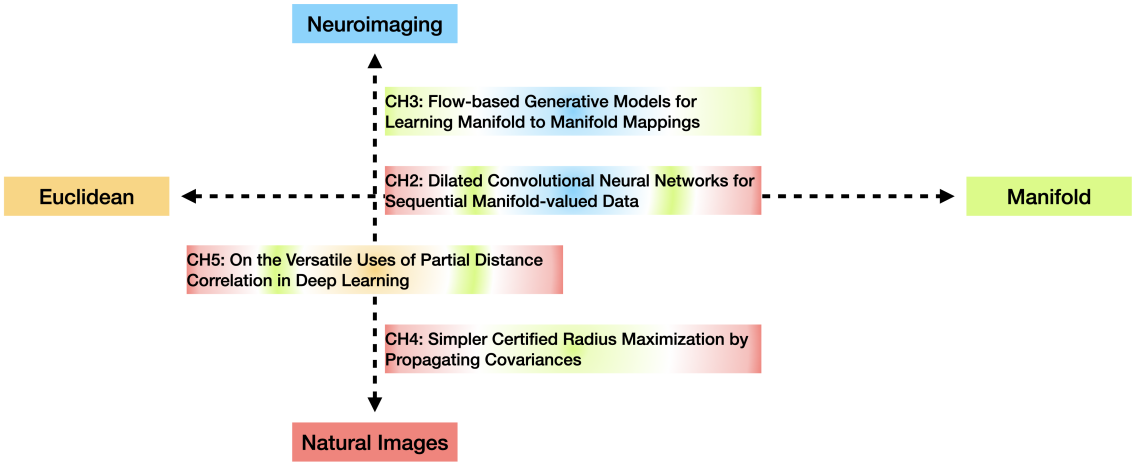

Figure 1.6: The overall scope of the thesis. The differential geometry in deep learning can be divided into four types according to the data structure and the application.

Dilated Convolutional Neural Networks for Sequential Manifold-valued Data

As discussed above, sizable empirical improvements are possible when the geometry of data spaces, such as structured data (e.g., graphs) or manifold-valued data (e.g., unit vectors or special matrices), are incorporated into the design of the model, architecture, and the algorithms. Motivated by neuroimaging applications, we propose to study formulations where the data are *sequential manifold-valued measurements*. This case is common in brain imaging, where the samples correspond to symmetric positive definite matrices or orientation distribution functions. Instead of a recurrent model which poses computational/technical issues, and inspired by results showing the viability of dilated convolutional models for sequence prediction, we describe a dilated convolutional neural network architecture for this task. On the technical side, our results show how the modules needed in our network can be derived while explicitly taking the Riemannian manifold structure into account. Our work also shows how the operations needed can leverage known results

for calculating the weighted Fréchet Mean (wFM). We present scientific results for group difference analysis in Alzheimer's disease (AD) where the groups are derived using AD pathology load: here the model finds several brain fiber bundles that are related to AD even when the subjects are all still cognitively healthy.

Flow-based Generative Models for Learning Manifold to Manifold Mappings

While recent proposals (like spherical CNN Cohen et al. (2018)) have extended a number of deep neural network architectures to manifold-valued data, and this has often provided strong improvements in performance, the literature on generative models for manifold data is quite sparse. Partly due to this gap, there are limited modality transfer/translation models for manifold-valued data whereas numerous such methods based on generative models are available for natural images. In this chapter, we will address this gap, motivated by a need in brain imaging – in doing so, we will expand the operating range of certain generative models (as well as generative models for modality transfer) from natural images to images with manifold-valued measurements. Our main result is a two-stream version of GLOW (flow-based invertible generative models) that can synthesize information of a field of one type of manifold-valued measurements given another. On the theoretical side, our formulation introduces three kinds of invertible layers for manifold-valued data, which are not only analogous to their functionality in flow-based generative models (e.g., GLOW) but also preserve the key benefits (determinants of the Jacobian are easy to calculate). For experiments, on a large dataset from the Human Connectome Project (HCP), we have obtained promising results where we can reliably and accurately reconstruct brain images of a field of orientation distribution functions (ODF) from diffusion tensor images (DTI), where the latter has a $5\times$ faster acquisition time but at the expense of worse angular resolution.

Simpler Certified Radius Maximization by Propagating Covariances

In both scientific and other missions with AI applications, robustness and calibration of models are important. One known strategy for adversarially training a robust model is to maximize its certified radius – the neighborhood around a given training sample for which the model's prediction remains unchanged. The scheme typically involves analyzing a "smoothed" classifier where one estimates the prediction corresponding to Gaussian samples in the neighborhood of each sample in the mini-batch, accomplished in practice by Monte Carlo sampling. In this chapter, we will investigate the hypothesis that this sampling bottleneck can potentially be mitigated by identifying ways to directly propagate the covariance matrix of the smoothed distribution through the network. To this end, our work finds that other than certain adjustments to the network, propagating the covariances must also be accompanied by additional accounting that keeps track of how the distributional moments transform and interact at each stage in the network. Our results show how satisfying these criteria yields an algorithm for maximizing the certified radius on datasets including Cifar-10, ImageNet, and Places365 while offering runtime savings on networks with moderate depth, with a small compromise in overall accuracy. In this thesis, we describe the details of the key modifications that enable practical use. Via various experiments, we have evaluated when our simplifications are sensible, and what the key benefits and limitations are.

On the Versatile Uses of Partial Distance Correlation in Deep Learning

Comparing the functional behavior of neural network models – whether it is a single network over time or two (or more networks) during or post-training – is an essential step in understanding what they are learning (and what they are not), and for identifying strategies for regularization or efficiency improvements. Despite recent progress, e.g., comparing vision transformers to CNNs, systematic compar-

ison of function, especially across different networks remains difficult. Classical statistical concepts such as canonical correlation analysis (CCA) are applicable in principle, but have been sparingly used due to efficiency issues. In this chapter, we propose to revisit a rigorous statistical concept, called distance correlation (and its partial variant), designed to evaluate correlation between feature spaces of different dimensions. Our work shows that its use opens the door to numerous immediate applications ranging from conditioning one model w.r.t. another, learning disentangled representations as well as learning diverse models that would immediately be more robust to adversarial attacks. Notice that, distance can be well defined in both an Euclidean space and a Riemannian manifold, we can easily extend the current formulation onto a Riemannian manifold if the geometry is known. Our experiments suggest a versatile regularizer (or constraint) with significant advantages, which avoids the drawbacks and limitations of CCA and divergence based measures respectively.

1.4 Outline

In Chapters 3 - 6, we describe our results on four problems and the proposed solutions in detail. We start with modifying the operation in the convolutional layer and the fully connected layer to work on $\mathsf{SPD}(3)$ and S^{n-1} manifold. And then, we describe results of applying the modified neural network to several well-known fiber bundles in Chapter 3 to perform statistical analysis of preclinical AD datasets. We show that the proposed method is beneficial when utilizing the covariance matrices ($\mathsf{SPD}(n)$) for video classification with better accuracy and faster speed. Motivated by the richer information availability (higher angular resolution) in ODF than DTI, knowing the limitation that ODF requires more scanning time than DTI, in Chapter 4, we discuss using an invertible generative model on manifold-valued data to generate ODF given DTI while preserving regions that are significantly different between groups. To further explore the benefit of covariance matrices ($\mathsf{SPD}(n)$), in Chapter 5, we explore whether with the use of covariance matrices and modifying the flow of the covariance matrices layer-by-layer with an upper bound

simplification, we can train a robust smoothed-network with some guarantees. All the above methods are based on the known structure of manifold spaces. Another important domain of manifold is the data-dependent manifold. Chapter 6 starts with one of the essential properties of differential geometry, the existence of distance. We introduce distance correlation into deep learning models to show benefits in three major applications: robustness, similarity and dis-similarity measurement between neural networks, and disentanglement in generative models.

2 PRELIMINARY

In this chapter, our aim is to provide a brief overview of key concepts from Riemannian geometry that are extensively utilized in most of the relevant applications in this thesis. We will also provide an overview of important background information, including statistical concepts and deep neural networks, for readers who are not familiar with these topics. The concepts and notations presented here will serve as a reference for subsequent chapters. However, we will introduce other concepts that are only relevant to specific chapters as needed. Due to the vastness of concepts within Riemannian geometry, we will focus on an abridged version to understand the steps in our algorithms. We suggest referring to textbooks such as Carmo (1992); Jost (2017) for a more detailed treatment. To complement our discussion, we will also introduce some statistical tools to clarify certain concepts that are useful in several chapters. Furthermore, we will discuss some related models in deep learning that demonstrate efficiency in various types of Euclidean data, including sequential data, image data, and more.

2.1 Riemannian Geometry

There are three types of structure in a Riemannian manifold: topological structure, differentiable structure, and Riemannian metric. The topological structure deals with topological concepts like continuity and convergence. The differentiable structure extends differentiability to the manifold and allows for the generalization of calculus by ensuring that the charts of the smooth manifolds are adequately compatible, with differentiability maintained during transitions between charts. Lastly, the Riemannian metric determines geometrical quantities such as curvature, distances, and angles on the manifold. This section will present these concepts in the appropriate order.

Topological Structure

As mentioned above, a Riemannian manifold is a topological manifold. In this section, we will delve deeper into the concept of topological structure. Topology can be defined as the branch of mathematics concerned with the properties of space that are preserved under continuous transformations, such as stretching or bending Munkres (1984). To start with, let us consider a set X.

Definition 2.1. *Let X be a set and $\mathcal{T}$ be a collection of subsets of X, called open sets. The topological space $(X, \mathcal{T})$ on the set X should satisfy the following axioms:*

1. *The empty set ϕ and X itself belong to $\mathcal{T}$*
2. *Any arbitrary (finite or infinite) union of members of $\mathcal{T}$ belongs to $\mathcal{T}$*
3. *The intersection of any finite number of members of $\mathcal{T}$ belongs to $\mathcal{T}$*

Definition 2.2. *A topological space X is said to be **Hausdorff** if for any $x, y \in X$ with $x \neq y$, there exist open set U, V such that $x \in U, y \in V$, and $U \cap V = \phi$*

Definition 2.3. *A base for a topological space $(X, \mathcal{T})$ is a collection of sets $\mathcal{B} \subseteq \mathcal{T}$, such that every open set in the space can be written as a union of sets in the base.*

Definition 2.4. *A topological space X is called second-countable if it has a countable base for its topology $\mathcal{T}$, which means it can be listed in a sequence.*

Definition 2.5. *A function $f : X \to Y$ between two topological space X, Y is a **homeomorphism** if the following properties are met:*

1. *f is bijection*
2. *f is continuous*
3. *the inverse function f^{-1} is continuous*

Using the definitions presented earlier, we can define an n-dimensional topological manifold. By employing the concept of homeomorphism, we can locally

map the topological space to Euclidean space and vice versa. As a result, we can transfer concepts from Euclidean space to the manifold space. We call a topological manifold $\mathcal{M}$ n-dimensional if it is locally equivalent to $\mathbb{R}^n$.

Definition 2.6. *An n-dimensional **topological manifold** $\mathcal{M}$ is a topological space that is Hausdorff, second-countable, and locally Euclidean of dimension n. For any point $x \in \mathcal{M}$, there is a neighborhood $U \subset \mathcal{M}$ containing x that is homeomorphic to $\mathbb{R}^n$. These homeomorphisms are typically called charts and denoted as $\varphi_U : U \subset \mathcal{M} \to \mathbb{R}^n$.*

There is also a special type of topological manifold $\mathcal{M}$, called a manifold with boundary, which contains a boundary $\partial\mathcal{M}$. A manifold with boundary is a topological space that is Hausdorff, second-countable, and locally Euclidean of dimension n, except for points on the boundary which are only locally Euclidean of dimension $n-1$. A detailed discussion of manifolds with boundary can be found in Lee (2011).

Differentiable Manifold

We introduce differentiable manifolds in this section. As we introduced above, the chart maps $\phi_U : U \subset \mathcal{M} \to \mathbb{R}^n$ are defined on the local neighborhood U.

Definition 2.7. *For every pair of charts φ_U and φ_V defined on U and $V \subset \mathcal{M}$, respectively, with $U \cap V \neq \emptyset$, the transition function $\psi : \varphi_U(U \cap V) \subset \mathbb{R}^n \to \varphi_V(U \cap V) \subset \mathbb{R}^n$ is defined by $\psi = \varphi_V \circ \varphi_U^{-1}$. If the transition function is infinitely differentiable, meaning it has continuous partial derivatives of all orders, for every pair of charts φ_U and φ_V, then we call $\mathcal{M}$ an n-dimensional differentiable manifold.*

As we will discuss in later chapters, we extend the definition of differentiable maps for various use cases, such as real-valued functions on a manifold $f : \mathcal{M} \to \mathbb{R}$, vector-valued functions $f : \mathcal{M} \to \mathbb{R}^m$, and even from one manifold to another $f : \mathcal{M} \to \mathcal{N}$. Here we will only present the two most useful definitions of differentiable maps.

Definition 2.8. $f : \mathcal{M} \to \mathbb{R}$ *is called* $C^\infty(\mathcal{M}, \mathbb{R})$ *differentiable if for every chart* φ_U*, the function* $f \circ \varphi_U^{-1}$ *is differentiable.*

Definition 2.9. *The map from one manifold to another* $f : \mathcal{M} \to \mathcal{N}$ *is called* $C^\infty(\mathcal{M}, \mathcal{N})$ *differentiable if* $\forall r \in C^\infty(\mathcal{N}, \mathbb{R})$, $r \circ f \in C^\infty(\mathcal{M}, \mathbb{R})$.

With the definition of a differentiable structure, we can introduce tangent vectors and tangent spaces for a given manifold $\mathcal{M}$. For every point $p \in \mathcal{M}$, we define an n-dimensional real vector space $T_p\mathcal{M}$ that is isomorphic to $\mathbb{R}^n$. Each element of the tangent space $v \in T_p\mathcal{M}$ is a tangent vector of a smooth curve on $\mathcal{M}$ that passes through the point p.

Definition 2.10. *First define a chart* $\varphi_U : U \to \mathbb{R}^n$ *where* $p \in U$ *and* U *is a open set. And let* $\gamma : [-1, 1] \to \mathcal{M}, \gamma(0) = p$*. The* ***tangent vector*** *to the curve* γ *at* $t = 0$ *is defined as* $v = (\varphi_U \circ \gamma)'(0)$ *which maps function* φ_U *to its directional derivative.*

Definition 2.11. *Further, the collection of all tangent vectors at* $p \in \mathcal{M}$ *is the* ***tangent space*** *denoted by* $T_p\mathcal{M}$*.*

Definition 2.12. *The* ***tangent bundle*** *of* $\mathcal{M}$ *is defined as the disjoint union of tangent spaces at all points of* $\mathcal{M}$*, denoted by* $T\mathcal{M} = \coprod_{p \in \mathcal{M}} T_p\mathcal{M}$*.*

Riemannian Manifold

Finally, we will define Riemannian manifolds in this section.

Definition 2.13. *A* ***Riemannian manifold*** $(\mathcal{M}, g)$ *is a differentiable manifold* $\mathcal{M}$ *equipped with a Riemannian metric* g*. The* ***Riemannian metric*** g *is defined by a local inner product on tangent vectors*

$$g_p(\cdot, \cdot) : T_p\mathcal{M} \times T_p\mathcal{M} \to \mathbb{R}, p \in \mathcal{M}$$

which satisfies the following properties:

1. $g_p(u, v) = g_p(v, u)$ *for all* $u, v \in T_p\mathcal{M}$

2. $g_p\left(\sum_{i=1}^n u_i, \sum_{j=1}^n v_i\right) = \sum_{i=1}^n \sum_{j=1}^n g_p(u_i, v_j)$
3. $g_p(u, u) \geqslant 0$
4. $g_p(u, u) = 0 \Leftrightarrow u = 0$

With the definition of a Riemannian metric, we can define the length of a tangent vector $v \in T_p\mathcal{M}$ as $\sqrt{g_p(v, v)}$. More importantly, we can define the length of a curve connecting two points p and q in $\mathcal{M}$.

Definition 2.14. *For a given curve $\gamma : [a, b] \to \mathcal{M}$, the length of γ is defined by $L(\gamma) = \int_a^b \sqrt{g_{\gamma(t)}(\dot{\gamma}(t), \dot{\gamma}(t))}dt$. $\dot{\gamma}(t)$ is the tangent vector of γ at t.*

And the distance between any two points p and q in $\mathcal{M}$ is defined as the infimum of the lengths of all possible piecewise smooth curves connecting p and q.

Definition 2.15. *The **geodesic distance** between two points $p, q \in \mathcal{M}$ is defined as*

$$d(p, q) = \inf_{\gamma \in \Gamma(p,q)} \int_0^1 \sqrt{g_{\gamma(t)}(\dot{\gamma}(t), \dot{\gamma}(t))}dt$$

where $\Gamma(p, q)$ is a set of all possible piecewise smooth curves that $\gamma(0) = p$ and $\gamma(1) = q$.

If the infimum is achievable by a smooth curve, we call it geodesic.

Exponential Map and Logarithm Map

Here, we will discuss two commonly used maps when dealing with geodesics: the exponential map and the logarithm map.

Theorem 2.1. *Let $\mathcal{M}$ be a differentiable manifold. Then for each $p \in \mathcal{M}$ and $v \in T_p\mathcal{M}$, there exists $\epsilon > 0$ and a geodesic $\gamma : (-\epsilon, \epsilon) \to \mathcal{M}$ such that $\gamma(0) = p$ and $\dot{\gamma}(0) = v$. $\dot{\gamma}(t)$ is the tangent vector of γ at t, as defined in Definition 2.14. Furthermore any two such geodesics agree wherever both are defined.*

The existence and uniqueness of the geodesic connecting two points in a Riemannian manifold can be proved using the existence and uniqueness of ordinary differential equations (ODEs). The reader can refer to Theorem 4.27 in the textbook by Lee (2018). We will denote this unique geodesic as γ_v, where $\dot{\gamma}(0) = v$.

Lemma 2.2. (***Rescaling Lemma***). *For every* $p \in \mathcal{M}, v \in T_p\mathcal{M}$, *and* $c, t \in \mathbb{R}$,

$$\gamma_{cv}(t) = \gamma_v(ct)$$

whenever either side is defined.

Using the rescaling lemma, we can establish a mapping from a subset of the tangent bundle to $\mathcal{M}$. This mapping assigns a geodesic to each line passing through the origin in $T_p\mathcal{M}$. We will now introduce the exponential map.

Definition 2.16. *Let a subset* $\mathcal{E} \subseteq TM$ *be the domain of the exponential map, that*

$$\mathcal{E} = \{v \in TM : \gamma_v \textit{ is defined on an interval containing } [0,1]\}$$

and the ***exponential map*** $\exp : \mathcal{E} \to \mathcal{M}$ *by*

$$\exp(v) = \gamma_v(1)$$

This is a very general definition of the exponential map. However, in this thesis, we will use the **restricted exponential map at** p, denoted by $\exp_p$, which is the restriction of exp to the set $\mathcal{E}_p = \mathcal{E} \cap T_p\mathcal{M}$. We will use the term "exponential map" in the thesis to refer specifically to the restricted exponential map at p.

We would like to point out that the exponential map is only locally defined in most cases due to the fact that the existence and uniqueness of ODEs are only locally preserved. Since the exponential map is a local diffeomorphism, there is an inverse map within a small neighborhood. We call the inverse map the **logarithm map**.

Definition 2.17. *Given two points* $p, q \in \mathcal{M}$*, if there exists* $v \in T_p\mathcal{M}$ *and* $\exp_p(v) = q$*, we define the* ***logarithm map*** *as:*

$$\log_p(q) = v$$

2.2 Statistical Background

In this thesis, we will discuss several applications where we also need to consider the statistical performance of the given model. Therefore, we will cover some basic knowledge of correlation and statistical testing, which play an essential role in the following chapters.

Correlation

For two given random variables X and Y that are real-valued $X, Y \in \mathbb{R}$, we can analyze their statistical dependence. One common method for measuring the dependence is through the use of a correlation coefficient $\rho_{X,Y}$. The concept of independence between two random variables X and Y can be defined as the absence of any influence of one variable on the other. This concept of independence leads to the following theorem.

Theorem 2.3. *If* X *and* Y *are independent, then* $\rho_{X,Y} = 0$*.*

In general, the converse of the theorem is not true. However, there are some special cases where zero correlation does imply independence. For example, if X and Y are jointly normally distributed, then zero correlation between X and Y implies that X and Y are independent.

To compute the correlation coefficient $\rho_{X,Y}$, a first attempt may be to detect linear dependence between these two variables using the Pearson correlation coefficient, also known as the product moment correlation coefficient Pearson (1895).

Definition 2.18. (***Pearson correlation***). *For a given pair of random variables* $X, Y \in \mathbb{R}$*, the Pearson's correlation coefficient, when applied to a population, is computed as following:*

$$\rho_{X,Y} = \frac{cov(X,Y)}{\sigma_X \sigma_Y} = \frac{\mathbb{E}[(X-\mu_X)(Y-\mu_Y)]}{\sigma_X \sigma_Y}$$

where, μ_X, μ_Y *are the means of* X, Y*, and* σ_X, σ_Y *are the corresponding standard deviations. We have* $\sigma_X^2 = \mathbb{E}[(X-\mu_X)^2]$*.*

The Pearson correlation coefficient, denoted by ρ, measures the linear dependence between two random variables, X and Y. Its value ranges from -1 to 1, where $\rho = \pm 1$ indicates a perfect positive or negative linear correlation, respectively. In other words, when $\rho = \pm 1$, there exists a linear function of the form $Y = wX + r$ with $w, r \in \mathbb{R}, w \neq 0$. When $\rho = 0$, there is no linear association between X and Y, and the two random variables are said to be linearly independent.

Pearson correlation is a useful tool for measuring the linear dependence between two scalar random variables. However, when dealing with high-dimensional data, where the variables are vectors of length greater than one, the concept of Pearson correlation becomes more complex. In this case, the covariance matrix $\Sigma_X = \mathbb{E}[(X-\mu_X)(X-\mu_X)^T]$ is no longer a scalar but a matrix. So the Pearson correlation coefficient is not a scalar, but a matrix in high dimensional data, which is harder to interpret. Therefore, other correlation measures may be more suitable in high-dimensional settings. Canonical-correlation analysis (CCA) Hotelling (1936); Hardoon et al. (2004) is indeed a useful tool for analyzing the correlation between two sets of high-dimensional random variables X, Y even if they are in different dimensions.

Given two random variables $X \in \mathbb{R}^n$ and $Y \in \mathbb{R}^m$, we may want to find an index that describes any possible dependence between them. One approach to finding such an index is through the use of canonical correlation analysis (CCA), which is based on linear combinations of the variables, i.e. $a^T X$ and $b^T Y$, where $a \in \mathbb{R}^n$ and $b \in \mathbb{R}^m$. Canonical correlation analysis tries to find two vectors $a \in \mathbb{R}^n$ and $b \in \mathbb{R}^m$ such that the random variables $U = a^T X$ and $V = b^T Y$ maximize their Pearson correlation $\rho = \text{corr}(U, V)$. We call U and V the first pair of canonical variables. We

may also estimate another pair of projection vectors that is uncorrelated with the first pair. This can be repeated until we reach the minimum of m and n.

Definition 2.19. (***Canonical-correlation analysis*** (***CCA***)). *Let Σ_{XY} be the cross-correlation matrix between random vectors X, Y. We have*

$$\rho = \max_{a,b} \frac{a^T \Sigma_{XY} b}{\sqrt{a^T \Sigma_{XX} a}\sqrt{b^T \Sigma_{YY} b}}$$

In equation 16.5-16.8 of Härdle and Simar (2015), the solution of the above equation is

$$U = c^T \Sigma_{XX}^{-\frac{1}{2}} X = a^T X$$
$$V = d^T \Sigma_{YY}^{-\frac{1}{2}} Y = b^T Y$$

where c is the eigenvector associated with the largest eigenvalue of $\Sigma_{XX}^{-\frac{1}{2}} \Sigma_{XY} \Sigma_{YY}^{-\frac{1}{2}} \Sigma_{YX} \Sigma_{XX}^{-\frac{1}{2}}$, and d is proportional to $\Sigma_{YY}^{-\frac{1}{2}} \Sigma_{YX} \Sigma_{XX}^{-\frac{1}{2}} c$. The subsequent pairs are found by using eigenvalues of decreasing magnitudes.

The CCA method is effective because it can be applied to high-dimensional data. However, there are two major drawbacks of CCA. The first is that CCA evaluates linear dependency, similar to Pearson correlation. The second drawback is related to the computation of a and b, which requires an estimation of Σ_{XX}, Σ_{XY}, and Σ_{YY}. In real-world use cases, these covariance matrices may change dynamically. For instance, when training two networks with parameters Θ_X and Θ_Y and wanting to analyze the correlation of their representations, CCA may be used. However, this can be difficult because the networks are updated at each training step, resulting in dynamic changes to X and Y. To use the CCA method, we would need to estimate the covariance matrices Σ_{XX}, Σ_{XY}, and Σ_{YY} accordingly. However, this estimation may be challenging due to the dynamic changes in X and Y during the training process, although some solutions have recently been proposed Meng et al. (2021a).

There is another correlation concept that is well defined in high dimensional data and can measure the non-linear dependence. This is called distance correlation.

For an observed random sample $(x, y) = \{(X_i, Y_i) : i = 1, \cdots, n\}$ from the joint distribution of random vectors X in $\mathbb{R}^m$ and Y in $\mathbb{R}^n$, compute the following:

$$a_{k,l} = \|X_k - X_l\|, \quad \bar{a}_{k,\cdot} = \frac{1}{n}\sum_{l=1}^{n} a_{k,l}, \quad \bar{a}_{\cdot,l} = \frac{1}{n}\sum_{k=1}^{n} a_{k,l},$$

$$\bar{a}_{\cdot,\cdot} = \frac{1}{n^2}\sum_{k,l=1}^{n} a_{k,l}, \quad A_{k,l} = a_{k,l} - \bar{a}_{k,\cdot} - \bar{a}_{\cdot,l} + \bar{a}_{\cdot,\cdot}$$

where $k, l \in \{1, \cdots, n\}$. Similarly, we can define $b_{k,l} = \|Y_k - Y_l\|$, and $B_{k,l} = b_{k,l} - \bar{b}_{k,\cdot} - \bar{b}_{\cdot,l} + \bar{b}_{\cdot,\cdot}$. Then we can define:

Definition 2.20. (***Distance correlation***)**.** *Székely et al.* (*2007*) *The empirical distance correlation* $\mathcal{R}_n(x, y)$ *is the square root of*

$$\mathcal{R}_n^2(x, y) = \begin{cases} \frac{\mathcal{V}_n^2(x,y)}{\sqrt{\mathcal{V}_n^2(x,x)\mathcal{V}_n^2(y,y)}} & , \mathcal{V}_n^2(x,x)\mathcal{V}_n^2(y,y) > 0 \\ 0 & , \mathcal{V}_n^2(x,x)\mathcal{V}_n^2(y,y) = 0 \end{cases}$$

where the empirical distance covariance (*variance*) $\mathcal{V}_n(x, y), \mathcal{V}_n(x, x)$ *are defined as* $\mathcal{V}_n^2(x, y) = \frac{1}{n^2}\sum_{k,l=1}^{n} A_{k,l}B_{k,l}$, $\mathcal{V}_n^2(x, x) = \frac{1}{n^2}\sum_{k,l=1}^{n} A_{k,l}^2$.

We will discuss this idea in more detail in the Chapter 6.

Permutation Test

When conducting experiments or observational studies, data is typically gathered with multiple observations on a particular parameter of interest within a population. In order to draw any conclusions about this parameter, a hypothesis test is formulated based on the sample data Casella and Berger (2002) as follows:

$$H_0 : \theta \in \Theta_0, \text{ versus } H_1 : \theta \in \Theta_0^c$$

where θ denotes a population parameter, Θ_0 is some subset of the parameter space, and Θ_0^c is its complement.

Definition 2.21. *The two hypotheses H_0, H_1 are complementary hypotheses in a hypothesis test. H_0 is called the **null hypothesis** and H_1 is called the **alternative hypothesis**.*

After observing the samples, the experimenter must decide either to accept the null hypothesis H_0 as true or to reject H_0 as false.

Definition 2.22. *A hypothesis testing procedure (hypothesis test) is a rule that specifies:*

1. *For which sample values the decision to accept H_0 is true*
2. *For which sample values H_0 is rejected and H_1 is accepted as true*

There are different methods of finding test procedures. We will discuss permutation test here. Readers who are interested in other methods can refer to the book Casella and Berger (2002) for more information.

A permutation test is a type of nonparametric statistical test used to assess the significance of a parameter or a difference between parameters Good (2000). The permutation test first requires that the data to be exchangeable under H_0.

Definition 2.23. *The data $X_1, \cdots, X_n$ are said to be exchangeable if any new data $X_{i(1)}, \cdots, X_{i(n)}$, where their labels $i(1), \cdots, i(n)$ are generated by rearranging the labels of original data $1, \cdots, n$, produces the same joint probability distribution as the original one.*

We consider the following scenario: we are given two groups, A and B, and the null hypothesis is that A and B have the same distribution. In order to test this hypothesis, we collect two random variables, X_A and X_B, from groups A and B, respectively.

Definition 2.24. (*Permutation test*). *The permutation test is performed in the following way:*

1. *Compute the test statistic T^* for X_A, X_B, such as the difference between the means of X_A, X_B.*

2. (*Permutations*). *In step* i, *pool the observation* X_A, X_B *and divide the pooled values into two groups of the same size of* X_A, X_B. *Compute and record the new test statistic* T_i.

3. *Repeat the above process* m *times to form the distribution*

The one-side p-value is calculated as the proportion of sampled permutations where the difference in means was greater than the original difference.

$$\text{p-value} = \frac{1}{m}\sum_{i=1}^{m} I\{T_i \geqslant T^*\}$$

where $I\{\cdot\}$ is an indicator function. And the two-side p-value of the test is calculated as the proportion of sampled permutations where the absolute value of difference in means was greater than the absolute value of original difference.

$$\text{p-value} = \frac{1}{m}\sum_{i=1}^{m} I\{|T_i| \geqslant |T^*|\}$$

2.3 Deep Learning Models

This thesis deals with deep learning for manifold-valued data. In this section, we will introduce some well-known deep learning models for Euclidean data, which will serve as a basis for proposing modifications to enable these models to operate on manifold-valued data.

Convolutional Neural Network

To begin with, we will introduce the convolutional neural network (CNN). CNNs LeCun et al. (1989); Krizhevsky et al. (2012); Simonyan and Zisserman (2015) use the convolution operation (2D convolution in most natural image applications) within many layers of the network. For the convolution layer, one must define the kernels K, which mathematically define the receptive field of the filter. Normally,

the spatial size of the kernel is $k \times k$. To make the following discussion easier, we assume that k is an odd number $k = 2\tau + 1$. Assume that we are given a gray-scale image $\mathcal{I}$ with size 224×224. Let us omit handling the boundary of the image aside for the moment. Then, the convolution operator at location i, j is:

$$\mathrm{out}(i, j) = \sum_{p,q=0}^{k-1} \mathcal{I}(i - \tau + p, j - \tau + q)K(p, q)$$

One can notice that this operation is slightly different from the textbook definition of convolution since the convolution requires the flip of K. But since K is normally trainable, the flip is not critically relevant for practical purposes, and this allows us to simplify the discussion.

When the input image is not a gray-scale image, for example, an RGB color image, there will be an extra dimension to store the color information, referred to as channels. When considering the channel dimension, the convolution kernel K should be $k \times k \times c_{in}$, where c_{in} is the channel size. The convolution will be:

$$\mathrm{out}(i, j) = \sum_{p,q=0}^{k-1} \sum_{r=0}^{c_{in}-1} \mathcal{I}(i - \tau + p, j - \tau + q, r)K(p, q, r)$$

Then if we concatenate c_{out} number of different kernels with the same window size k, we obtain the final convolution layer, which takes the input with channel c_{in} and the output with channel c_{out}.

The key benefit of CNNs is that the kernel is shared at every location of the image. Thus, the features extracted by these kernels should have the property that they are invariant to location translation. Since objects can appear at any location within the image, CNNs are naturally suitable to problems or applications involving natural images.

Residual Neural Network

In early experiments with CNNs, researchers realized that, in general, the deeper networks yield better final performance. The assumption is that if the network Θ_A is deeper than Θ_B and the initial layers of Θ_A share the same structure as Θ_B, then Θ_A can finally learn the same weights as Θ_B in the initial layers and the remaining layers can learn the identity map. Then, the performance of Θ_A is at least similar to Θ_B. But the experimental results show that if the CNN is too deep, the performance will be lower than a shallower CNN. The analysis shows that the network experiences the gradient vanishing/ explosion during training when it becomes very deep Glorot and Bengio (2010).

This led to the design of the Residual Neural Network (Resnet) He et al. (2016a). Let us call each layer of the neural network f_i. Then, the normal CNN will be $f = f_n \circ f_{n-1} \circ \cdots f_0$. The Resnet brings the skip connection into the design, such that the input of the next layer is not only the network output but also the input data x itself.

$$g_i(x) = f_i(x) + x$$
$$g = g_n \circ g_{n-1} \circ \cdots g_0$$

The above equation shows the skip connection for every single layer. In real-world implementations, Resnet models are implemented with double- or triple-layer skips with nonlinear activation layers in between. An example figure is shown in the Figure 2.1.

The key benefit is that the training of a deeper network can be stable, and gradient vanishing/ explosion problems are minimized or eliminated. Another benefit is that the network layers g_i can be learned to be identical maps if f_i is all-zero. Thus, the deeper network has the potential to be "at least" as good as a shallower network.

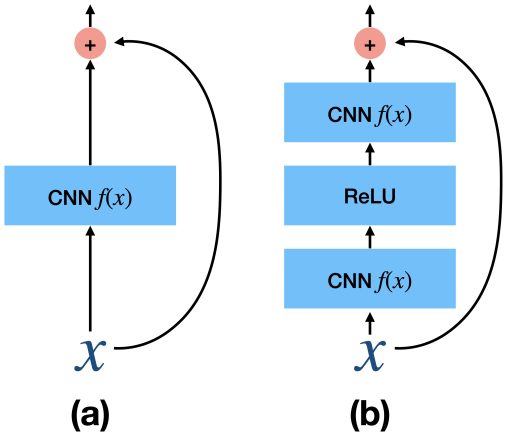

Figure 2.1: A demonstration of Resnet. (a) the skip connection for every single layer. (b) skip connection for every two layers with an activation layer in between.

Dilated Convolutional Neural Network

If the input data is a long sequence of 1D data, $x = (x_0, x_1, \cdots x_{t-1})$, using CNNs poses some challenges. The main drawback is the receptive field size. If we do not consider the pooling layer, and the kernel size is $k = 2\tau + 1$ at every layer, then we can notice that in the second layer, each pixel output can view $k = 2\tau + 1$ pixels in the first layer. For the third layer, each pixel output can view $k = 2\tau + 1$ pixels in the second layer. However, since the kernels are shared, and the stride is 1 by default, each pixel output in the third layer can only cover $2\tau + 1 + 2\tau = 4\tau + 1$ pixels in the first layer. And we can continue this trend where in the n_{th} layer, each pixel can view $2(n-1)\tau + 1$ pixels in the first layer. This will cause some problems as the receptive field is linearly dependent on the number of layers. Thus, to cover the entire t steps of the input data, the network would need $O(t)$ number of layers, which will be too deep in most applications.

On the other hand, if there are some holes between consecutive elements of kernels of a CNN model, it can increase the receptive field significantly. This idea is called dilated CNN.

Definition 2.25. (***Dilated Convolutions***). *Bai et al. (2018). Given a 1-D input se-*

quence $x : N \to \mathbb{R}^t$ *and a kernel* $K : \{0, \cdots, k-1\} \to \mathbb{R}$*, the dilated convolution function* $(x_{\star_d} K) : N \to \mathbb{R}$ *is defined:*

$$(x_{\star_d} K)(s) = \sum_{i=0}^{k-1} K(i) x(s - id)$$

When $d = 1$, this is equivalent to the standard convolution operator. In a dilated CNN, the receptive field size will depend on the network's depth and the choice of k and d. And if d expands exponentially as the network goes deeper, the receptive field of the network will also increase exponentially. This operation automatically handles the problem that the network will be too deep if the input data is too long. Actually, the depth of the network will be $O(\log(t))$ in this setup, which is much shallower than $O(t)$ for the normal CNN.

Normalizing Flow Models

The network structures discussed above are the basic components of designing a neural network. For example, we can apply a CNN on the image dataset with class labels to perform classification, or we can use a similar structure as a CNN on the image dataset with a pixel-wise label to perform segmentation.

We can notice that the CNN (or Resnet) is not invertible in most cases. For example, for a convolutional layer with kernel K with no channel information, we write the convolution operation as $x_{out} = f(x_{in}, K)$ where x_{in} is the input, and x_{out} is the output. We know that

$$x_{out}(i, j) = \sum_{p,q=0}^{k-1} x_{in}(i - \tau + p, j - \tau + q) K(p, q)$$

But there is no trivial solution to solve x_{in} when given x_{out} and K. For example, if K is a 2D Gaussian kernel, f will be a blurring function, and f^{-1} will be an unblurring function, which is an underdetermined problem.

In this section, we will introduce a specific network structure, where both f and

f^{-1} are well defined. This is called normalizing flow Rezende and Mohamed (2015); Kingma and Dhariwal (2018). This type of architecture is often used to estimate the distribution of the given input data or generate new data as a generative model.

Flow-based generative models aim to optimize the log-likelihood of input data from an unknown distribution by using an invertible function to map the unknown distribution in the input space to a known distribution in the latent space. By doing so, the invertible function attempts to simplify the sampling process from the input space distribution, as sampling from a known distribution is generally more straightforward. This approach has been used in several studies, such as those by Rezende and Mohamed (2015); Kingma and Dhariwal (2018); Yang et al. (2019) using techniques like variational inference, Glow, and PointFlow.

The GLOW model Kingma and Dhariwal (2018) considers a scenario where $\{\mathbf{x}_i\}$ is a set of i.i.d. samples drawn from an unknown distribution $p^*(\mathbf{x})$ that is parameterized by θ. In the rest of this section, the distribution $p_\theta(\mathbf{x})$ is used as a substitute for $p^*(\mathbf{x})$. The objective is to learn θ based on a dataset $\mathcal{D}$ by maximizing the likelihood of the model given the dataset. This is achieved by minimizing the equivalent form of the negative log-likelihood.

$$\ell(\theta|\mathcal{D}) = \frac{1}{N}\sum_{i=1}^{N} -\log p_\theta(\mathbf{x}_i)$$

To minimize the expression above, knowledge of p_θ is required. However, one way to overcome this limitation is to use a mapping from a known distribution in the latent space. The latent space is represented as $\mathbf{z}$, and the generative step is formulated as $\mathbf{z} \sim p(\mathbf{z})$, $\mathbf{x} = g(\mathbf{z})$. A Gaussian distribution $\mathcal{N}(\mathbf{z}; \mathbf{0}, I)$ can be used as $p(\mathbf{z})$ in this approach.

From the above discussion, we know that f is the inverse of g (as a feed-forward network). For normalizing flow Rezende and Mohamed (2015), f is composed as a sequence of invertible functions $f = f_K \circ f_{K-1} \circ \ldots \circ f_1$. Hence, we have

$$\mathbf{x} \overset{f_1}{\longleftrightarrow} \mathbf{h}_1 \overset{f_2}{\longleftrightarrow} \mathbf{h}_2 \dots \overset{f_K}{\longleftrightarrow} \mathbf{z}$$

where $\mathbf{h}_t$ is the hidden representation at layer t. Using $\mathbf{h}_0 = \mathbf{x}$ and $\mathbf{h}_K = \mathbf{z}$, the log-likelihood of $p_\theta(\mathbf{x})$ is

$$\begin{aligned} \log p_\theta(\mathbf{x}) &= \log p(\mathbf{z}) + \log|\det(d\mathbf{z}/d\mathbf{x})| \\ &= \log p_\theta(\mathbf{z}) + \sum_{j=0}^{K} \log|\det(d\mathbf{h}_j/d\mathbf{h}_{j-1})| \end{aligned}$$

The GLOW model proposed in Kingma and Dhariwal (2018) consists of three layers, each with a Jacobian $d\mathbf{h}_j/d\mathbf{h}_{j-1}$ that is a triangular matrix. This choice of a triangular Jacobian matrix simplifies the log-determinant computation in the model.

$$\log|\det(d\mathbf{h}_j/d\mathbf{h}_{j-1})| = \sum(\log|\mathrm{diag}(d\mathbf{h}_j/d\mathbf{h}_{j-1})|)$$

The GLOW block is composed of three types of layers: (**a**) Actnorm, (**b**) Invertible 1×1 convolution, and (**c**) Affine Coupling layers. The input data is first squeezed before being fed into the block. The data is then split in a manner similar to the method proposed in Dinh et al. (2017).

(**a**) **Actnorm.** normalizes the input to be a zero-mean and identity standard deviation.

$$Y = \frac{1}{\sigma} \odot (X - \mu)$$

μ, σ are initialized from the data and then trained independently.

(**b**) 1×1 **convolution.** applies the invertible matrix R on the channel dimension.

$$R \times X$$

$X \in \mathbb{R}^{s_r \times c_r}$ and $R \in \mathbb{R}^{c_r \times c_r}$ where s_r is the resolution of the input variables while c_r is the number of channels.

(c) Affine Coupling. uses the idea of split+concatenation.

$$\begin{aligned} S, T &= NN(X_a) \\ Y_b &= S \odot X_b + T \\ Y_a &= X_a \end{aligned}$$

To obtain the final output Y, the input variable X is split along the channel into X_a and X_b. Then, Y_a and Y_b are concatenated. The matrices S and T are real-valued and have the same dimension as X_b, and are used for element-wise scaling and translation, respectively.

The layers mentioned above all have a closed-form inverse, which can be utilized during the generation phase.

3 DILATED CONVOLUTIONAL NEURAL NETWORKS FOR SEQUENTIAL MANIFOLD-VALUED DATA

In this chapter, motivated by neuroimaging applications, we study formulations where the data are *sequential manifold-valued measurements*. This case is common in brain imaging, as we discussed in Chapter 2. We will analyze fiber bundles that connect different gray matter regions in the brain. We will consider the fiber bundle, estimated via tractography, as a path and measurements along this path will provide information regarding anisotropy/diffusivity at each spatial location. These measurements correspond to symmetric positive definite matrices or orientation distribution functions. Instead of a recurrent model which poses computational/technical issues, and inspired by recent results showing the viability of dilated convolutional models for sequence prediction, we develop a dilated convolutional neural network architecture for this task. In Section 3.2, we show how the modules needed in our network can be derived while explicitly taking the Riemannian manifold structure into account. We show how the operations needed can leverage known results for calculating the weighted Fréchet Mean (wFM). Finally, in Section 3.4, we present results for group difference analysis in Alzheimer's disease (AD) where the groups are derived using AD pathology load Wasserthal et al. (2018): here the model finds several brain fiber bundles that are related to AD even when the subjects are all still cognitively healthy. The work presented in this chapter appeared as a conference paper in ICCV 2019 Zhen et al. (2019).

3.1 Introduction

The reader may recall that the classical definition of convolution assumes that the data are scalar or vector-valued and lie on discrete equally spaced intervals. This assumption is ideal for natural images and central to how we use convolutional filters in deep neural networks but is far less appropriate for other domains where the data are structured, such as meshes, graphs or measurements on a manifold.

As we discussed in Section 1.2, in computer vision and machine learning, such problems that need deep learning models for structured data, are studied under a topic called geometric deep learning Bronstein et al. (2017), which has led to a number of elegant approaches including convolutional neural networks (CNN) on non-Euclidean data Cohen et al. (2018); Kondor et al. (2018). The reason this is important is that mathematically, non-Euclidean data violates a number of key properties of Euclidean spaces such as a global linear structure and coordinate system, as well as shift invariance/equivariance. As a result, the core operations we use in classical statistics and machine learning as well as within deep neural network architectures often need to be tailored based on the geometry and specifics of the data at hand, as discussed in Chapter 2. When such adjustments are made in modern deep learning architectures, a number of authors have reported sizable improvements in the performance of the learning algorithms Chakraborty et al. (2022, 2018b); Kondor and Trivedi (2018); Cohen et al. (2018); Huang et al. (2018); Huang and Gool (2017); Cohen and Welling (2016b).

We should note that specializing learning methods to better respect or exploit the structure (or geometry) of the data is not a new development. Time series data are common in finance Tsay (2005), and as a result, has been analyzed using specialized methods in statistics for decades. Surface normal vectors on the unit sphere have been widely used in graphics Straub et al. (2015), and probability density functions, as well as covariance matrices, are common in both machine learning and computer vision Srivastava et al. (2007); Dominici (2002). In neuroimaging, which is a key focus of this chapter, the structured measurement at a voxel of an image may capture water diffusion Basser et al. (1994); Wang and Vemuri (2005); Lenglet et al. (2006); Jian et al. (2007); Aganj et al. (2009); Cheng et al. (2012) or local structural change Hua et al. (2008); Zacur et al. (2014); Kim et al. (2017). The latter example is commonly known as the Cauchy deformation tensor (CDT) Kim et al. (2017) and has been utilized to achieve improvements over brain imaging methods such as tensor-based morphometry Leporé et al. (2006); Ridgway (2009); Ashburner and Friston (2000). When the mathematical properties of such data are exploited, one often needs new loss functions and specialized optimization schemes. This

step often involves first defining an intrinsic metric for the underlying geometry (structure) of the data. It is important to note that within geometric deep learning for *manifolds*, two types of settings are often considered. The **first** case is where the data are functions on a manifold. The **second** case corresponds to the setting where data are sample points on a manifold, such as a Riemannian manifold. In this chapter, we study the second setting, which is not covered in the form described here in existing works including Bronstein et al. (2017).

When the structure or geometry of the data informs the formulation of the learning task (or algorithm), we obtain differential geometry inspired algorithms where the role of the extrinsic or intrinsic metric induced by the data is explicit. Many datasets do *not* have a temporal or sequential component associated with each sample. However, the analysis of temporal (or sequential) data is an important area of machine learning and vision, e.g., within action recognition Afsari et al. (2012); Bissacco et al. (2001); Turaga et al. (2008) and video segmentation He et al. (2012), the study of analogous geometric ideas in this regime, especially within deep learning, is limited. Specifically, there are few existing proposals describing deep neural network models for structured (or manifold-valued) *sequential* data. Recently in Chakraborty et al. (2018c), we proposed a *recurrent* model for the manifold of symmetric positive definite (SPD) matrices. This work replaces a number of blocks within a recurrent model with "statistical recurrent units". But it is known that training recurrent models is more involved than convolutional architectures – shortly, our experiments will show that a $2\times$ speed-up (by using a convolutional instead of a recurrent model) can be achieved (in Section 3.4). While the current consensus, within the community, is that sequential data should involve a recurrent network Elman (1990), as noted by Bai et al. (2018), emerging results indicate that convolutional architectures often perform superior to recurrent networks on "sequential" applications such as audio synthesis. In fact, even historically, convolutional models were used for 1-D *sequential* data Hinton (1989); LeCun et al. (1995). Now, given that most use-cases of learning sequential models on manifold-valued data will *not* require the infinite memory capabilities offered by a recurrent model, it seems natural to investigate the extent to which convolutional models may suffice.

Notice that in order to get the long effective memory from a CNN model, one needs to increase the depth and/or increase the receptive field: this is provided by extensions such as dilated convolutions. We find that the two key ingredients in Bai et al. (2018) to achieve similar or better performance than a recurrent model for sequential tasks involves (**a**) using dilations to increase the receptive field of each convolution and (**b**) using residual connections to design a deeper but stable network. It seems logical that these developments should be an ideal starting point in designing models and algorithms for **sequential manifold-valued data** – the goal of this work. Our key **contribution** is the design of a Dilated CNN model for sequential manifold-valued data and showing its applicability in performing statistical analysis of brain images, specifically, diffusion-weighted MR images. To do so, we (**a**) define dilation for the convolution operator on the manifold of interest (**b**) define residual connections for our architecture (**c**) define weight normalization/dropout to add regularization/stability for the deeper network. We show that this yields an efficient formulation for sequential manifold-valued data, where few exist in the literature at this time. On the scientific side, we show that such a construction gives us the ability to identify structural connectivity changes in asymptomatic individuals who are at risk for developing Alzheimer's disease (AD) but are otherwise cognitively healthy.

3.2 Preliminaries

The motivation of this work is the analysis of sequential manifold-valued data, using deep neural network architectures. As described above, our architecture utilizes ideas presented earlier in the context of dilated convolutional neural networks (DCNN) on Euclidean spaces Bai et al. (2018). We have reviewed the basic idea of DCNN in Section 2.3. Here, we will review the standard DCNN formulation in detail and then describe our proposed manifold-valued DCNN framework.

Dilated convolutions Bai et al. (2018). Given a 1-D input sequence $\mathbf{x} : \mathbf{N} \to \mathbb{R}^n$ and a kernel $w : \{0, \cdots, k-1\} \to \mathbb{R}$, the dilated convolution function $(\mathbf{x} \star_d w) : \mathbf{N} \to \mathbb{R}^n$

is:

$$(\mathbf{x} \star_{\mathrm{d}} w)(\mathrm{s}) = \sum_{\mathrm{i}=0}^{\mathrm{k}-1} w(\mathrm{i})\mathbf{x}(\mathrm{s} - \mathrm{i}\mathrm{d}), \tag{3.1}$$

where $\mathbf{N}$ is the set of natural numbers, and k and d are the kernel size and the dilation factor respectively. Notice that with $\mathrm{d} = 1$, we get the normal convolution operator. In a dilated CNN, the receptive field size will depend on the depth of the network as well as on the choice of k and d. Thus, the authors in Bai et al. (2018) suggested the use of *residual connections* He et al. (2016a) – this was found to provide stability for deeper networks. Notice that, unlike the standard residual network connection, here the authors used a 1×1 convolution layer in order to match the width of the input and the output. Additionally, in order to regularize the network, the authors used *weight normalization* Salimans and Kingma (2016) and *dropout* Srivastava et al. (2014). The weight normalization was applied to the kernel of the dilated convolution layer. The dropout was implemented by randomly zeroing out an entire output channel of a dilated convolution layer. Finally, as an activation function, the authors used ReLU non-linearity. A schematic diagram of a standard dilated CNN is given in Figure 3.1.

Next, we discuss generalizing the operations needed within a DCNN so that they can operate on manifold-valued data. Specifically, we will generalize the following operations: (**a**) Dilated convolution (**b**) Residual connection (**c**) Weight Normalization (**d**) ReLU and (**e**) Dropout, to the setting where data are manifold-valued.

Some time ago, in Chakraborty et al. (2022), we proposed a CNN architecture for manifolds and/or manifold-valued data. We can utilize some of these ideas towards deriving the dilated convolution operation. Before discussing the details of the definition of dilated CNN for manifold-valued data, we will first introduce some notations, concepts, and terminology. Readers can find more detailed notation and definition in Section 2.1.

Assumptions. We follow the same notation as Section 2.1. We use $(\mathcal{M}, \mathrm{g})$ to denote

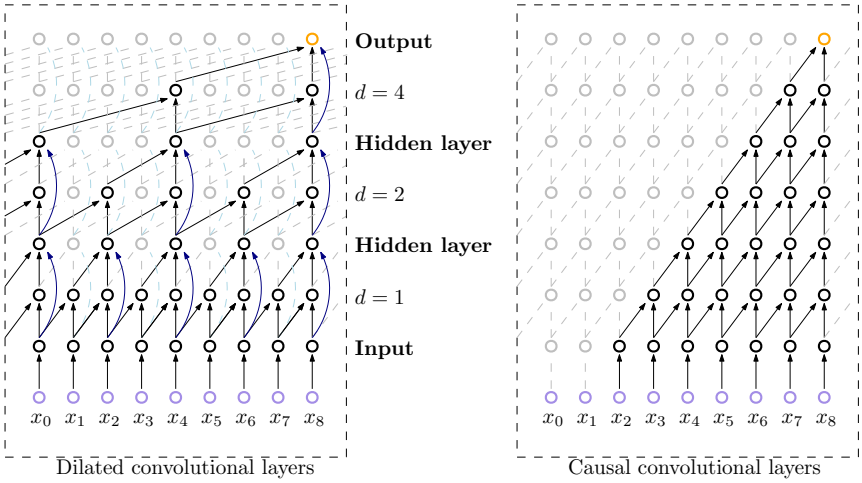

Figure 3.1: Schematic diagram of dilated CNN and causal CNN (see Bai et al. (2018) for definition and additional description).

a Riemannian manifold $\mathcal{M}$ with the Riemannian metric g and $d_{\mathcal{M}} : \mathcal{M} \times \mathcal{M} \to [0, \infty)$ denotes the distance induced by the metric g. We assume that the samples on $\mathcal{M}$ lie inside a regular geodesic ball of radius r centered at p, $\mathcal{B}_r(p)$, for some $p \in \mathcal{M}$ and $r = \min\left\{r_{cvx}(\mathcal{M}), r_{inj}(\mathcal{M})\right\}$. Here, r_{cvx} and r_{inj} are the convexity and injectivity radius of $\mathcal{M}$, see Groisser (2004).

Weighted Fréchet mean (wFM). Let $\{X_i\}_{i=1}^N$ be samples on $\mathcal{M}$. In Chakraborty et al. (2022) we define the convolution operation using the weighted Fréchet mean (wFM) Maurice Fréchet (1948) of $\{X_i\}$. Consider a one dimensional kernel $\{w(i)\}_{i=1}^N$ satisfying the convexity constraint, i.e.,

(1) $\forall i, w(i) > 0$

(2) $\sum_i w(i) = 1$

Then, the wFM (uniqueness is guaranteed by the statement above) is defined as:

$$\mathsf{wFM}\left(\{X_i\}, \{w\}\right) = \arg\min_{M} \sum_{i=1}^{N} w(i) d_{\mathcal{M}}^2(X_i, M), \tag{3.2}$$

Note that the distance metric $d_{\mathcal{M}}(X_i, M)$ follows the Definition 2.15.

Group of isometries. The set $I(\mathcal{M})$ of all isometries of $\mathcal{M}$ forms a group with respect to function composition. We will use G to denote this group and for $g \in G$, and $X \in \mathcal{M}$, let $g.X$ denote the result of applying the isometry g to point X ('.' simply denotes the group action).

Key application focus. Diffusion-weighted imaging (DWI) is a magnetic resonance imaging (MRI) technique that measures the diffusion of water molecules to generate contrast in MRI, and has been widely applied to measure the loss of structural connectivity in the brain. At each voxel in the image, water diffusion can be variously represented: two common options are using an elliptical approximation (see Figure 3.2(a)) where a 3×3 covariance matrix expresses the diffusivity properties or an orientation distribution function where one represents the probability densities of water diffusion over different orientations. We discussed this acquisition briefly in Example 1.6. Here, we will introduce more about what we can do with the DTI images. We can first divide the 3D image into anatomically meaningful parcels in Figure 3.2(b) and then run standard tractography routines to estimate the strength of connectivity between each pair of anatomical parcels Raamana and Strother (2018). The fiber bundles, hence estimated, are shown in Figure 3.2(c). For analysis, one often focuses on certain important fiber bundles instead of analyzing the full set of fibers. Notice that if we specify a starting and ending anatomical region for a fiber bundle, we can consider the corresponding covariance matrices encountered on this "path" as multi-variate manifold-valued measurements of this function. This is precisely the type of sequential manifold-valued data that we will seek to model in this chapter.

3.3 Dilated Convolutions for Manifold-valued Measurements

We now describe how to obtain the specific components needed in our architecture for manifold-valued data.

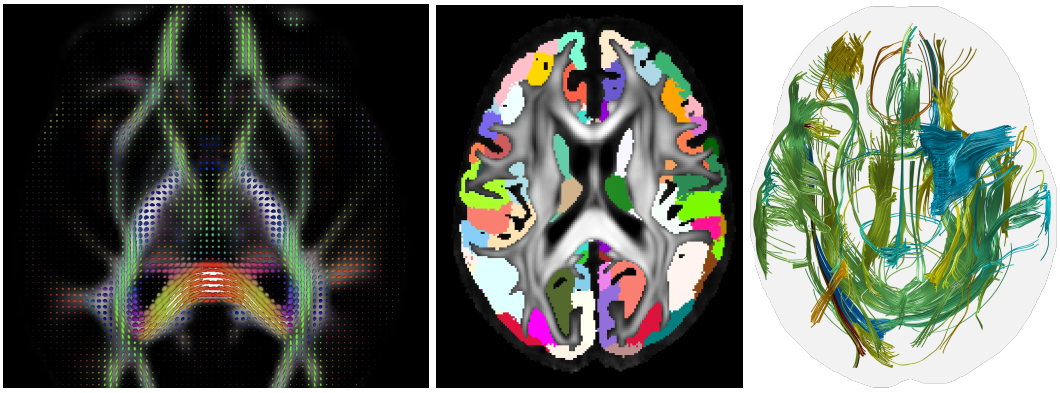

Figure 3.2: (*Left-Right*) (a) Diffusion MRI visualizes each voxel as an ellipsoid, measuring the water molecule's ability to diffuse. (b) Parcels are specific locations within the brain that share similarity in one or more properties and are also referred to as regions. (c) Fiber bundles display the connectivity between different brain regions.

Dilated convolution operator. Given a 1-D input sequence $X : \mathbf{N} \to \mathcal{M}$ and a kernel $w : \{0, \cdots, k-1\} \to \mathbb{R}$ satisfying the convexity constraint, the dilated convolution function $(X \star_d w) : \mathbf{N} \to \mathcal{M}$ is defined as:

$$(X \star_d w)(s) = \arg\min_{M} \sum_{i=0}^{k-1} w(i) d^2_{\mathcal{M}}(X(s - id), M), \tag{3.3}$$

where as before, k and d are the kernel size and dilation factor respectively. Observe that the convexity constraint on the kernel is merely to ensure that the result also lies on the manifold. We will use the weighted Fréchet mean (wFM) as a dilated convolution operator. This choice is mathematically justified because

(1) (3.1) is the minimizer of the weighted variance which is wFM, if the choice of distance is the ℓ_2 distance.

(2) We will show in Proposition 3.1 that the dilated convolution operator is equivariant to the action of G

This is a direct analog of its Euclidean counterpart. Notice that the dilated convolu-

tion operator defined in (3.1) is equivariant to translations, i.e., if $\mathbf{x}$ is translated by some amount $\mathbf{t}$, so is the result $(\mathbf{x} \star_d w)$. On the manifold $\mathcal{M}$, the analog of translation is the action of G, hence the equivariance of $(X \star_d w)$ with respect to G is a desirable property.

Proposition 3.1. *Using notations in* (3.3) *and given w satisfying the convexity constraint, let $F : X \mapsto (X \star_d w)$. Then, F is G-equivariant, i.e., F is equivariant to the action of G.*

Proof. Observe that, if $g \in G$ acts on X, then, $X(s - id) \mapsto g.X(s - id)$, for all s, d, i. Since g is an element of isometry group, therefore, $d_{\mathcal{M}}(g.X(s - id), g.M) = d_{\mathcal{M}}(X(s - id), M)$, for all $M \in \mathcal{M}$. So, $g.M = (g.X \star_d w)(s)$ iff $M = (X \star_d w)(s)$, which concludes our proof. □

In (3.3), since $(X \star_d w)$ is a $\mathcal{M}$ valued function, we will use M as a manifold-valued function, i.e., $M(s) = (X \star_d w)(s)$. Similar to the Euclidean dilated convolution layer, we learn multiple dilated kernels (given by the number of output channels) for a dilated convolutional layer.

Residual connection. Let X and F be the input and output of a dilated convolutional layer where the numbers of channels are c_{in} and c_{out}. Then, analogous to the Euclidean residual connection, we define the residual connection using two steps:

(1) First, concatenate X and $F(X)$ to get $(c_{in} + c_{out})$ number of channels.

(2) Use wFM to extract c_{out} number of outputs.

More formally, let $R(X, F_X)$ be the output of the residual connection, then the k^{th} channel of the residual connection, $R_k(X, F(X))$ is given by:

$$R_k(X, F(X))(s) \overset{def}{\equiv} \arg\min_{M} \left(\sum_{i=1}^{c_{in}} w_k(i) d^2_{\mathcal{M}}(X_i(s), M) + \sum_{j=1}^{c_{out}} w_k(j + c_{in}) d^2_{\mathcal{M}}(F_j(s), M) \right),$$
$$\text{s.t.} \sum\nolimits_i w_k(i) = 1, \forall w_k(i) > 0, \tag{3.4}$$

where, $k \in \{1, \cdots, c_{out}\}$ and X_i and F_j denotes the i^{th} and j^{th} channel of X and F respectively.

Weight normalization, ReLU, and dropout. The weight normalization in the standard Euclidean convolutional network is not needed here since we impose a convexity constraint on the kernel. We argue that since Dropout is a regularizer, we will not use dropout for our manifold-valued DCNN implementation because of the implicit regularization due to the convexity constraint. As argued in Chakraborty et al. (2022), wFM is both

(**1**) a contraction mapping Chakraborty et al. (2022) and

(**2**) a nonlinear mapping

and hence ReLU or any other non-linearity is not strictly necessary. Here, similar reasoning explains why a ReLU is not strictly needed (since the contraction and non-linear mapping are provided directly by wFM).

Equivariance and invariance. A few reasons why convolutional networks are so powerful are

(**1**) translational equivariance of a convolution layer and so, weights can be shared across an image

(**2**) translational invariance property of the entire convolutional network which is the property of the fully connected last layer.

As we showed above, the way we defined our dilated convolution operator leads to equivariance to the action of G. But we still have not shown that the *last layer* can be designed in a way that the output of the network does not change with respect to the action of G. So, we still need an analogous G-invariant last layer.

Invariant last layer. Analogous to the Euclidean recurrent model/ dilated CNN, in the last layer we will only consider the output of the last time point of a sequence, i.e., if X is the output of the last dilated convolutional layer with c number of channels, then the input of our last layer is $\{X_i(N)\}_{i=1}^{c}$, where $X(N) \in \mathcal{M}$ is the value of the last time point. We know already that $\{X_i(N)\}$ are G-equivariant. So, in order to make the entire dilated convolutional network G invariant, we need an invariant last layer. This is analogous to the translational invariant property of a fully connected (FC)

layer in the traditional (Euclidean) dilated CNN. We design our last invariant layer as follows: **(a)** We will first learn n_C number of wFM (let denoted by $\{\mu_i\}_{i=1}^{n_C}$) of $\{X_i(N)\}_{i=1}^{c}$ using (3.2), where n_C is a hyperparameter. **(b)** For all $i \in \{1, \cdots, c\}$, and for all $j \in \{1, \cdots, n_C\}$, we compute the distance between $X_i(N)$ and μ_j, denoted by d_{ij}. **(c)** Thus, for each $X_i(N)$, we get n_C number of feature representations. **(d)** We will use a standard fully connected (FC) layer with $c \times n_C$ features as input and the desired number of outputs.

Proposition 3.2. *The last layer is* G*-invariant.*

Proof. Observe that $d_{ij} = d_{\mathcal{M}}(X_i(N), \mu_j)$. From Proposition 3.1, we know that μ_j is G-equivariant, hence, $\mu_j \mapsto g.\mu_j$, for some $g \in G$ if $\forall i, X_i(N) \mapsto g.X_i(N)$. But, $d_{\mathcal{M}}(X_i(N), \mu_j) = d_{\mathcal{M}}(g.X_i(N), g.\mu_j)$, which concludes the proof. □

In order to reduce the number of parameters in the last layer, we propose a parameter efficient last layer which is defined as using a FC layer on the tangent space, i.e., input $\{\mathrm{Log}(X_i(N))\}_{i=1}^{c}$ as input to the FC layer, where Log is the Riemannian inverse exponential map.

Now, we have all components of our dilated CNN on manifold-valued data. A schematic of our model is shown in Figure 3.3. The building block for a 2-layer manifold DCNN is shown in Algorithm 3.1. Note that the network parameters are scalar-valued, with a convexity constraint. In order to enforce the convexity constraint, i.e., $\{w(i)\} \geqslant 0$ and $\sum_i w(i) = 1$, we will learn $\left\{\sqrt{w(i)}\right\}$, which can be any real value. We will enforce the sum constraint by normalization. Thus we will use SGD to learn $\left\{\sqrt{w(i)}\right\}$.

3.4 Experiments

In this section, we apply the manifold DCNN to answer the following questions:

(1) By replacing a RNN with our DCNN with a manifold constraint, what improvement in terms of the number of parameters/time can we achieve, without sacrificing performance?

Algorithm 3.1: A basic i^{th} DCNN building block with two convolution layers

Input: $N, c^1_{in}, c^1_{out}, c^2_{out}, c_{res}, k_1, d_1, k_2, d_2, nC, c$
Output: y_o

$x^{i-1} \leftarrow \text{Input}(c^1_{in}, N)$
$y_1 \leftarrow \text{Dilated\_Conv}(x^{i-1}, c^1_{in}, c^1_{out}, k_1, d_1)$
$y_1 \leftarrow \text{Dilated\_Conv}(y_1, c^1_{out}, c^2_{out}, k_2, d_2)$
$x^i \leftarrow \text{Residual}(x^{i-1}, y_1, c^1_{in}, c^2_{out}, c_{res})$
$y_o \leftarrow \text{Inv}(x^i, nC, c)$ (For last DCNN block)

(2) For computer vision applications, how much improvement can we get?

(3) When using our method for scientific analysis of neuroimaging data, can we obtain promising results that show that such models can enable discoveries beyond current capabilities?

Next, we will answer the questions above by analyzing the comparative performance of manifold DCNN via four experiments:

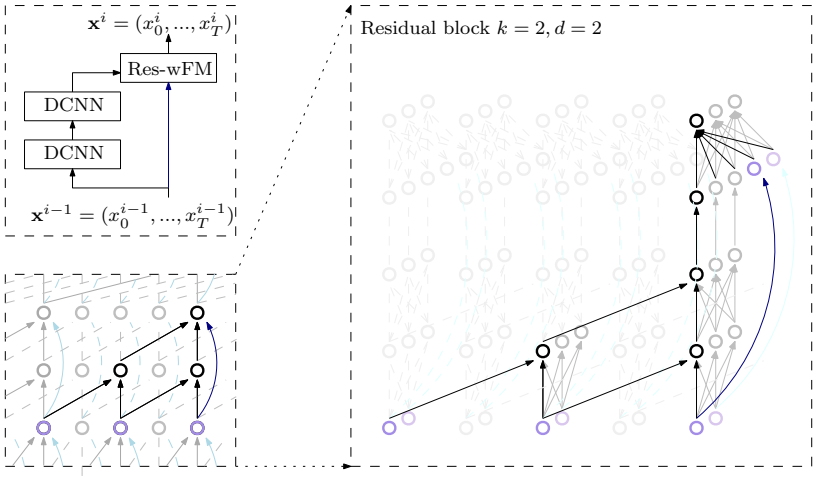

Figure 3.3: Schematic diagram of the residual block of manifold DCNN. There are two DCNN blocks and one residual connection in one block. wFM is used to extract the $c_{out} = 3$ channels from the concatenation.

(**1**) two computer vision applications of classifying videos and

(**2**) two neuroimaging experiments for scientific discoveries related to Alzheimer's disease.

Improvement in Terms of Parameters/Time on Synthetic and Real Computer Vision Datasets

In this section, we organize two sets of experiments:

(**1**) Classification of different moving patterns on the Moving MNIST data

(**2**) Classification of 11 actions on the UCF-11 data.

Both these experiments serve as empirical evidence of the efficiency of manifold DCNN in terms of the number of parameters and time per epoch. We compared our method with five state-of-the-art sequential models: SPD-SRU Chakraborty et al. (2018c), LSTM Hochreiter and Schmidhuber (1997), SRU Oliva et al. (2017), TT-GRU and TT-LSTM Yang et al. (2017). For all methods except TT-GRU and TT-LSTM, before the sequence process module, we used a convolution block. For manifold DCNN and SPD-SRU (also for manifold-valued data), between the convolution block and the sequence process unit, we include a covariance block analogous to Yu and Salzmann (2017). The architecture of this experiment is shown in the Figure 3.4.

As one of the key operations of DCNN is wFM, below we will use an efficient recursive provably consistent estimator of wFM on the space of covariance matrices (SPD with some added small noise along diagonal). Let $X(s)$ be an SPD matrix for all $s \in \mathbf{N}$, and then the n^{th} recursive wFM estimator, M_n is given as:

$$M_0 = X(s) \qquad M_n = \Gamma_{M_{n-1}}^{X(s-n*d)} \left(\frac{w(n)}{\sum_{j=0}^{n} w(j)} \right), \tag{3.5}$$

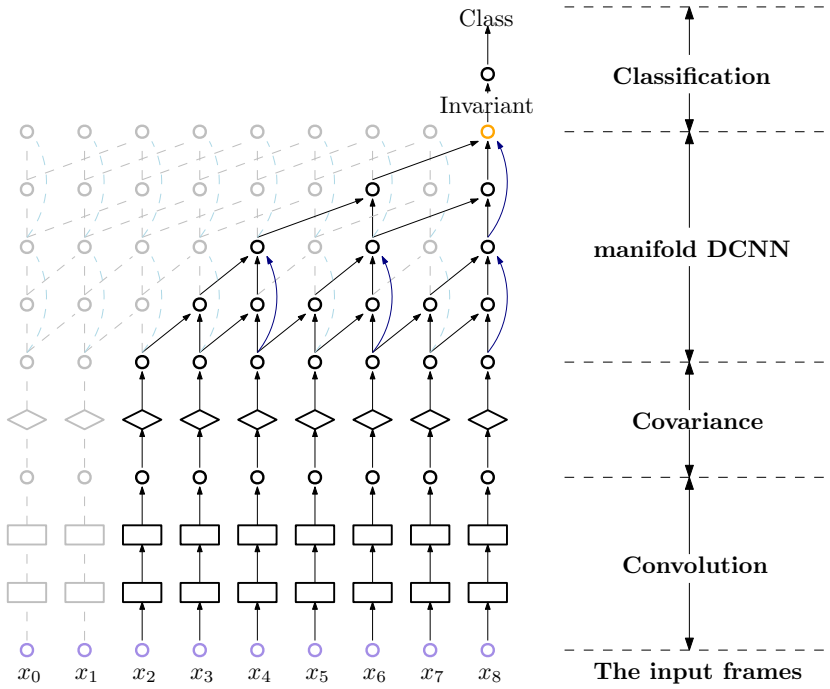

Figure 3.4: Schematic diagram of the network architecture for vision datasets. We use two CNNs to extract the features. And we calculate the covariance between feature channels to get the SPD matrices. In the last layer, we use a G-invariant and a fully connected layer to do the classification.

where Γ is the shortest geodesic on the manifold of SPD matrices equipped with the canonical affine invariant Riemannian metric Ho et al. (2013).

Moving MNIST: Moving pattern classification

We generated the Moving MNIST data according to the algorithm proposed in Srivastava et al. (2015). In this experiment, we classify the moving patterns of different digits. For each moving pattern, we generated 1000 sequences with length 20 showing 2 digits moving in the same pattern in a 64×64 frame. The moving speed and the direction are fixed inside each class, but the digits are chosen randomly. In this experiment, the difference in the moving angle from two sequences across different classes is at least 5°.

Results: In the Table 3.1, the results show that our method not only achieves the best test accuracy with the smallest number of parameters but is also 1.5 times faster than the SPD-SRU which has the second smallest # of parameters. The kernel of CNN we use has size 5×5 with the input channel and output channel set to 5 and 10 respectively. All parameters are chosen in a way to use the fewest number of parameters without deteriorating the test accuracy.

Scalability. We assess the running time (training and testing) of manifold DCNN with respect to the SPD matrix size. From Figure 3.5(a), we can see that as the matrix size increases, the training time increases, while the testing time remains almost the same. This is a desirable property as it indicates that inference time does not depend on matrix size. Also, for different orientations differences, manifold DCNN gives almost perfect classification accuracy with very small standard deviation, as shown in the Figure 3.5(b).

UCF-11: Action classification

The UCF-11 dataset Liu et al. (2009) contains 1600 video clips of 11 different classes, such as basketball shooting, diving, etc. The video lengths (frame sequences) vary from 204 to 1492, with the resolution of each frame being 320×240. We sample every 3 frames, resize each frame to 160×120, and clip the frame sequences to have the length of 50. For our method, we chose two convolution layers with kernels 7×7 and output channels 4 and 6 before the DCNN block. Hence, the dimension of the covariance matrices is 7×7. For the manifold DCNN block, we use three

Table 3.1: Comparative results on Moving MNIST. Our model achieves the highest accuracy (in blue) with the least # of parameters in all setups.

Model	**# params.**	**time (s) / epoch**	**Test acc.**		
			30°versus 60°	10°versus 15°	10°versus 15°versus 20°
DCNN	**1517**	~ 4.3	**1.00 ± 0.00**	**1.00 ± 0.01**	**0.95 ± 0.01**
SPD-SRU	1559	~ 6.2	**1.00 ± 0.00**	0.96 ± 0.02	0.94 ± 0.02
TT-GRU	2240	~ **2.0**	**1.00 ± 0.00**	0.52 ± 0.04	0.47 ± 0.03
TT-LSTM	2304	~ **2.0**	**1.00 ± 0.00**	0.51 ± 0.04	0.37 ± 0.02
SRU	159862	~ 3.5	**1.00 ± 0.00**	0.75 ± 0.19	0.73 ± 0.14
LSTM	252342	~ 4.5	0.97 ± 0.01	0.71 ± 0.07	0.57 ± 0.13

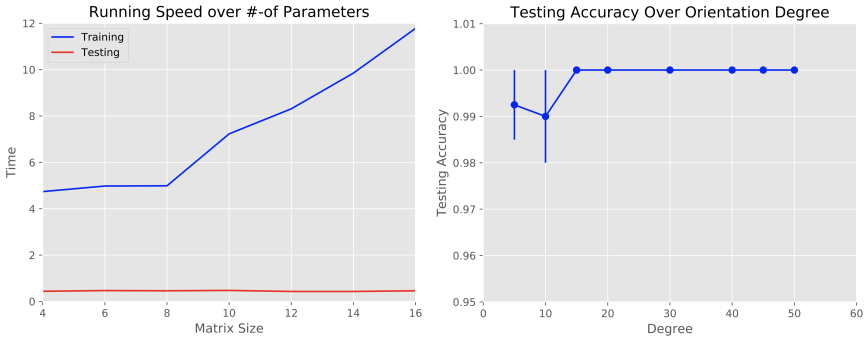

Figure 3.5: *Left:* time versus matrix size. As the matrix size increases, the training time inevitably increases but the testing time consistently remains extremely small. *Right:* accuracy versus degree difference of orientation in the dataset. Beyond the degree difference as small as 15°, the error bar becomes negligible implying our model quickly becomes very robust.

residual blocks, with channels set to be $[1, 3, 3]$; $[3, 3, 4]$ and $[4, 4, 4]$ respectively. The kernel size is 5 for each residual block with the initial dilation number being 1 (if not specified, the initial dilated number is always 1 in this chapter.). For TT-GRU and TT-LSTM, we follow the same setting as given in Yang et al. (2017). For SPD-SRU, SRU, and LSTM, we use the same parameters as in Chakraborty et al. (2018c). All models achieve $> 90\%$ training accuracy.

Results. Test accuracy with the number of parameters and time per epoch is shown in Table 3.2. We can see the number of parameters for our method is comparable with SPD-SRU with higher test accuracy ($\approx 4\%$ improvement) and much faster runtime ($\approx 2.5\times$). Note that without residual connections, the accuracy drops to 0.809 ± 0.044: in other words, residual connections are useful.

Take-home message. *With the above two experiments, we can conclude that manifold DCNN* $(\boldsymbol{a})$ *is faster,* $(\boldsymbol{b})$ *uses fewer parameters and* $(\boldsymbol{c})$ *gives better or comparable classification accuracy compared to the state-of-the-art.*

Group Effects in Preclinical Alzheimer's Disease

Cardinal features of Alzheimer's disease (AD) include the development of beta-amyloid plaques (amyloid), neurofibrillary tangles (tau), and progressive neurodegeneration (characterized by MRI) Jack et al. (2018). Autopsy studies among individuals with AD dementia indicate that degeneration of myelinated axons in the context of amyloid and tau pathology is a defining feature of dementia status Perez-Nievas et al. (2013). Techniques for measuring axonal degeneration in vivo include analysis of cerebrospinal fluid, as well as diffusion-weighted imaging; however, few studies have tested the extent to which early amyloid accumulation may be associated with neural injury. Our goal is to utilize our method to **identify white matter fiber bundles that are affected *early* in the preclinical disease process**. Positron emission tomography (PET) imaging with Pittsburgh compound B (PiB), which identifies amyloid deposition, can be used as an indicator of AD pathology Ikonomovic et al. (2008). Thus, we compared healthy individuals who were positive for AD pathology (PiB+) to healthy individuals who were negative for pathology (PiB-). Additionally, we compared individuals who carried a risk gene for AD (APOE+) to non-carriers (APOE-).

Diffusion-weighted imaging (DWI)

Data acquisition. Diffusion-weighted imaging was completed on a General Electric (GE) 3 Tesla scanner with a 32-channel head coil and a spin-echo echo-planar imaging pulse sequence among participants who are asymptomatic. Multi-shell

Table 3.2: Comparative results on UCF-11 data. Our model achieves the best accuracy and the fastest speed with a small number of parameters.

Model	# params.	time (s)/ epoch	Test acc.
manifold DCNN	3393	~ **33**	**0.823 ± 0.018**
SPD-SRU	**3337**	~ 76	0.784 ± 0.014
TT-GRU	6048	~ 42	0.78
TT-LSTM	6176	~ **33**	0.78
SRU	2535630	~ 50	0.75
LSTM	14626425	~ 57	0.70

DWI data were collected using b-values $b = 0,\ b = 500,\ b = 800,\ b = 2000$, with $2 \times 2 \times 2mm$ resolution. The signal was corrected using MRTrix3 Veraart et al. (2016) and FSL's 'eddy' Andersson and Sotiropoulos (2016). Diffusion tensor imaging (DTI) and the orientation distribution functions (ODF), which were used as the representative of the DWI, were performed using the Diffusion Imaging in Python (DIPY) toolbox Garyfallidis et al. (2014). To generate fiber bundles of interest, the data was processed using TRACULA Yendiki et al. (2011, 2014, 2016). With this pipeline, we generated 18 major fiber bundles Wakana et al. (2007), as shown in the Figure 3.2(c). Regions of interest (ROI) in the template space, were inversely warped back to the subject space to generate the fiber bundles and each data point used in the analysis for each participant.

Analysis. From the previous experiments, we can see that manifold DCNN performs well on classification problems with faster computation speed and fewer parameters. Due to the fast runtime *and* the small number of parameters, we can use permutation testing to perform group analysis. The statistical testing is performed on each fiber bundle between the two groups, to determine if the DCNN model between the two groups is different. To summarize, the setup is:

(1) Group 1 (PiB+) versus Group 2 (PiB-),

(2) Group 1 (APOE+) versus Group 2 (APOE-).

Now, we will give some details of the DCNN models for DTI and ODF representations before the statistical analysis.

(i) Diffusion tensor imaging (DTI). Since all of the data samples lie on the SPD manifold, the model is similar to the classification model above. The only difference between classification model and this group analysis model is that instead of the prediction of the classes, we are fitting the two groups of data into two trainable models, θ_1 and θ_2 and assessing if the distributions of θ_1 and θ_2 are statistically different.

(ii) Orientation distribution function (ODF). Orientation distribution function (ODF) represents the probability densities of water diffusion over different orientations. In order to perform the statistical analysis, we discretized the space of

orientations, i.e., $\mathbf{S}^2$. We sampled 724 equally spaced points on the sphere $\mathbf{S}^2$ to represent the ODF. Let the ODF be denoted by $\mathbf{x}_t$, then after the discretization, we have $\sum_{i=1}^{724} \mathbf{x}_t^i = 1$. As ODF is a probability density function, we use square root parameterization Brody and Hughston (1998); Srivastava et al. (2007) to represent ODF. Using the square root parameterization, we map $\mathbf{x}_t$ onto the positive orthant of the unit hypersphere of dimension 723, i.e., $\mathbf{S}^{723}$. As in Section 3.3, a key component of DCNN is the definition of wFM, which we can define on $\mathbf{S}^{n-1}$:

$$\begin{aligned}\mathbf{y}(s) &= \mathsf{wFM}\left(\{w(i)\}, \{\mathbf{x}(s - d * (k-1) : d : s)\}\right) \\ &= \arg\min_{M} \sum_{i=0}^{k-1} w(i)\, d_S^2\left(\mathbf{x}(s - d * i), M\right),\end{aligned} \tag{3.6}$$

Here d_S is the rotation invariant geodesic distance on $\mathbf{S}^{723}$ and $\mathbf{x}(s)$ is a sample on $\mathbf{S}^{723}$ for $s \in \mathbf{N}$. Analogous to the SPD manifold, we can define a recursive wFM estimator $\mathbf{m}_n$:

$$\mathbf{m}_0 = \mathbf{x}(s) \qquad \mathbf{m}_n = \Gamma_{\mathbf{m}_{n-1}}^{\mathbf{x}(s-n*d)}\left(\frac{w(n)}{\sum_{j=0}^{n} w(j)}\right), \tag{3.7}$$

where Γ is the shortest geodesic on $\mathbf{S}^{723}$. Using the above-defined estimator of wFM, we can define DCNN on $\mathbf{S}^{723}$ as in Section 3.3.

Note. Our baseline model, SPD-SRU cannot deal with the S^n manifold as we do here.

Statistical analysis: permutation testing

As briefly discussed in Section 2.2, we review how permutation testing will be used here. Suppose we train our model for each of the two groups for each fiber bundle $\mathtt{fb}$ we provided, with parameters $\theta_1^{\mathtt{fb}}$ and $\theta_2^{\mathtt{fb}}$. Our goal is to test whether the fiber bundle $\mathtt{fb}$ is statistically different between the two groups. Thus, we model the manifold-valued data and perform statistical analysis in the parameters space. Since

Table 3.3: Description of data/participant demographics used in the study.

Experiments		PiB			APOE	
	Total	Positive	Negative	Total	Positive	Negative
Number	196	29	167	669	247	422
Age (years) (mean (SD))	62.40 (6.33)	66.29 (4.95)	61.75 (6.30)	65.61 (8.68)	64.55 (7.99)	66.23 (9.00)
Sex (female; %)	134 (68%)	21 (72%)	113 (68%)	426 (64%)	159 (64%)	267 (63%)

the models for each group lie in the same parameter space, the statistical analysis can be performed in the parameter space by bootstrapping. We can measure the distance between two models as $\sigma^{fb} = ||\theta_1^{fb} - \theta_2^{fb}||$ to represent the distance between the group-wise fitted models' distributions in parameter space. Then, we need to evaluate how statistically significant the distance is – and if the value is large enough, it is unlikely to happen by chance. A simple way to perform the test for statistical significance is via permutation testing. If we randomly shuffle (via a random permutation) the group information for all our samples (i.e., subjects) and run our model for both "random" groups, we will get new parameters $\hat{\theta}_1^{fb}$ and $\hat{\theta}_2^{fb}$. We define $\hat{\sigma}^{fb} = ||\hat{\theta}_1^{fb} - \hat{\theta}_2^{fb}||$ as a random variable. After permuting 5000 times, we can estimate the distribution of the $\hat{\sigma}^{fb}$ – this is the Null distribution (See Figure 3.6 as examples). The p-value is defined as the ranking of the σ^{fb} within the empirical distribution of $\hat{\sigma}^{fb}$. If the p-value is less than the significance threshold $\alpha = 0.05$, we can conclude that this is **not** likely to happen by chance.

Since the length of different fiber bundles varies from 11 to 73, we construct a DCNN model with 3 layers of residual units, with channels being $1, 3, 3$; $3, 3, 5$ and $5, 8, 10$ respectively. The 1-D kernel size is 3. We use all the data we have to pre-train the model. After pre-training, we fine tune the model during the permutation testing.

Result 1: Group analysis: PiB+ versus PiB-

The study included imaging data acquired from 196 cognitively unimpaired (healthy) participants acquired in a local cohort at the University of Wisconsin. We provide demographic information from participants with PiB and APOE measures in Table 3.3. Initial analyses were performed using single-shell data, where the model

was run on all 18 fiber bundles, one by one, with the parameters mentioned above. We performed permutation tests for each fiber bundle individually.

Results for the 18 fibers are shown in Table 3.4 (column 2). We find that two of the 18 fibers satisfied the threshold of 0.05, which means that statistically these fiber bundles are different across the two groups. Since the sample sizes were small, the results presented are uncorrected p-values (multiple testing correction was not performed).

Fiber bundles evaluated in this analysis included those which are known **to be affected** in AD, including the superior longitudinal fasciculus and cingulum bundle, as well as control tracts that are **not likely to be affected** by AD, such as the corticospinal tract. We found significant differences between PiB+ and PiB- groups in fiber bundles that are likely to be affected by AD, including the superior longitudinal fasciculus and Corpus callosum - forceps minor.

When compared with the SPD-SRU model, which also reported brain imaging experiments, the results show only one out of 18 fibers survives. Also, we find that our model runs **much faster** (about $5\times$), which is very important when running permutation testing thousands of times. It takes 3.5 days to run permutation testing 5000 times using DCNN, while the SPD-SRU takes 18 days. When we keep the number of GPUs fixed, the difference between 3.5 and 18 will be even more sizable if we expand the number of permutation testing to 10000 or more.

Result 2: Group analysis: APOE+ versus APOE-

The APOE analysis was performed using data from 669 subjects with APOE information, with 247 of them being positive for APOE4 (a risk factor for AD). Analyses were also conducted using the multi-shell dMRI to generate ODF information. Similar to the preceding group difference analysis, the model was run on all 18 fiber bundles with the parameters described previously on both DTI and ODF.

The results for 18 fibers are shown in Table 3.4 in column 3. It is noteworthy that SPD-SRU can only deal with the SPD manifold. So for ODF, which lies on S^n, we can *only* run our DCNN model to do the group analysis.

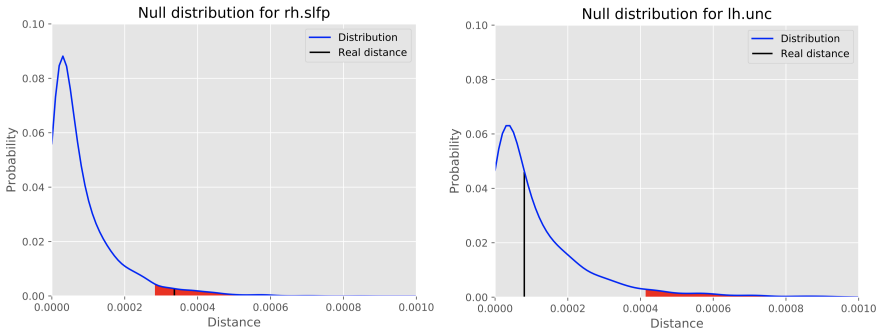

Figure 3.6: The Null distribution for one fiber bundle with $\alpha = 0.05$. If the real distance (black line) lies in the threshold (red area), that test is believed to not happen by chance.

Here, we found that four of the 18 fiber bundles met the significance threshold of 0.05 with DTI, while SPD-SRU only captured one. Five fiber bundles were identified when using ODF. We found differences by APOE genotype in the forceps minor, cingulum projecting to parietal cortex, anterior thalamic projections, superior longitudinal fasciculus projecting to parietal cortex and inferior longitudinal fasciculus. We did not find differences in fiber bundles unlikely to be affected by AD, such as the corticospinal tract in both experiments. Fiber bundles that were consistently identified in both the DTI and ODF analyses included the inferior longitudinal fasciculus and the anterior thalamic projections.

Discussion of preclinical AD analysis results

While amyloid and tau pathology are defining features of AD, methods are also needed to detect AD-associated neurodegeneration Jack et al. (2018). Neurodegeneration may signal future cognitive decline. However, methods for detecting early and subtle neurodegeneration, particularly of myelinated axons, are not yet available, especially in preclinical AD. This is why our results here seem promising.

The results suggest significant differences in underlying fiber bundle microstructure among individuals who meet biological criteria for AD (based on PiB status) as well as differences by APOE genotype. Of note, our algorithm identified significant

differences in the cingulum bundle by PiB status; this white matter fiber bundle connects medial temporal lobe and parietal cortices as part of a memory network that is impacted by AD, and is vulnerable to degeneration in the early stages of AD. Differences in the cingulum bundle were also apparent among carriers of the APOE4 allele, a genetic risk factor for sporadic AD. Likewise, superior longitudinal fasciculus differed by AD biomarker status and APOE genotype. Projections identified as being significantly different included fiber bundles projecting to parietal cortices. Parietal cortices are significantly impacted by AD pathology and are among the first to show amyloid accumulation. The results presented here may suggest that amyloid accumulation negatively impacts adjacent white matter fiber bundles. It may also be possible that degeneration of fiber bundles is a function of AD pathology spreading to anatomically linked brain regions via white matter fiber bundles, although further longitudinal evaluation is needed to test the hypothesis. In summary, statistical analysis enabled by our proposed algorithm was capable of identifying differences in biologically meaningful brain regions.

Take-home message. *Our DCNN model was able to capture more fiber differences with significant effects compared to the SPD-SRU. It is also noteworthy that our model is much more efficient: only* 60*s for one realization of the permutation test* ($\times$*# of realizations*)*, while the SPD-SRU model* $> 5\times$ *times slower. Compared with the SPD-SRU, which can only handle DTI* (SPD)*, our method is more general: handles both DTI* (SPD) *and ODF* ($\mathbf{S}^n$) *data.*

3.5 Conclusions

We present a new Dilated CNN formulation to model sequential and spatio-temporal manifold data, where few alternatives are available at that time. Compared with the standard sequential model (RNN), our method can improve the performance when evaluated on the number of parameters and runtime. We show that when using wFM, Weight normalization, ReLU, and Dropout are no longer needed in this formulation. On the experimental side, for video analysis, we show that improvements can be obtained with fewer parameters and shorter running time. Importantly, we

Table 3.4: p-values (uncorrected) for all fibers in different groups. The highlights are the fiber bundles that satisfy the significance threshold. Runtime for DCNN is 5× times faster than SPD-SRU (not included here).

Fiber Name	p-value				
	Experiment 1 PiB+ versus PiB-		**Experiment 2** APOE+ versus APOE-		
	DCNN on DTI	SPD-SRU	DCNN on DTI	SPD-SRU	DCNN on ODF
fmajor_PP	0.443	0.923	0.207	0.600	0.778
fminor_PP	**0.008**	0.158	**0.035**	**0.025**	N/A
lh.atr_PP	0.323	0.632	0.30	0.991	**0.028**
rh.atr_PP	0.295	0.143	0.86	0.271	0.563
lh.cab_PP	0.276	0.363	0.76	0.644	0.500
rh.cab_PP	0.311	0.263	0.78	0.848	0.444
lh.ccg_PP	0.230	0.267	**0.042**	0.609	**0.043**
rh.ccg_PP	0.093	0.087	**0.048**	0.532	**0.048**
lh.cst_AS	0.561	0.143	0.58	0.350	0.800
rh.cst_AS	0.629	0.278	0.35	0.667	0.769
lh.ilf_AS	0.309	0.895	0.47	0.977	**0.042**
rh.ilf_AS	0.405	0.889	0.46	0.563	0.857
lh.slfp_PP	0.482	0.615	0.68	0.107	0.192
rh.slfp_PP	0.571	0.941	**0.047**	0.154	**0.050**
lh.slft_PP	**0.005**	**0.041**	0.92	0.649	0.556
rh.slft_PP	0.790	0.462	0.53	0.947	0.333
lh.unc_AS	0.623	0.158	0.23	0.860	0.933
rh.unc_AS	0.298	0.895	0.34	0.324	0.182

* N/A: This ODF fiber bundle did not pass Quality Check (QC) after pre-processing. Therefore, we left it out of the analysis to avoid inconsistencies in the parameters used for pre-processing the full set of fiber bundles.

show that our algorithmic contributions facilitate scientific discovery relevant to AD, and may facilitate early disease detection at the preclinical stage. The analysis enabled by our formulation revealed subtle neurodegeneration of white matter fiber bundles affected by AD pathology, in brain regions implicated in prior studies of AD. We applied our proposed DCNN model to another dataset from the Dominantly Inherited Alzheimer Network at Washington University at St. Louis that

focuses on Dominantly Inherited Alzheimer's Disease (DIAD). In this dataset, participants are grouped into specific gene mutation carriers who will develop DIAD, and non-carriers. We extracted 50 major fiber bundles using TractSeg Wasserthal et al. (2018). To make a meaningful comparison, we also re-ran our method on the dataset discussed in this chapter using the TractSeg method. We identified 14 fibers within the 50 fibers that differed by amyloid status and 16 fibers that differed by mutation status. Across the two datasets, 9 fibers (including arcuate fasciculus and cingulum) were found to be altered in both sporadic AD and DIAD. Additional longitudinal studies are needed to determine the temporal relationship between the accumulation of amyloid and neurodegeneration in the development of dementia. The code for the algorithms described in this chapter is available at `https://github.com/zhenxingjian/DCNN`.

4 FLOW-BASED GENERATIVE MODELS FOR LEARNING MANIFOLD TO MANIFOLD MAPPINGS

In Chapter 3, we discussed how to extend a DCNN model to SPD matrices and distribution functions. There are also many other proposals (like spherical CNN) that have extended a number of deep neural network architectures to manifold-valued data, and this has often provided strong improvements in performance. However, the literature on generative models for manifold data is quite sparse. Partly due to this gap, there are also relatively few modality transfer/translation models for manifold-valued data whereas numerous such methods based on generative models are available for natural images. This chapter addresses this gap, motivated by a need in brain imaging (similar as Chapter 3) – in doing so, we expand the operating range of certain generative models (as well as generative models for modality transfer) from natural images to images with manifold-valued measurements. Our main result is the design of a two-stream version of GLOW (flow-based invertible generative models) that can synthesize information of a field of one type of manifold-valued measurements given another. On the theoretical side, we introduce three kinds of invertible layers for manifold-valued data in Section 4.3, which are not only analogous to their functionality in flow-based generative models (e.g., GLOW) but also preserve the key benefits (determinants of the Jacobian are easy to calculate). For experiments, in Section 4.4, on a large dataset from the Human Connectome Project (HCP), we show promising results where we can reliably and accurately reconstruct brain images of a field of orientation distribution functions (ODF) from diffusion tensor images (DTI), where the latter has a $5\times$ faster acquisition time but at the expense of worse angular resolution. The work presented in this chapter appeared as a conference paper in AAAI 2021 Zhen et al. (2021b).

4.1 Introduction

The results of most methods in the literature dealing with differential geometry in deep learning, focus on harnessing the power of DNNs for better analysis of manifold or structured data, and the results are impressive. But most approaches are discriminative in nature. In other words, the goal is to characterize the conditional distribution $P(Y|\phi(X))$ based on the predictor variables or features X, here X is manifold-valued and the responses or labels Y are Euclidean. The technical thrust is on the design of mechanisms to specify $\phi(\cdot)$ so that it respects the geometry of the data space. In contrast, work on the generative side is very sparse, and to our knowledge, only a couple of methods for a few specific manifolds have been proposed thus far Brehmer and Cranmer (2020); Rey et al. (2020); Miolane and Holmes (2020); Huang et al. (2019). As a result, the numerous application settings where generative models have shown tremendous promise, namely, semi-supervised learning, data augmentation Antoniou et al. (2017); Radford et al. (2016) and synthesis of new image samples by modifying a latent variable Kingma and Dhariwal (2018); Sun et al. (2019) as well as numerous others, currently cannot be easily evaluated for domains with data-types that are not Euclidean.

GANs for Manifold data: what is challenging? There are some reasons why generative models have been applied to manifold data sparingly. A practical consideration is that many application areas where manifold data are common, such as shape analysis and medical imaging, we cannot often provide the sample sizes needed to train off-the-shelf generative models such as Generative adversarial networks (GANs) Goodfellow et al. (2014) and Variational auto-encoders (VAEs) Kingma and Welling (2014); Doersch (2016). There are also several issues on the technical side. Consider the case where a data sample corresponds to an image where each pixel is a manifold variable (such as a covariance matrix). This means that each sample lives on a product space of the manifold of covariance matrices. Attempting to leverage state of the art methods for GANs such as Wasserstein GANs (WGANs) Arjovsky et al. (2017) will involve, as a first step, defining appropriate generators that take uniformly distributed samples on a product space of manifolds

and transforming it into "realistic" samples which are also samples on a product space of manifolds. In principle, this can be attempted via recent developments by extending spherical CNNs or other architectures for manifold data Chakraborty et al. (2018a). Next, one would not only need to define optimal transport Fathi and Figalli (2010) or Wasserstein distances Huang et al. (2019) in complicated spaces, but also develop new algorithms to approximate such distances (e.g., Sinkhorn iterations) to make the overall procedure computationally feasible. An interesting attempt to do so was described in Huang et al. (2019). In that paper, Huang et al. introduced a WGAN-based generative model that can generate low-resolution low-dimension manifold-valued images. On the other hand, VAEs are mathematically more convenient in comparison for such data, and as a result, a few recent works show how they can be used for dealing with manifold-valued data Miolane and Holmes (2020). Other VAE style approaches Huang et al. (2022) that have appeared after the initial publication of this work may also be extended to manifolds but this has not been attempted so far. While these methods inherit VAE's advantages such as ease of synthesis, VAEs are known to suffer from optimization challenges as well as a tendency to generate smoothed samples. In general, it is not clear how the numerical issues, in particular, will be amplified once we move to manifold data where the core operations of calculating geodesics and distances, evaluating derivatives, and so on, must also invoke numerical optimization routines.

Contributions. Instead of GANs or VAEs, the use of flow-based generative models Rezende and Mohamed (2015); Kingma and Dhariwal (2018), will enable latent variable inference and log-likelihood evaluation. It turns out, as we will show in our development shortly, that the key components (and layers) needed in flow-based generative models with certain mathematical/procedural adjustments, extends nicely to the manifold setting. The goal of this work is to describe our theoretical developments and show promising experiments in brain imaging applications involving manifold-valued data.

4.2 Preliminaries

This subsection briefly summarizes some differential geometric concepts/notations we will use in this chapter. Readers will find a more comprehensive treatment in Chapter 2 and in Boothby (1986).

Definition 4.1. (*We will use Definitions 2.13 and 2.15 from Chapter 2 here. We re-state such definitions as a quick reference for readers.*) *Let* $(\mathcal{M}, g)$ *be an orientable complete Riemannian manifold with a Riemannian metric* g*, i.e.,* $\forall p \in \mathcal{M} : g_p : T_p\mathcal{M} \times T_p\mathcal{M} \to \mathbb{R}$ *is a bi-linear symmetric positive definite map, where* $T_p\mathcal{M}$ *is the tangent space of* $\mathcal{M}$ *at* $p \in \mathcal{M}$*. Let* $d : \mathcal{M} \times \mathcal{M} \to [0, \infty)$ *be the distance induced from the Riemannian metric* g*.*

Definition 4.2. *Let* $p \in \mathcal{M}$, $r > 0$*. Define* $\mathcal{B}_r(p) = \{q \in \mathcal{M} | d(p, q) < r\}$ *to be an open ball at* p *of radius* r*.*

Definition 4.3 (Local injectivity radius Groisser (2004))**.** *The local injectivity radius is defined as* $r_{inj}(p) = \sup \left\{ r | Exp_p : (\mathcal{B}_r(\mathbf{0}) \subset T_p\mathcal{M}) \to \mathcal{M} \right\}$ *where* Exp_p *is defined and is a diffeomorphism onto its image at* $p \in \mathcal{M}$*. The injectivity radius Manton (2004) of* $\mathcal{M}$ *is defined as* $r_{inj}(\mathcal{M}) = \inf_{p \in \mathcal{M}} \left\{ r_{inj}(p) \right\}$.

Within $\mathcal{B}_r(p)$, where $r \leqslant r_{inj}(\mathcal{M})$, the mapping $Exp_p^{-1} : \mathcal{B}_r(p) \to \mathcal{U} \subset T_p\mathcal{M} \subset \mathbb{R}^m$, is called the inverse Exponential/Log map, m is the dimension of $\mathcal{M}$. For each point $p \in \mathcal{M}$, there exists an open ball $\mathcal{U} = \mathcal{B}_r(q)$ for some $q \in \mathcal{M}$ such that $p \in \mathcal{U}$, where $r = r_{inj}(\mathcal{M})$. Thus, we can cover $\mathcal{M}$ by an indexed (possibly infinite) cover

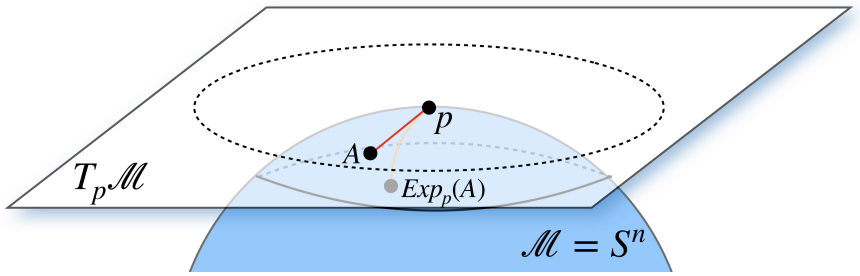

Figure 4.1: Schematic description of an exemplar manifold ($\mathbf{S}^n$) and the corresponding tangent space at a "pole".

$\left\{\left(\mathcal{B}_r(q), \text{Exp}_q^{-1}\right)\right\}_{q\in\mathcal{I}}$. This set is an example of a *chart* on $\mathcal{M}$; for an example, see Krauskopf et al. (2007) and also Figure 4.1.

For notational simplicity, we will denote a chart covering $p \in \mathcal{M}$ by Φ_p, since in general, we can use an arbitrary chart instead of an inverse Exponential map. Note that the domain for two chart maps may not necessarily be disjoint.

Given a differentiable function $F : \mathcal{M} \to \mathcal{M}$ defined as $x \mapsto \Psi^{-1}\left(\widetilde{F}(\Phi(x))\right)$, where Φ and Ψ are the functions in the chart covering x and $F(x)$ respectively and for some differentiable $\widetilde{F} : \mathbb{R}^m \to \mathbb{R}^m$, the Jacobian of F (denoted by $\frac{dy}{dx}$) is defined as:

$$\frac{dy}{dx} := \frac{\partial\Psi \circ \Phi^{-1}}{\partial\Phi(x)} \frac{d\widetilde{F}}{d\widetilde{x}}\Big|_{\Phi(x)} \tag{4.1}$$

The reason for the peculiar notation is that the derivative cannot be defined on manifold-valued data, so $\frac{dy}{dx}$ is not meaningful: we use the notation $\dot{\approx}$ to acknowledge this difference. Also note that Ψ, Φ are the same only when

(1) using the global charts for space X and $F(X)$

(2) X and $F(X)$ are on the same manifold.

Definition 4.4 (**Group of isometries of** $\mathcal{M}$ $(I(\mathcal{M}))$**.** *A diffeomorphism* $\iota : \mathcal{M} \to \mathcal{M}$ *is an isometry if it preserves distance, i.e.,* $d(\iota(x), \iota(y)) = d(x, y)$. *The set* $I(\mathcal{M})$ *forms a group with respect to function composition.*

Rather than writing an isometry as a function ι, we will write it as a group action. Henceforth, let G denote the group $I(\mathcal{M})$, and for $g \in G$, $x \in \mathcal{M}$, let $g \cdot x$ denote the result of applying the isometry g to point x. Similar to the terminologies in Chakraborty et al. (2018c), we will use the term "translation" to denote the group action ι. This is due to the distance preserving property and is inspired by the analogy from the Euclidean space.

4.3 Flow-based Generative Models

In Section 2.3, we briefly introduced the flow-based generative models for Euclidean data. Here in this section, we will introduce flow-based generative models for manifold-valued data in detail. To start with, we will recall the Euclidean formulation and specify which components need to be generalized to get the manifold-valued formulation.

Flow-based Models: Euclidean Case

Recall that flow-based generative models Rezende and Mohamed (2015); Kingma and Dhariwal (2018); Yang et al. (2019) aim to maximize the log-likelihood of the input data from an unknown distribution. The idea involves mapping the *unknown distribution in the input space* to a *known distribution in the latent space* using an invertible function, f. At a high level, sampling from a known distribution is simpler, so an invertible f can help draw samples from the input space distribution.

Let $\{\mathbf{x}_i\}$ be i.i.d. samples drawn from an unknown distribution $p^*(\mathbf{x})$. Let this unknown distribution be parameterized by θ. In the rest of this chapter, we use $p_\theta(\mathbf{x})$ as a proxy for $p^*(\mathbf{x})$. We learn θ over a dataset $\mathcal{D}$. We maximize the likelihood of the model θ given the dataset $\mathcal{D}$ by minimizing the equivalent formulation of negative log-likelihood as:

$$\ell(\theta|\mathcal{D}) = \frac{1}{N}\sum_{i=1}^{N} -\log p_\theta(\mathbf{x}_i) \tag{4.2}$$

But to minimize the above expression, we need to know p_θ. One way to bypass this problem is to learn a mapping from a known distribution in the latent space. Let the latent space be $\mathbf{z}$. Then, the generative step is given by $\mathbf{z} \sim p(\mathbf{z})$, $\mathbf{x} = g(\mathbf{z})$. Here $p(\mathbf{z})$ can be a Gaussian distribution $\mathcal{N}(\mathbf{z}; \mathbf{0}, I)$.

Let f be the inverse of g. For normalizing flow Rezende and Mohamed (2015), f is composed as a sequence of invertible functions $f = f_1 \circ f_2 \circ \ldots \circ f_K$. Hence, we

have

$$\mathbf{x} \overset{f_1}{\longleftrightarrow} \mathbf{h}_1 \overset{f_2}{\longleftrightarrow} \mathbf{h}_2 \dots \overset{f_K}{\longleftrightarrow} \mathbf{z}$$

where $\mathbf{h}_t$ is the hidden representation at layer t. Using $\mathbf{h}_0 = \mathbf{x}$ and $\mathbf{h}_K = \mathbf{z}$, the log-likelihood of $p_\theta(\mathbf{x})$ is

$$\log p_\theta(\mathbf{x}) = \log p(\mathbf{z}) + \log|\det(d\mathbf{z}/d\mathbf{x})| \tag{4.3}$$

$$= \log p_\theta(\mathbf{z}) + \sum_{j=0}^{K} \log|\det(d\mathbf{h}_j/d\mathbf{h}_{j-1})| \tag{4.4}$$

where $\det(\cdot)$ is determinant of a square matrix $\cdot$.

In Kingma and Dhariwal (2018), the GLOW model is composed of three different layers whose Jacobian $d\mathbf{h}_j/d\mathbf{h}_{j-1}$ is a triangular matrix, simplifying the log-determinant:

$$\log|\det(d\mathbf{h}_j/d\mathbf{h}_{j-1})| = \sum(\log|\mathrm{diag}(d\mathbf{h}_j/d\mathbf{h}_{j-1})|) \tag{4.5}$$

where $\mathrm{diag}(\cdot)$ is the vector of diagnose elements of a matrix $\cdot$.

The three layers in the basic GLOW block (shown in Figure 4.2), summarized in Table 4.1 are all invertible functions. These are **(a)** Actnorm, **(b)** Invertible 1×1 convolution, and **(c)** Affine Coupling layers. Note that the data is squeezed before it is fed into the block. Then, the data is split as in Dinh et al. (2017).

(a) Actnorm. normalizes the input to be a zero-mean and identity standard deviation. In (4.6), μ, σ are initialized from the data and then trained independently.

(b) 1×1 convolution. applies the invertible matrix R on the channel dimension. In (4.7), $X \in \mathbb{R}^{s_r \times c_r}$ and $R \in \mathbb{R}^{c_r \times c_r}$ where s_r is the resolution of the input variables while c_r is the number of channels.

(c) Affine coupling. uses the idea of split+concatenation. In (4.8), the input variable X is *split* along the channel to X_a, X_b, and then Y_a, Y_b are *concatenated* to get the final output Y. Here, S (and T) are real-valued matrices of the same dimension

as X_b for element-wise scaling (and translation).

In Kingma and Dhariwal (2018), authors use a closed form for the inverse of these layers. Notice that calculating the determinant of the Jacobian is simple for all these layers except the affine coupling layer in (4.8) (Table 4.1).

Since $\frac{dY_a}{dX_b} = 0$, $\frac{dY_a}{dX_a} = I$, the Jacobian determinant is $\det(S)$.

$$\det\left(\frac{dY}{dX}\right) = \det\left(\begin{bmatrix} \frac{dY_a}{dX_a} & \frac{dY_a}{dX_b} \\ \frac{dY_b}{dX_a} & \frac{dY_b}{dX_b} \end{bmatrix}\right) = \det\left(\begin{bmatrix} I & 0 \\ \frac{dY_b}{dX_a} & S \end{bmatrix}\right) = \det(S) \tag{4.9}$$

Next steps. With the description above, we can now list the key operational compo-

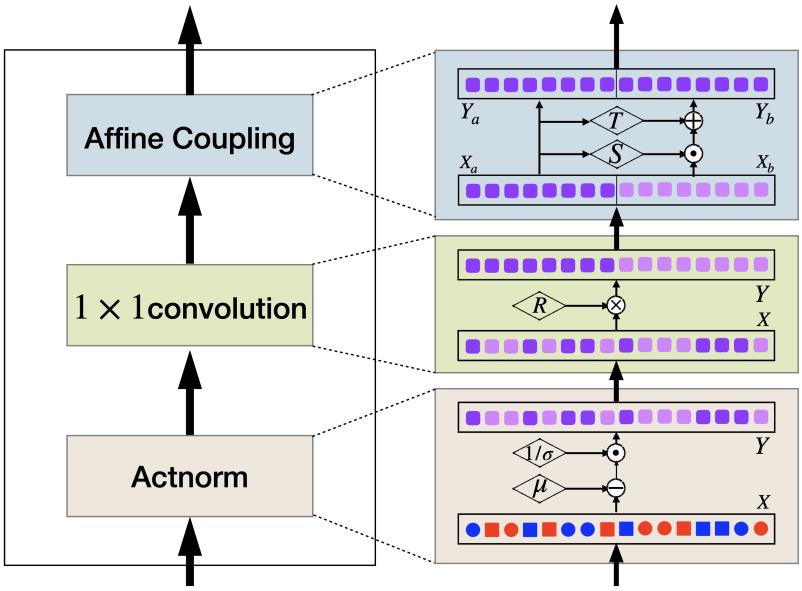

Figure 4.2: The basic block of GLOW Kingma and Dhariwal (2018). The color represents the mean while the shape represents the standard deviation. The target distribution on the latent space is the "Grape" rounded rectangles. *Actnorm* normalizes the data to almost "Grape" rounded rectangles, while the disturbance part is "Lavender". 1×1 *convolution* organizes the channels. *Affine Coupling* operates on half of the channels to fit the target distribution. The "$\cdot$" here is the element-wise multiplication, while "$\times$" is the matrix multiplication. "$+/-$" are of the normal definition.

Table 4.1: Definition of *Actnorm*, 1×1 *convolution* and *Affine Coupling* layers in basic GLOW block. $\odot$ is the elementwise multiplication. The function $\mathsf{NN}()$ is a nonlinear mapping.

Actnorm	1×1 convolution	Affine Coupling
$Y = \frac{1}{\sigma} \odot (X - \mu) \quad (4.6)$	$Y = R \times X \quad (4.7)$	$S, T = \mathsf{NN}(X_a)$ $Y_b = S \odot X_b + T \quad (4.8)$ $Y_a = X_a$

nents in (4.6)-(4.9), which we need to modify for our manifold-valued extension.
Key ingredients. In (4.6) and (4.8), the operators are

(1) elementwise multiplication for σ, S

(2) the addition of bias for μ, T

(3) In (4.7), we require invertible matrices

(4) Finally, to compute the log-likelihood, we need the calculation of derivative in (4.9)

Thus we can verify that the key ingredients to define the model in GLOW are

(1) elementwise multiplication

(2) addition of bias

(3) invertible matrix

(4) derivative calculation.

In theory, if we can modify those components from Euclidean space to manifolds, we will obtain a flow-based generative model on a Riemannian manifold. Observe that **(1)** and **(3)** are matrix multiplications, which are non-trivial to define on a manifold. In Definition 4.3, we can use the chart map to map the manifold to a

Table 4.2: Definition of *Actnorm*, 1×1 *convolution* and *Affine Coupling* layers in our manifold GLOW block, with forward function on the top and reverse function in the bottom. Here Φ and Ψ^{-1} are the Chart Map and its inverse. S is a diagonal matrix, so S^{-1} can be computed elementwise. T^{-1} represents the inverse of the group action. The R is chosen as the rotation matrix. Thus, $R^{-1} = R^{T}$.

Actnorm	1×1 convolution	Affine Coupling
$Y = \Psi^{-1}(S \times \Phi(X)) \cdot T$ (4.10)	$Y = \Psi^{-1}(R \times \Phi(X))$ (4.11)	$S, T = \mathsf{NN}(\Phi(X_a))$ $Y_b = \Psi^{-1}(S \times \Phi(X_b)) \cdot T$ (4.12) $Y_a = X_a$
$X = \Phi^{-1}(S^{-1} \times \Psi(Y \cdot T^{-1}))$ (4.13)	$X = \Phi^{-1}(R^{-1} \times \Psi(Y))$ (4.14)	$S, T = \mathsf{NN}(\Phi(Y_a))$ $X_b = \Phi^{-1}(S^{-1} \times \Psi(Y_b \cdot T^{-1}))$ (4.15) $X_a = Y_a$

subspace of $\mathbb{R}^m$ where a matrix multiplication can be used. This also provides a way to solve item **(4)** based on the chart map. In (4.1), we show how to compute the Jacobian of a differentiable function F from one manifold to another, respecting the charts of the manifolds. For the item **(2)**, adding a bias can be viewed as a "translation" in the Euclidean space, while in Definition 4.4 we define the translation on manifold-valued data using the group action. With these in hand, we are ready to present our proposed manifold version of these layers next.

Flow-based Models: Riemannian Manifold Case

We will now introduce the manifold counterpart of the key operations. See Table 4.2 for a summary of functions.

(a) Actnorm. Let s_r be the spatial resolution and c_r be the channel size, $X \in \mathcal{M}^{s_r \times c_r}$. We modify (4.6) to manifold-valued data using the operators we mentioned above in Key ingredients. The bias term is replaced by the group operators T while the multiplication $1/\sigma$ is replaced by the diagonal matrix S of size $m \times m$ in the space after the chart mapping $\Phi(\cdot)$. The layer function is defined as in (4.10).

Determinant of the Jacobian can be computed as shown below in (4.16). In general, s_r can be a tuple, i.e., for 3D data, it is a 3 dimensional tuple.

$$\det\left(\frac{dY}{dX}\right) = \prod_{s_r \times c_r} \left(\prod s_i\right) \det\left(\Psi \circ \Phi^{-1}\right) \tag{4.16}$$

(b) 1×1 convolution. We define a 1×1 convolution to offer the flexibility of interaction between channels. Here R is a $c_r \times c_r$ matrix applied after chart mapping $\Phi(\cdot)$. In general, we can learn any $R \in GL(c_r)$, i.e., a full rank matrix like in (4.7). But in practice, maintaining full rank is a hard constraint and may become unbounded. As a regularization, we choose R to be a rotation matrix. This layer function is defined as in (4.11) using the same notation as in (4.7).
Determinant of the Jacobian can be computed as shown below in (4.17). Notice that for R to be a rotation matrix, the contribution from $\det(R)$ is ± 1.

$$\det\left(\frac{dY}{dX}\right) = \prod_{s_r} \det(R) \det\left(\Psi \circ \Phi^{-1}\right) \tag{4.17}$$

(c) Affine coupling. For manifold-valued data, given $X \in \mathcal{M}^{s_r \times c_r}$ (where s_r and c_r are spatial and channel resolutions), we first split the data along the channel dimension, i.e., partition c_r into two parts denoted by $X_a \in \mathcal{M}^{s_r \times c_a}$ and $X_b \in \mathcal{M}^{s_r \times c_b}$, where $c_r = c_a + c_b$. From (4.8), we need to modify the scaling and translation. Here, $S \in (\mathbb{R}^{m \times m})^{s_r \times c_a}$ and $T \in G^{s_r \times c_a}$. These two operators play the same roles as in (4.8), scaling and translation. We need S to be full rank. If needed, one may use constraints like orthogonality or bounded matrix for numerical stability. After performing the coupling, we simply combine Y_a and Y_b to get $Y \in \mathcal{M}^{s_r \times c_r}$ as our output. This function is defined in (4.12).

Determinant of the Jacobian can be computed as:

$$\begin{bmatrix} \frac{dY_a}{dX_a} & \frac{dY_a}{dX_b} \\ \approx & \approx \\ \frac{dY_b}{dX_a} & \frac{dY_b}{dX_b} \\ \approx & \approx \end{bmatrix} \tag{4.18}$$

Similar to (4.9), observe that $\frac{dY_b}{\underset{\approx}{dX_a}}$ involves taking the gradient of a neural network! But fortunately, we only require the determinant of the Jacobian matrix, and the independence of Y_a on X_b saves the calculation of $\frac{dY_b}{\underset{\approx}{dX_a}}$ since $\frac{dY_a}{\underset{\approx}{dX_b}} = 0$. Thus, given $X, Y \in \mathcal{M}^{s_r \times c_r}$, the Jacobian determinant is given as

$$\det\left(\frac{dY}{\underset{\approx}{dX}}\right) = \prod_{s_r \times c_r} \det(S) \det\left(\Psi \circ \Phi^{-1}\right) \tag{4.19}$$

Distribution on the latent space. After the cascaded functional transformations described above, we transform X to the latent space $Z \in \mathcal{M}^{s_h \times c_h}$. We define a Gaussian distribution on Z, namely $P(Z; M, \Sigma)$, by inducing a multi-variate Gaussian distribution from $\mathbb{R}^m$ as

$$\exp\left(-\frac{(\Phi(Z) - \Phi(M))^T \Sigma^{-1} (\Phi(Z) - \Phi(M))}{2}\right) \Big/ C(\Sigma) \tag{4.20}$$

where$M \in \mathcal{M}^{s_h \times c_h}$ and $\Sigma \in \mathsf{SPD}(m)^{s_h \times c_h}$ (SPD denotes a symmetric positive definite matrix, similar to our use in Chapter 2 and 3). $C(\Sigma)$ is the normalization constant to make the total probability to be 1.

Learning Mappings between Manifolds

We can now ask the question: *can we draw manifold-valued data conditioned on another manifold-valued sample?* Due to the nature of the invertibility of our generative model, this seems to be possible since all we need to develop, in addition to what has been covered, is a scheme to sample data from Euclidean space conditioned on a vector-valued input.

Recently, extensions of the GLOW model (in a Euclidean setup) have been used to generate samples from space $\mathcal{X}$ conditioned on space $\mathcal{Y}$, see Sun et al. (2019). In this section, we roughly follow Sun et al. (2019) by using connections in a latent space but in a manifold setting to generate a sample from a manifold $\mathcal{M}$, conditioned on a sample on manifold $\mathcal{N}$. The underlying assumption is that there exists a (smooth) function from $\mathcal{N}$ to $\mathcal{M}$. The generation steps are as follows.

(1): Given variables $X \in \mathcal{M}^{s_x \times c_x}$ and $Y \in \mathcal{N}^{s_y \times c_y}$ with the dimension of the manifolds $\mathcal{M}$ and $\mathcal{N}$ to be m and n respectively, we use the two parallel GLOW models (as discussed above) to get the corresponding latent space. Let it be denoted by Z_m and Z_n respectively.

(2): After getting the respective latent spaces, we need to fit a distribution on it. Since we wish to generate samples from $\mathcal{M}$, the distribution on the respective latent space Z_m must be induced from the variables in Z_n, i.e., the latent space for $\mathcal{N}$. We do not have any constraint on the distribution parameters for Z_n, so, we use a Gaussian distribution with a fixed M and Σ on Z_n. The parameters for the Gaussian distribution on Z_m are defined as functions of Z_n. Formally, we define $P(Z_m; M_m, \Sigma_m)$ using (4.20), where, $M_m = \Psi_M^{-1}(F_M(\Phi_N(Z_n)))$ and $\Sigma_m = F_S(\Phi_N(Z_n))$. Here, the two functions F_M and F_S are modeled using a neural network. The scheme is shown in Figure 4.3.

Specific examples of manifolds. Finally, in order to implement (4.10), (4.11) and (4.12) mentioned in the previous sections, basic operations specific to a manifold are

(1) the choice of distance, d

(2) the isometry group, G

(3) the chart map Φ and its inverse, Φ^{-1}

We use three types of non-Euclidean Riemannian manifolds in the experiments presented in this work, they are

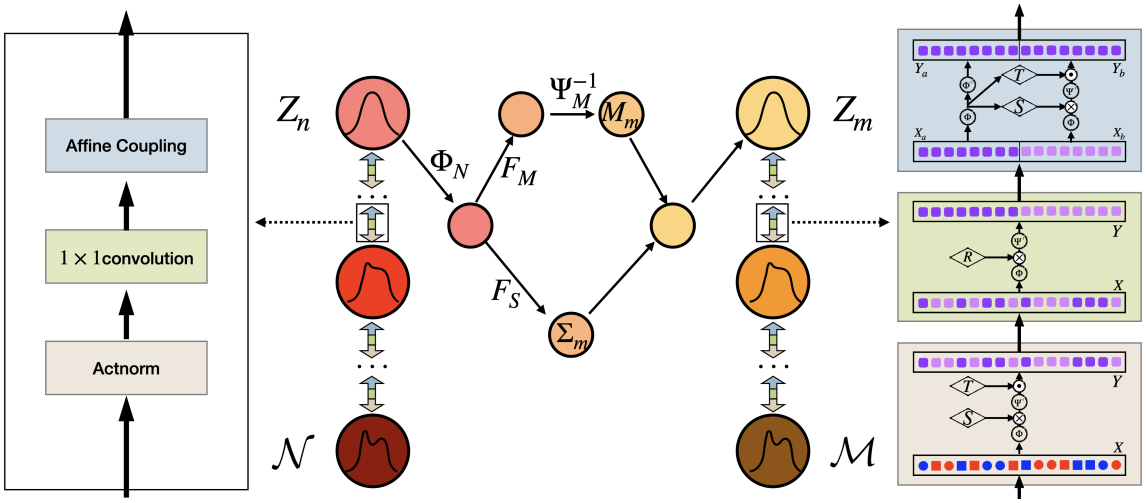

Figure 4.3: Transfer from the source manifold $\mathcal{N}$ to the target manifold $\mathcal{M}$ with the generative model. The detail of blocks of our model. The meanings of colors and shapes are the same as Figure 4.2, while all variables lie on the manifold instead of Euclidean space. The major difference between our manifold-valued GLOW and the original GLOW model is that we use a tangent space transformation before and after every operation. Different from Figure 4.2, there is no element-wise multiplication. The "$\cdot$" here is the group operation on the manifold-valued data. The "$\times$" is also the matrix multiplication in the tangent space.

(1) hypersphere, $\mathbf{S}^{n-1}$

(2) space of positive real numbers, $\mathbb{R}_+$

(3) space of $n \times n$ symmetric positive definite matrices $(\mathsf{SPD}(n))$.

We give the explicit formulation for the operations in Table 4.3.

4.4 Experiments

We demonstrate the experimental results of our model using two setups. First, we generate texture images based on the local covariances, which serves as a sanity check evaluation relative to another generative model for manifold-valued data

Table 4.3: The explicit formulation for the basic operations. Here P is an anchor point for chart map, which can be one of the poles. $\theta = d(P, X)$, and $SO(m)$ is the group of $m \times m$ special orthogonal matrices. $\mathbf{v} \in \mathbb{R}^{n-1}$. Chol is the Cholesky decomposition.

	$\mathbf{S}^{n-1}$	$\mathbb{R}_+$	$SPD(n)$
$d(X, Y)$	$\arccos\left(X^T Y\right)$	$\lvert\log(X/Y)\rvert$	$\lVert\text{logm}X^{-1}Y\rVert$
G	$SO(n-1)$	$\mathbb{R} \setminus \{0\}$	$SO(m)$
$\Phi(X)$	$\frac{\theta}{\sin(\theta)}\left(X - P\cos(\theta)\right)$	$\log(X)$	$\text{Chol}(X)$
$\Phi^{-1}(\mathbf{v})$	$P\cos(\lVert\mathbf{v}\rVert) + \frac{\mathbf{v}}{\lVert\mathbf{v}\rVert}\sin(\lVert\mathbf{v}\rVert)$	$\exp(\mathbf{v})$	$\mathbf{v}\mathbf{v}^T$

available at this time. The second experiment, which is our main scientific focus, generates orientation distribution function (ODF) images Hess et al. (2006) using diffusion tensor imaging (DTI) Basser et al. (1994); Alexander et al. (2007). *Note that, in this setting we construct the DTI scans from under-sampled diffusion directions.*

Baseline. Very recently, the $\mathcal{M}_e$-flow Brehmer and Cranmer (2020) was introduced, which provides a generative model for manifold-valued data. $\mathcal{M}_e$-flow uses an encoder to encode the manifold-valued data in the high-dimensional space into a low-dimensional Euclidean space. During generation, the model will generate the low-dimensional Euclidean data and warp it back to the manifold in the high-dimensional space. The benefit of this method is that it can learn the dimension of the unknown manifold, including natural images like ImageNet Deng et al. (2009). But for a known Riemannian manifold, the dimension d of the manifold is fixed. For example, $SPD(n)$ is of dimension $d = n(n+1)/2$, while $\mathbf{S}^m$ is of dimension $d = m$. Thus, for a known Riemannian manifold, $\mathcal{M}_e$-flow learns the chart using an encoder neural network and applies all the operations in the learned space with (known) dimension d. Another interesting recent proposal, manifoldWGAN, Huang et al. (2019) showed that it is possible to generate 32×32 resolution $SPD(3)$ matrices using WGAN. Due to the involved calculations needed by WGAN, extending it into high-dimension manifold-valued data including ODF ($\mathbf{S}^m$) will require non-trivial changes. Further, manifoldWGAN in its current form does not deal with

conditioning the generated images based on another manifold-valued data but is an interesting future direction to explore.

Now, we present experiments for generating texture images before moving to the more challenging ODF generation task.

Generating Texture Images

The earth texture images dataset was introduced in Yu et al. (2019). The train (and test) set have 896 (and 98) images. All images are augmented by random transformations and cropped to size 64×64. Our goal here is to generate texture images based on the local covariances of the three (R, G, B) channels. So the two manifolds are $\mathsf{SPD}(3)$ (for covariance matrix) and $\mathbb{R}^3_+$ (for texture images). Since $\mathcal{M}_e$-flow can only take the Euclidean data as the "conditioning variable", we vectorize the local covariances as the condition variable for $\mathcal{M}_e$-flow. The dimension of the learned space for $\mathcal{M}_e$-flow is chosen as 64 (default configuration from StyleGAN Karras et al. (2020)). For our case, we build two parallel manifold-GLOW with 8 blocks on each side. After every 2 blocks, the spatial resolution is reduced to half. In the latent space, we train a residual network with 3 residual blocks to map the distribution of the $\mathsf{SPD}(3)$ to $\mathbb{R}^3_+$. Example results are shown in Figure 4.4. Even in this simple setting, due to the encoder in the $\mathcal{M}_e$-flow, the generated images lose sharpness. Our model uses the information of the local covariances to generate superior texture images.

Main Focus: Diffusion MRI Dataset

Our main focus is the conditional synthesis of structural brain image data. We discussed the diffusion-weighted magnetic resonance imaging (dMRI) in Section 3.2. The main difference here is that we symmetrically/equally sampling 362 points (compared to 724 in Section 3.2) on the continuous ODF Garyfallidis et al. (2014). Each measurement is a 362D vector (non-negative entries; sum to 1). Using the square root parameterization Brody and Hughston (1998); Srivastava et al. (2007), the data at each voxel lies on the positive part of $\mathbf{S}^{361}$ manifold.

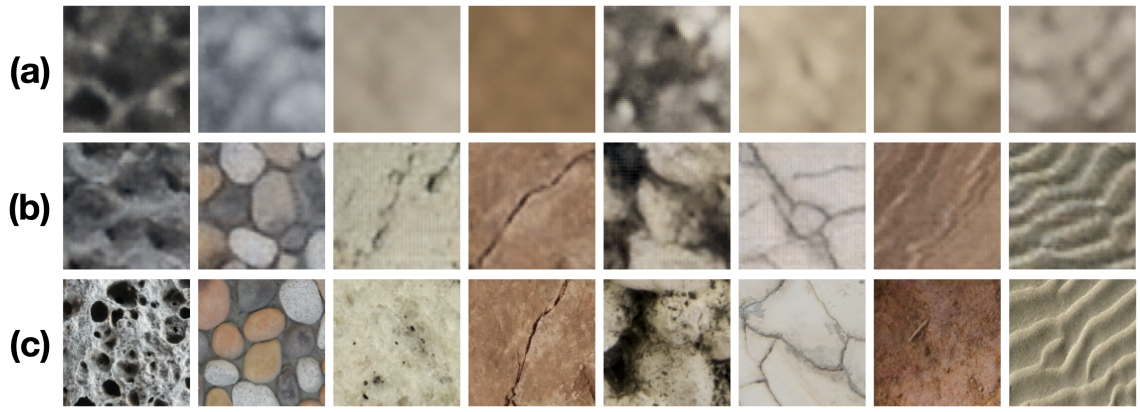

Figure 4.4: Generated images from (a) $\mathcal{M}_e$-flow, (b) ours, and (c) the ground truth. The condition is the local covariances of the RGB channels.

Here in this section, we seek to generate a 3D brain image where each voxel is a ODF from the corresponding DTI image (each voxel is a 3×3 SPD matrix). To make the setup more challenging (and scientifically interesting), we generate the DTI images only from randomly under-sampled diffusion directions. We now explain the following:

(**1**) rationale for the application

(**2**) data description

(**3**) model setup

(**4**) evaluations

Note that in the experiment, since we draw samples from the distribution on the latent space, conditioned on DTI, to get the target representation, we call it generation rather than reconstruction.

Why generating ODF from DTI is important? For dMRI, different types of acquisitions involve longer/shorter acquisition times. DTI can be acquired quickly, using diffusion-weighted imaging sequences that can be less than 5 mins. In contrast, to calculate ODF, longer imaging sequences with more diffusion directions and

often multi-shell sequences are needed, requiring longer scan times. To shorten the acquisition time with minimal compromise in the image quality, we require mechanisms to transform data acquired from shorter acquisitions (DTI) to a higher spatial resolution image: a field (or image) of ODFs. *This serves as our main motivation.*

However, there are some problems when attempting to do so.

(1) the per voxel degrees of freedom for ODF representation is 361 (lies on $\mathbf{S}^{361}$) while for DTI is 6 (lies on $\mathsf{SPD}(3)$). Hence, it is an ill-posed problem.

(2) requires mathematical tools to "transform" from one manifold (DTI representation) to another (ODF representation) while preserving structural information.

Now, we describe some details of the data, models and present the results.
Dataset. The dataset we choose for our experiments is the Human Connectome Project (HCP) Essen et al. (2013). The total number of subjects with diffusion measurements available is 1065: 852 were used as training and 213 as the test set. Demographic details are reported in Table 4.4 (please see Essen et al. (2013) for more details of the dataset). All raw dMRI images are pre-processed with the HCP diffusion pipeline with FSL's 'eddy' Andersson and Sotiropoulos (2016). After correction, ODF and DTI pairs were obtained using the Diffusion Imaging in Python (DIPY) toolbox Garyfallidis et al. (2014). Due to the memory requirements of the model and 3D nature of medical data, generation of an ODF image of the entire brain at once remains out of reach at this point, hence we resize the original data into $32 \times 32 \times 32$ but the process can proceed in a sliding window fashion as well.

Table 4.4: The demographics used in the study.

Dataset	**Age**				**Gender**	
	22-25	26-30	31-35	36+	Female	Male
All	224	467	364	10	575(54.0%)	490(46.0%)
Train	178	370	295	9	463(54.3%)	389(45.7%)
Test	46	97	69	1	112(52.6%)	101(47.4%)

Reduction in the memory costs. Since the entire 3D models for brain images are still too large to fit into the GPU memory, we need to further simplify the model without sacrificing the performance too much. Recently, NanoFlow Lee et al. (2020) was introduced to reduce the number of parameters for sequential data processing. The assumption of NanoFlow is that the Affine Coupling layer, if fully trained, can estimate the distribution for any parts of the input data in a fixed order. There will be some performance drop compared with training different Affine Coupling layers for different parts of the data. But the gain from reducing the parameters is significant. Thus, in our setup, due to the large 3D input, we apply the NanoFlow trick for DTI and ODF separately. For example, for the DTI data, we first split the entire data into 2τ slices called $\{X_1, X_2, ..., X_{2\tau}\}, \tau \geqslant 1$. Then we can share the two neural networks S and T in the Affine Coupling layer among these slices. The input of two neural networks S and T in Affine Coupling layer would be $X_{2k-1}, k = 1, 2, ..., \tau$, while the output will be the estimated mean and variance of $X_{2k}, k = 1, 2, ..., \tau$ respectively. Due to sharing weights, the number of parameters reduces and becomes feasible for training our 3D DTI and ODF setups.

Model Setup. In order to set up our model, we first build two flow-based streams for DTI and ODF separately. Then, in the latent space, we train a transformation operating between the Gaussian distribution variable on the manifold $\mathbf{S}^{361}$ and the Gaussian distribution variable on the manifold $\mathrm{SPD}(3)$. This architecture with two flow-based models and the transformation module can be jointly trained as shown in Figure 4.5. We use 6 basic blocks of our manifold GLOW, and after every 2 blocks, reduce the resolution by half. This setup is the same for both DTI and ODF. We use 3 residual network blocks to map the latent space from DTI to ODF. The samples are presented to the model in paired form, i.e., a DTI image (field of SPD matrices) and a corresponding ODF image (a field of ODFs). To reduce the number of parameters for this 3D data, we use a similar idea as NanoFlow Lee et al. (2020) that shares the Affine Coupling layer for DTI and ODF separately, with setting $\tau = 32$. As a comparison, for the baseline model $\mathcal{M}_e$-flow, the learned dimension will be $32 \times 32 \times 32 \times d$ where $d = 6$ for DTI and $d = 361$ for ODF. While $\mathcal{M}_e$-flow could be trained for our texture experiments, here, the memory

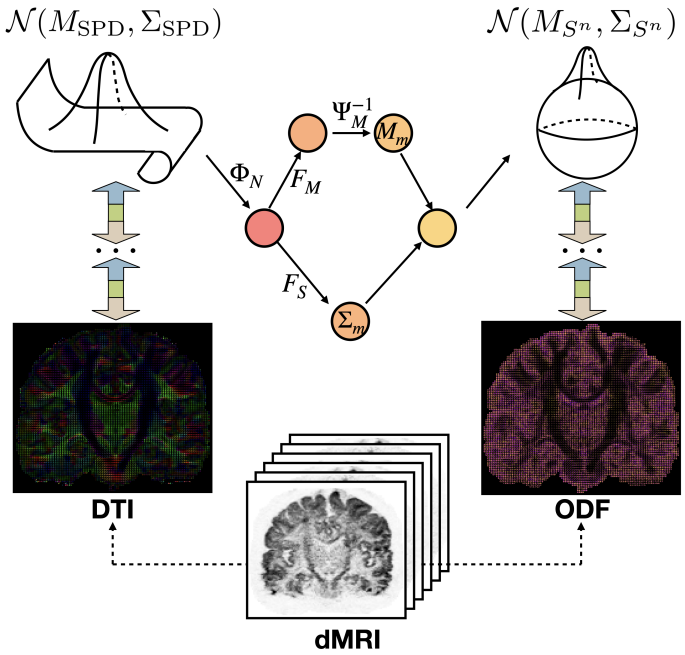

Figure 4.5: The transformation from DTI to ODF. Both are generated from dMRI. But there might not be dMRI available in some situations. Thus, we want to train the network to transfer DTI to ODF. The latent space is the Gaussian distribution variable.

requirements are quite large, quantitatively the number of parameters required for $\mathcal{M}_e$-flow and our model are 1.3*e*18 and 2.1*e*8 respectively. A similar situation arises in the Euclidean space version of GLOW which also does not leverage the intrinsic Riemannian metric: therefore, the memory cost will be 6000$\times$ more than the natural images which have dimension $224 \times 224 \times 3$. This is infeasible even on clusters and therefore, results from these baselines are very difficult to obtain.

Choice of metrics. We will use "*reconstruction error*" using the distance in Table 4.3. Although the task here is generation, measuring reconstruction error assesses how "similar" the original ODF is to the generated ODF, generated directly from the corresponding DTI representation. We also perform a group difference analysis to identify statistically different regions across groups (grouped by a dichotomous variable). Since HCP only includes healthy subjects (HCP aging is smaller), we can

perform a group difference test based on gender, i.e., *male vs. female*. We evaluate overlap: how/whether group-wise different regions on the generated/reconstructed data agrees with those on the actual ODF images.

Generation results. We present quantitative and qualitative results for generation of ODF from its DTI representation. In Figure 4.6(a), we show a few example slices from the given DTI and the generated ODF. Overall, the reconstruction error was $4.0(\pm 0.45) \times 1e\text{-}4$. Since perceptually comparing fidelity between generated and ground truth images is difficult, we perform the following quantitative analysis: **(a)** a histogram of the reconstruction error over all 213 test subjects (shown in Figure 4.6(b)) **(b)** an error matrix showing how similar the generated ODF image is with the other "incorrect" samples of the population. The goal is to assess if the generated ODF is distinctive across different samples (shown in Figure 4.6(c)) . From the histogram presented in Figure 4.6(b), we can see that the reconstruction error is consistently low over the entire test population. Now, we generate Figure 4.6(c) as follows. For each subject in the test population, we randomly select 29 samples (subjects) from the population and compute the reconstruction error with the generated ODF. This gives us a 30×30 matrix (similar to the confusion matrix). Figure 4.6(c) shows the average of 10 runs: lighter shades mean a larger reconstruction error. So, we should ideally see a dark diagonal, which is approximately depicted in the plot. This suggests that for the test population, the generation is meaningful (preserves structures) and distinctive (maintains variability across subjects). There are only few experiments described in the literature on generation of dMRI data Huang et al. (2019); Anctil-Robitaille et al. (2020). While Huang et al. (2019) shows the ability to generate 2D (32×32) DTI, the techniques described here can operate on 3D ODF ($\mathbf{S}^{361}$) data and should offer improvements.

Group difference analysis. We now quantitatively measure if the reconstruction is good enough so that the generated samples can be a good proxy for downstream statistical analysis and yield improvements over the same analysis performed on DTI. We run permutation testing with 10000 independent runs and compute the per-voxel p-value to see which voxels were statistically different between the groups for the following settings **(a)** original ODF **(b)** generated ODF **(c)** DTI **(d)** functional

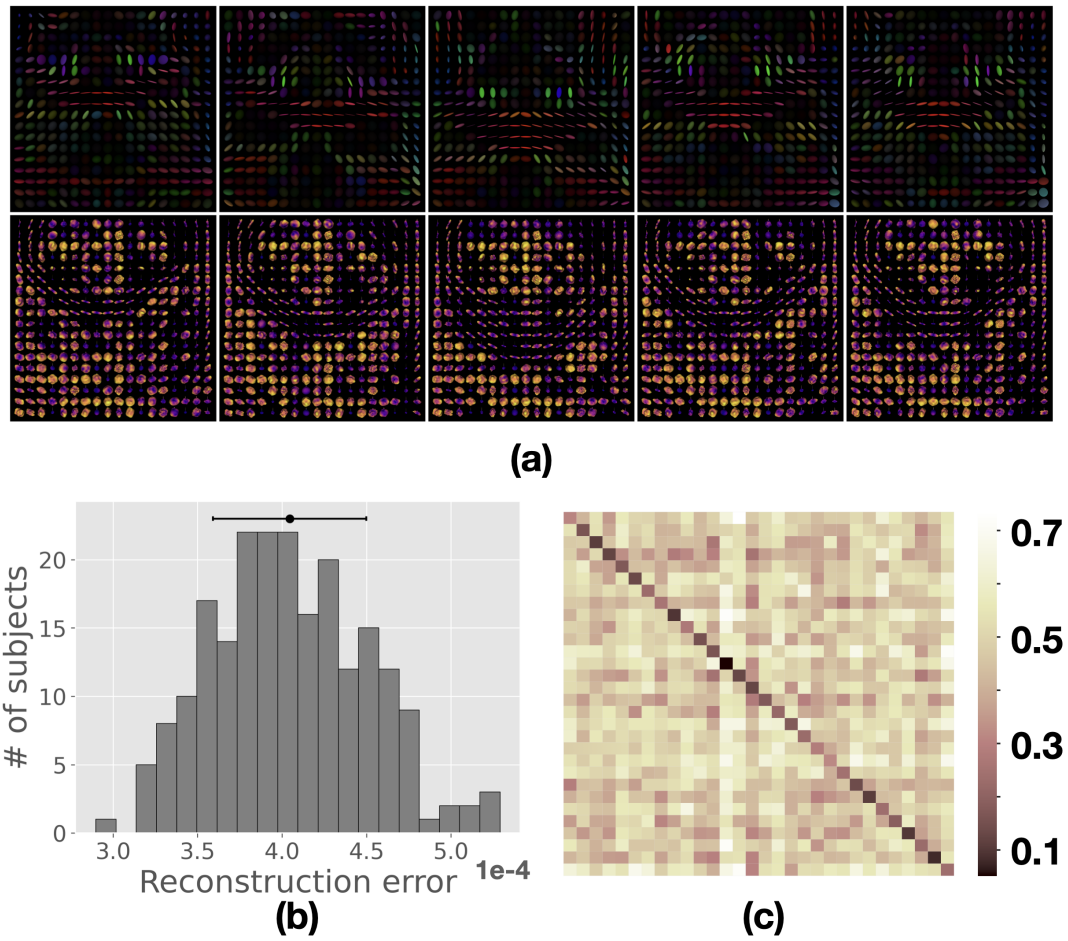

Figure 4.6: (a) Generated ODF from corresponding DTI. Each pair here contains the input DTI (*top*) and the generated ODF (*bottom*) (b) The distribution of reconstruction error over the testing population. (c) Reconstruction error over the test population shows that the generated image is closest to its own corresponding image (diagonal dominance))

anisotropy (FA) representation (commonly used summary of DTI). Both DTI and FA are commonly used for assessing statistically significant differences across genders Menzler et al. (2011); Kanaan et al. (2012). But since ODF contains more structural information than either the FA or DTI, our generated ODF should be able to pick up more statistically significant regions over DTI or FA. We evaluate the

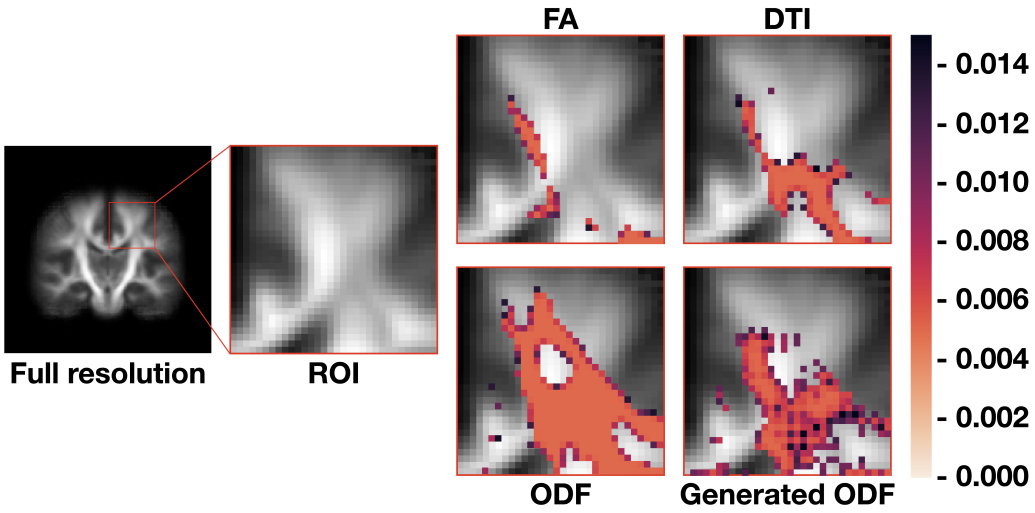

Figure 4.7: The p-value of one of the ROIs of the entire brain scan with full-resolution. We show that our proposed method can generate meaningful ODF with respect to the group level differences.

intersection of significant regions with the original ODF (the original ODF contains the most information). We compute the *intersection over union* (IoU) measure. For the whole brain, FA will have IoU 0.04, while DTI has IoU 0.16. The generated ODF has IoU 0.22. We see that the generated ODF has a larger intersection in the statistically significant regions with the original ODF and offers improvements over DTI. This provides some evidence that the generated ODF preserves the signal that is different across the male/female groups. We also show a zoomed in example of a ROI for the full-resolution images in Figure 4.7. The p-values for different ROIs are all < 0.001 in both the original ODF and our generated ODF, indicating consistency of our results, at least in terms of regions identified in downstream statistical analysis. Note that the analysis on the real ODF images serves as the ground truth.

4.5 Conclusions

A number of deep neural network formulations have been extended to manifold-valued data in the last two years. While most of these developments are based on models such as CNNs or RNNs, in this work, we study the generative regime: we introduce a flow-based generative model on the Riemannian manifold. We show that the three types of layers, Actnorm, Invertible 1×1 convolution, and Affine Coupling layers in such models, can be generalized/ adapted for manifold-valued data in a way that preserves invertibility. We also show that with the transformation in the latent space between the two manifolds, we can generate manifold-valued data based on the information from another manifold. We demonstrate good generation results in the representation of ODF given DTI on the Human Connectome dataset. While the current formulation shows mathematical feasibility and promising results, additional work on the methodological and the implementation side is needed to reduce the runtime to a level where the tools can be deployed in scientific labs. The code for the algorithms described in this chapter is available at `https://github.com/zhenxingjian/Dual_Manifold_GLOW`.

5 SIMPLER CERTIFIED RADIUS MAXIMIZATION BY PROPAGATING COVARIANCES

In previous chapters, we mainly discussed how manifold-valued data (both SPD matrices and distribution functions) can be used to measure the micro-structural information in dMRI and how the presented methods can facilitate improved analysis of such data. In this chapter, we will continue to focus on SPD matrices but we will switch the application focus. Instead of neuroimaging problems, here we will see how directly working with SPD matrices (derived in a specific way) can help improve the robustness of a given neural network.

Before we get into the details, recall that one strategy for adversarially training a robust model is to maximize its certified radius – the neighborhood around a given training sample for which the model's prediction remains unchanged (assuming we are interested in classification problems). The scheme typically involves analyzing a "smoothed" classifier where one estimates the prediction corresponding to Gaussian samples in the neighborhood of each sample in the mini-batch, accomplished in practice by Monte Carlo sampling. We will discuss why such method can be problematic in Section 5.2. Further, we investigate the hypothesis that this sampling bottleneck can potentially be mitigated by identifying ways to directly propagate the covariance matrix of the smoothed distribution through the network. To this end, we find that other than certain adjustments to the network, propagating the covariances must also be accompanied by additional accounting that keeps track of how the distributional moments transform and interact at each stage in the network which we will discuss in Section 5.3. We show how satisfying these criteria yields an algorithm for maximizing the certified radius on datasets including Cifar-10, ImageNet, and Places365 while offering runtime savings on networks with moderate depth, with a small compromise in overall accuracy. The experimental results are presented in Section 5.4. We describe the details of the key modifications that enable practical use. Via various experiments, we evaluate when our simplifications are sensible, and what the key benefits and limitations are. The work described in this

chapter appeared as a conference paper in CVPR 2021 Zhen et al. (2021a).

5.1 Introduction

The prevailing approach for evaluating the performance of a deep learning model involved assessing its overall accuracy profile on one or more benchmarks of interest. But the realization that many models were not robust to even negligible adversarially-chosen perturbations of the input data Szegedy et al. (2014); Biggio et al. (2013); Ilyas et al. (2018); Carlini and Wagner (2017b), and may exhibit highly unstable behavior Bojarski et al. (2016); Lécuyer et al. (2019); Mansour (2018) has led to the emergence of robust training methods (or robust models) that offer, to varying degrees, immunity to such adversarial perturbations. Adversarial training has emerged as a popular mechanism to train a given deep model robustly Papernot et al. (2016); Tramèr et al. (2018). Each mini-batch of training examples shown to the model is supplemented with adversarial samples. It makes sense that if the model parameter updates are based on seeing enough adversarial samples which cover the perturbation space well, the model is more robust to such adversarial examples at test time Goodfellow et al. (2015); Huang et al. (2015); Madry et al. (2018). The approach is effective although it often involves paying a premium in terms of training time due to multiple gradient calculations Shafahi et al. (2019). However, many empirical defenses can fail when the attack is stronger Carlini and Wagner (2017a); Uesato et al. (2018); Athalye and Carlini (2018).

While ideas to improve the efficiency of adversarial training continue to evolve in the literature, a complementary line of work seeks to avoid adversarial sample generation entirely. One instead derives a *certifiable robustness* guarantee for a given model Weng et al. (2018); Wong and Kolter (2018); Zhang et al. (2018); Mirman et al. (2018); Zhang et al. (2019); Singh et al. (2018); Balunovic et al. (2019). The overall goal is to provide guarantees that *no perturbation* within a certain range will change the prediction of the network. An earlier proposal, interval bound propagation (IBP) Gowal et al. (2018), used convex relaxations at different layers of the network to derive the guarantees. Unfortunately, the bounds tend to get very

loose as the network depth increases, see Figure 5.1. Thus, the applicability to large high resolution datasets remains under-explored at this time.

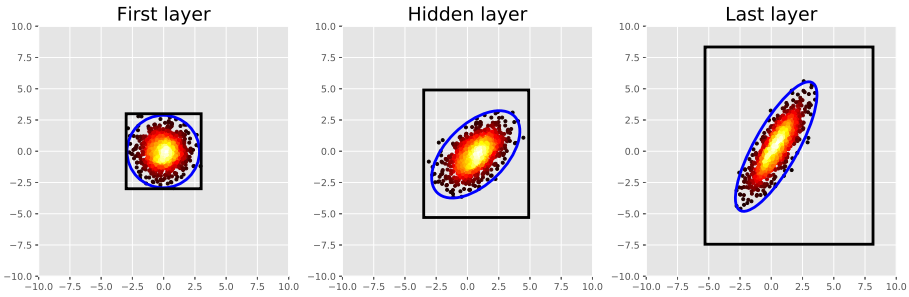

Figure 5.1: Example of three methods for certifiable robustness on a two layers MLP. We show results of the input layer, hidden layer, and the output layer here. Black boxes based on using IBP Gowal et al. (2018). Red dots come from the sampling idea from Zhai et al. (2020). Ovals are covariance matrices if they are tracked exactly while considering interactions.

Recently, following the idea in Li et al. (2019); Lécuyer et al. (2019) at a high level, Cohen et al. (2019) introduced an interesting randomized smoothing technique, which can be used to certify the robust radius C_R. Assume that we have a base network $f_\theta(\cdot)$ for classification. On a training image $\mathbf{x} \in \mathbb{R}^d$, the output $f_\theta(\mathbf{x}) \in \mathcal{Y}$ is the predicted label of the image $\mathbf{x}$. Using $f_\theta(\cdot)$, we can build a "smoothed" neural network $g_\theta(\cdot)$.

$$g_\theta(\mathbf{x}) = \arg\max_{c \in \mathcal{Y}} \mathbb{P}(f_\theta(\mathbf{x} + \boldsymbol{\varepsilon}) = c), \text{where } \boldsymbol{\varepsilon} \sim \mathcal{N}(0, \sigma^2 I)$$

Here, σ can be thought of as a trade-off between the robustness and the accuracy of the smoothed classifier $g_\theta(\cdot)$. One can obtain a theoretical certified radius C_R which states that when $\|\boldsymbol{\delta}\|_2 \leqslant C_R$, the classifier $g_\theta(\mathbf{x} + \boldsymbol{\delta})$ will have same label y as $g_\theta(\mathbf{x})$. MACER Zhai et al. (2020) nicely extended these ideas and also presented a differentiable form of randomized smoothing showing how it enables maximizing the radius. Internally, a sampling scheme is used, where empirically, the number

of samples to get an accurate estimation could be large. As Figure 5.2 shows, one needs 100 samples for a good estimation of the distribution of ImageNet.

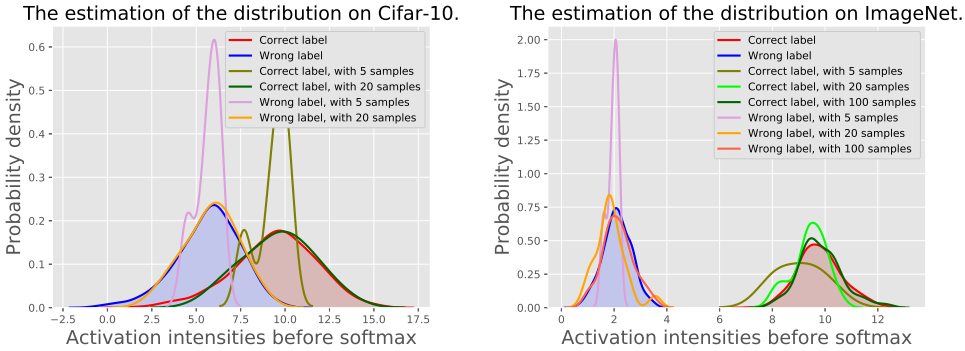

Figure 5.2: Example of Monte Carlo estimation on a different dataset. If the distributions of the correct and wrong labels are farther, the network is more robust. As the size of images grows, the number of samples for a good estimate also increases.

Main intuition. MACER Zhai et al. (2020) showed that by sampling from a Gaussian distribution and softening the estimation of the distribution in the last layer, maximizing the certified radius is feasible. It is interesting to ask if tracking the "maximally perturbed" distribution directly – in the style of IBP – is possible without sampling. Results in Xiao et al. (2018) showed that the pre-activation vectors are i.i.d. Gaussian when the channel size goes to infinity. While unrealistic, it provides us a starting point. Since a Gaussian distribution can be fully characterized by the mean and the covariance matrix, we can represent the Gaussian distribution by a product manifold structure $(\mathbb{R}^n \times \mathrm{SPD}(n))$, where the mean is in $\mathbb{R}^n$ and the covariance matrix is in $\mathrm{SPD}(n)$. Then, we can track these two quantities as it passes through the network until the final layer, where the radius is calculated. If implemented directly, this scheme must involve keeping track of how pixel correlations influence the entries of the covariances from one layer to the next, and the bookkeeping needs grow rapidly. Alternatively, Xiao et al. (2018) uses the fixed point of the covariance matrix to characterize it while it passes through the network, but this idea is not adaptable for maximizing the radius task in Zhai et al. (2020). We

will use other convenient approximations of the covariance to make direct tracking of the distribution of the perturbation feasible.

Other applications of certified radius maximization. Training a robust network is also useful when training in the presence of noisy labels Angluin and Laird (1987); Goldberger and Ben-Reuven (2017); Patrini et al. (2017). Normally, both crowd-sourcing from non-experts and web annotations, common strategies for curating large datasets introduce noisy labels. It can be difficult to train the model directly with the noisy labels without additional care Zhang et al. (2017). Current methods either try to model the noise transition matrix Goldberger and Ben-Reuven (2017); Patrini et al. (2017), or filter "correct" labels from the noisy dataset by collecting a consensus over different neural networks Han et al. (2018); Jiang et al. (2018); Malach and Shalev-Shwartz (2017); Ren et al. (2018). This leads us to consider *whether we can train the network from noisy labels without training any auxiliary network?* A key observation here is that the margin of clean labels should be smoother than the noisy labels (as shown in Figure 5.3).

Contributions. This chapter shows how several known results characterizing the behavior of (and upper bounds on) covariance matrices (SPD matrices) that arise from interactions between random variables with known covariance structure can be leveraged to obtain a simple scheme that can propagate the distribution (perturbation applied to the training samples) through the network. While SPD matrices in the previous chapter directly encoded the measurements, here the SPD matrices are artificially obtained from the data and pertain to an approximation of its perturbations. The approach leads to a sampling-free method that performs favorably when compared to Zhai et al. (2020) and other similar approaches when the network depth is moderate. We show that our method is $5\times$ faster on Cifar-10 dataset and $1.5\times$ faster on larger datasets including ImageNet and Places365 relative to the current state-of-the-art without sacrificing much of the performance. Also, we show that the idea is applicable for training with noisy labels.

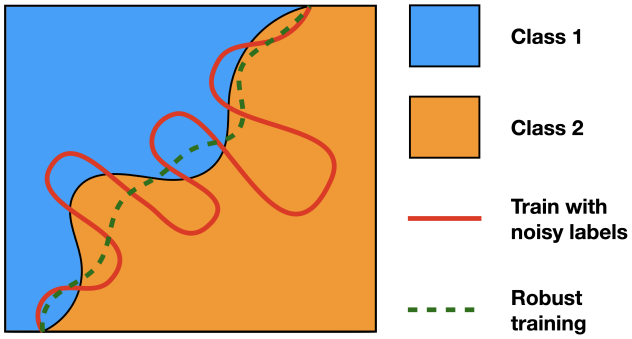

Figure 5.3: When directly training with noisy labels, the margin will resemble the red line. Using a robust network, the margin will resemble the green line.

5.2 Robust Radius via Randomized Smoothing

We will briefly review the relevant background on robust radius calculation using Monte-Carlo (MC) sampling.

What is the robust radius? In order to measure the robustness of a neural network, the *robust radius* has been shown to be a sensible measure Weng et al. (2018); Cohen et al. (2019). Given a trained neural network f_θ, the ℓ_2-*robustness at data point* $(\mathbf{x}, y)$ is defined as the **largest** radius R of the ball centered at $\mathbf{x}$ such that all samples within the ball will be classified as y by the neural network f_θ. Analogously, the ℓ_2-*robustness of* f_θ is defined as the **minimum** ℓ_2-robustness at data point $(\mathbf{x}, y)$ over the dataset. But calculating the robust radius for the neural network can be hard; Weng et al. (2018) provides a hardness result for the ℓ_1-robust radius. In order to make computing ℓ_2-robustness tractable, the idea in Cohen et al. (2019) suggests working with a tight lower bound, called the "Certified Radius", denoted by $0 \leqslant C_R \leqslant R$. Let us now briefly review the ideas in Cohen et al. (2019) in the context of a given base classifier $f_\theta(\cdot)$.

Note that we want to certify that there will be *no adversarial samples* within a radius of C_R. By smoothing out the perturbations ε around the input image/data $\mathbf{x}$ for the base classifier $f_\theta(\cdot)$, intuitively it will be harder to find an adversarial sample, since it will actually require finding a "region" of adversarial samples. If

we can estimate a lower bound on the probability of the base classifier to correctly classify the perturbed data $\mathbf{x} + \boldsymbol{\varepsilon}$, denoted as $\underline{p_{c_x}}$, as well as an upper bound of the probability of an incorrect classification $\overline{p_{\widetilde{c}}} \leqslant 1 - \underline{p_{c_x}}$, where c_x is the true label of $\mathbf{x}$ and $\widetilde{c}$ is the "most likely to be confused" incorrect label, a nice result for the smoothed classifier $g_\theta(\cdot)$ is available,

Theorem 5.1. *Cohen et al. (2019) Let $f_\theta : \mathbb{R}^d \to \mathcal{Y}$ be any deterministic or random function, and let $\boldsymbol{\varepsilon} \sim \mathcal{N}(\mathbf{0}, \sigma^2 I)$. Let g_θ be the randomized smoothing classifier defined as $g_\theta(\mathbf{x}) = \arg\max_{c \in \mathcal{Y}} p(f_\theta(\mathbf{x} + \boldsymbol{\varepsilon}) = c_x)$. Suppose $c_x, \widetilde{c} \in \mathcal{Y}$ and $\underline{p_{c_x}}, \overline{p_{\widetilde{c}}} \in [0, 1]$ satisfy $p(f_\theta(\mathbf{x} + \boldsymbol{\varepsilon}) = c_x) \geqslant \underline{p_{c_x}} \geqslant \overline{p_{\widetilde{c}}} \geqslant \max_{\widetilde{c} \neq c_x} p(f_\theta(\mathbf{x} + \boldsymbol{\varepsilon}) = \widetilde{c})$. Then $g_\theta(\mathbf{x} + \boldsymbol{\delta}) = c_x$ for all $\|\boldsymbol{\delta}\|_2 < C_R$, where $C_R = \frac{\sigma}{2}(\Phi^{-1}(\underline{p_{c_x}}) - \Phi^{-1}(\overline{p_{\widetilde{c}}}))$.*

The symbol Φ denotes the CDF of the standard Normal distribution. Φ and Φ^{-1} are involved because of smoothing the Gaussian perturbation $\boldsymbol{\varepsilon}$. The proof of this theorem can be found in Cohen et al. (2019).

How to compute the robust radius? Using Theorem 5.1, we will need to compute the lower bound $\underline{p_{c_x}}$, the main ingredient to compute C_R. In Cohen et al. (2019), the authors introduced a sampling-based method to compute the lower bound of $\underline{p_{c_x}}$ in the test phase. The procedure first samples n_0 noisy samples around $\mathbf{x}$ and passes it through the base classifier f_θ to estimate the classified label *after* smoothing. Then, we sample n noisy samples, where $n \gg n_0$, to estimate the lower bound of $\underline{p_{c_x}}$ for a certain confidence level α.

5.3 Track Distribution Approximately

In the last section, we discussed how to calculate $\underline{p_{c_x}}$ in a sampling (Monte Carlo) based setting. However, this method is based on counting the number of correctly classified samples, which is not differentiable during training. In order to tackle this problem, Zhai et al. (2020) introduced an alternative – soft randomized smoothing

– to calculate the lower bound

$$\underline{p_{c_x}} = \mathbb{E}_{\varepsilon \sim \mathcal{N}(\mathbf{0}, \sigma^2 I)} \left[\frac{e^{\beta u_\theta^{c_x}(x+\varepsilon)}}{\sum_{c' \in \mathcal{Y}} e^{\beta u_\theta^{c'}(x+\varepsilon)}} \right] \tag{5.1}$$

where u_θ is the network f_θ **without the last softmax layer**, i.e.,

$$f_\theta = \arg\max \operatorname{softmax}(u_\theta)$$

while β is a hyperparameter.

From Figure 5.2, observe that if we have enough MC samples, we can reliably estimate $\underline{p_{c_x}}$ effectively by counting the number of correctly classified samples. If we can bypass MC sampling to estimate the final distribution, the gains in runtime can be significant. However, directly computing the joint distribution of the perturbations of all the pixels is infeasible: we need simplifying assumptions.

Gaussian pre-activation vectors. The first assumption is to use a Gaussian distribution to fit the pre-activation vectors. As briefly mentioned before, this is true when the channel size goes to infinity by the central limit theorem. In practice, when the channel size is large enough, e.g., a Resnet-based architecture He et al. (2016a), this assumption may be acceptable with a small error (evaluated later in experiments). Therefore, we will only consider the first two moments, which is reasonable for Gaussian perturbation Wishart (1928).

Second moments. Our second assumption is that in each layer of a convolution network, the second moments are identical for the *perturbation* of all pixels. That is the input pixels share identical second moments from a fixed Gaussian perturbation ε. Due to weight sharing and the linearity of the convolution operators, the second moments will only depend on the kernel matrix without the position information. A more detailed discussion is in Observation 5.2.

Notations and setup. Let N be the number of channels. We use Σ as the covariance matrices of the distribution of the pixel after the perturbation in the input layer. Here, we define each pixel as a random vector with dimension of N (number of channels). That is to say, $\Sigma \in \mathsf{SPD}(N)$. The input perturbation comes from Gaussian

perturbation ε where $\Sigma = \sigma^2 I$. As the image passes through the network, the input perturbation directly influences the output at each pixel as a function of the network parameters. We use $\Sigma_i \in \mathbb{R}^{N\times N}$, shorthand for $\Sigma_{\mathbf{x}_i}$, to denote the covariance of the perturbation distribution associated with pixel i of image $\mathbf{x}$ denoted as $\mathbf{x}_i$. We call $\Sigma[i,j]$ as the (i,j)-entry of Σ. Notice that the N changes from one layer to the other as the number of channels are different. So, the size of Σ will change. Let M_q be the number of pixels in the q^{th} layer input, i.e., for $q = 1$, M_1 is the number of pixels in the 1^{st} hidden layer of the network.

Similarly, $\boldsymbol{\mu}_{\mathbf{x}_i}$ or $\boldsymbol{\mu}_i \in \mathbb{R}^N$ is the mean of the distribution of the pixel $\mathbf{x}_i$ intensity after the perturbation. In the input layer, since the perturbation $\varepsilon \sim \mathcal{N}(0, \sigma^2 I)$, $\boldsymbol{\mu}_i = \mathbf{x}_i$. At the u_θ layer, the number of channels is the number of classes, with the number of pixels being 1. We use $\boldsymbol{\mu}[c_{\mathbf{x}}]$ and $\Sigma[c_{\mathbf{x}}, c_{\mathbf{x}}]$ to denote the $c_{\mathbf{x}}$ component of $\boldsymbol{\mu}$ and $(c_{\mathbf{x}}, c_{\mathbf{x}})$-entry of Σ respectively. To denote the cross-correlation between two pixels $\mathbf{x}_i, \mathbf{x}_j$, we use $E_{\mathbf{x}_i\mathbf{x}_j}$ or $E_{ij} \in \mathbb{R}^{N\times N}$. Note that this cross-correlation is across channels. For the special case where channel size $N = 1$, we will use $\sigma^{(i)} \in \mathbb{R}$ to represent the variance in the i^{th} layer. Let us define,

$$c_{\mathbf{x}} = \arg\max_{c\in\mathcal{Y}} \boldsymbol{\mu}[c], \quad \widetilde{c} = \arg\max_{c\in\mathcal{Y}, c\neq c_{\mathbf{x}}} \boldsymbol{\mu}[c] \tag{5.2}$$

Let the number of classes $C = |\mathcal{Y}|$. Then, we can state the following.

Observation 5.1. *Using u_θ, the prediction of the model can be written as $f_\theta(\mathbf{x}) = \arg\max_{c\in\mathcal{Y}}$ softmax$(u_\theta(\mathbf{x}))$. Assume $u_\theta(\mathbf{x}) \sim \mathcal{N}(\boldsymbol{\mu}, \Sigma)$, where $\mathbf{x} \sim \mathcal{N}(\boldsymbol{\mu}_{\mathbf{x}}, \Sigma_{\mathbf{x}})$, $\boldsymbol{\mu} \in \mathbb{R}^C$ and $\Sigma \in \mathbb{R}^{C\times C}$. Then the estimation of $\underline{p_{c_{\mathbf{x}}}}$ is*

$$\underline{p_{c_{\mathbf{x}}}} = \Phi\left(\frac{\boldsymbol{\mu}[c_{\mathbf{x}}] - \boldsymbol{\mu}[\widetilde{c}]}{\sqrt{\Sigma[c_{\mathbf{x}}, c_{\mathbf{x}}] + \Sigma[\widetilde{c}, \widetilde{c}] - 2\Sigma[c_{\mathbf{x}}, \widetilde{c}]}}\right) \tag{5.3}$$

Notice that propagating $\boldsymbol{\mu}$ through the network is simple, since tracking the mean is the same as directly passing it through the network when there is no nonlinear activation, and requires no cross-correlation between pixels. But tracking Σ at each step of the network can be challenging and some approximation techniques have

been used in literature for simple networks Scaglione et al. (2008). To see this, let us consider a simple 1-D example.

Bookkeeping problem. Consider a simple 1-D convolution with a kernel size k. By Observation 5.1, we will need the distribution of $u_\theta(\mathbf{x})$ of the i^{th} layer (i.e., the network without the softmax layer). Directly, this will involve taking into account k^1 pixels in the $(i-1)^{th}$ layer, and k^2 pixels in the $(i-2)^{th}$ layer. We must calculate the covariance Σ and also calculate the cross-correlation E between all k^q pixels in $(i-q)^{th}$ layer. This trend stops when we hit $k^q > M_{i-q}$, where M_{i-q} is the number of pixels at $(i-q)$ layer, but it is impractical anyway.

If we temporarily assume that the network involves no activation functions, and if the input perturbation is identical for all pixels, then the variance of all pixels after perturbation is also identical. Thus, the variance of each pixel only relies on the variance of the perturbation and not on the pixel intensity. This may allow us to track one covariance matrix instead of M for all M pixels.

Observation 5.2. *With the input perturbation ε set to be identical along the spatial dimension and without nonlinear activation function, for the q^{th} hidden convolution layer with $\{\mathbf{h}_i\}_{i=1}^{M_q}$ output pixels, we have $\Sigma_{\mathbf{h}_i}^{(q)} = \Sigma_{\mathbf{h}_j}^{(q)}, \forall i, j \in \{1, \cdots, M_q\}$.*

Observation 5.2 only reduces the cost marginally: instead of computing all the covariances of the perturbation for all pixels, $\Sigma_1^{(i-q)}, \Sigma_2^{(i-q)}, \cdots, \Sigma_k^{(i-q)}$, we only need to compute a single $\Sigma^{(i-q)}$. Unfortunately, we still need to compute all different $E_{ij}^{(i-2)}$ that will contribute to $u_\theta(\mathbf{x})$. Thus, due to these cross-correlation terms E_{ij}, the overall computation is still not feasible. In any case, the assumption itself is unrealistic: we *do* need to take nonlinear activations into account which will break the identity assumption of the second moments. For this reason, we explore a useful approximation which we discuss next.

How to Make Distribution Tracking Feasible

From the previous discussion, we observe that a key bottleneck of tracking distribution across layers is to track the interaction between pairs of pixels, i.e., cross-

correlations. Thus, we need an estimate of the cross-correlations between pixels. In Hanebeck et al. (2001), the authors provide an upper-bound on the joint distribution of two multivariate Gaussian random variables such that the upper bounding distribution contains **no cross-correlations**. This result will be crucial for us.

Formally, let $\mathbf{x}_1, \mathbf{x}_2 \in \mathbb{R}^N$ be two random vectors representing two pixels with N channels. Without any loss of generalization, assume that $\mathbf{x}_1 \sim \mathcal{N}(\mathbf{0}, \Sigma_1)$, and $\mathbf{x}_2 \sim \mathcal{N}(\mathbf{0}, \Sigma_2)$ (if the mean is not $\mathbf{0}$, we can subtract the mean without affecting the covariance matrix). Also, assume that we do not know the cross-correlation between $\mathbf{x}_1$ and $\mathbf{x}_2$, i.e., E_{12}. Instead, the correlation coefficient r is bounded by r_{max}, i.e., $|r| \leqslant r_{max}$.

With the above assumptions, we can bound the covariance matrix of the joint distribution of two N-dimensional random vectors $\mathbf{x}_1, \mathbf{x}_2$ by two independent random vectors $\widehat{\mathbf{x}_1}, \widehat{\mathbf{x}_2}$. We will use the notation "$\widehat{\cdot}$" to denote the upper bound estimation of "$\cdot$". The upper bound here means that $[\widehat{\Sigma} - \Sigma]$ is a positive semi-definite matrix, where $\widehat{\Sigma}$ is the joint distribution of the two independent random vectors $\widehat{\mathbf{x}_1}$ and $\widehat{\mathbf{x}_2}$. Here, Σ is the joint distribution of $\mathbf{x}_1, \mathbf{x}_2$ with correlation. Formally,

Theorem 5.2. *Hanebeck et al. (2001) When* $\widehat{\mathbf{x}_1} \sim \mathcal{N}(\mathbf{0}, \widehat{\Sigma_1} = \tau_1\Sigma_1)$, *and* $\widehat{\mathbf{x}_2} \sim \mathcal{N}(\mathbf{0}, \widehat{\Sigma_2} = \tau_2\Sigma_2)$, *the covariance matrix* $\mathbf{B} = \widehat{\Sigma} = \begin{bmatrix} \tau_1\Sigma_1 & 0 \\ 0 & \tau_2\Sigma_2 \end{bmatrix}$ *bounds the joint distribution of* $\mathbf{x}_1$ *and* $\mathbf{x}_2$, *i.e.,* $\mathbf{B} \succeq \Sigma = \begin{bmatrix} \Sigma_1 & E_{12} \\ E_{21} & \Sigma_2 \end{bmatrix}$, *where* $\tau_1 = \frac{1}{\eta - \kappa}$, $\tau_2 = \frac{1}{\eta + \kappa}$, $\kappa^2 \leqslant \frac{1-2\eta}{1-r_{max}^2} + \eta^2$, *and* $0.5 \leqslant \eta \leqslant \frac{1}{1+r_{max}}$.

With this result in hand, we now discuss how to use it to makes the tracking of moments across layers feasible.

How to use Theorem 5.2? By Observation 5.2, we can store *one covariance matrix* over the convolved output pixels at each layer. Notice that due to the presence of the cross-correlation between output pixels, we also need to store cross-correlation matrices, which was our bottleneck! But with the help of Theorem 5.2, we can essentially construct independent convolved outputs, called $\left\{\widehat{\mathbf{h}_i}\right\}$, that bound the covariance of the original convolved outputs, $\{\mathbf{h}_i\}$. To apply this theorem, we need

to estimate the bounding covariance matrix **B**, which can be achieved with the following simple steps (the notations are consistent with Theorem 5.2)

(1) We estimate the bound on correlation coefficient r_{max}

(2) Assign $\eta = \frac{1}{1+r_{max}}$

(3) Assign $\kappa = 0$ which essentially implies $\tau_1 = \tau_2$

Remark: When computing the variance $\Sigma[c_x, c_x]$ in the i^{th} layer, we need only k upper bound of covariances $\widehat{\Sigma_1}^{(i-1)}, \widehat{\Sigma_2}^{(i-1)}, \cdots, \widehat{\Sigma_k}^{(i-1)}$ from the $(i-1)^{th}$ layer. Moreover, using the assumption that the covariance matrices of the $(i-1)^{th}$ layers to be identical across pixels when the input perturbation is identical, we only compute $\widehat{\Sigma}^{(i-1)}$, which in turn requires computing only one upper bound of covariance. *Hence, the computational cost reduces to linear in terms of the depth of the network.*
Ansatz: The assumption of identical pixels (when removing the mean) is sensible when the network is linear. But the assumption is undesirable. So, we will need a mechanism to deal with the nonlinear activation function setting. Further, we will need to design the mechanics of how to track the mean and covariance for different type of layers. We will describe the details next.

Robust Training by Propagating Covariances

Overview. We described simplifying the computation cost by tracking the upper bounds on the perturbation of the independent pixels. We introduce details of an efficient technique to track the covariance of the distribution across different types of layers in a CNN. We will also describe how to deal with nonlinear activation functions.

We treat the i^{th} pixel, after perturbation, as drawn from a Gaussian distribution $\mathbf{x}_i \sim \mathcal{N}(\boldsymbol{\mu}_i, \Sigma)$, where $\boldsymbol{\mu}_i \in \mathbb{R}^N$ and Σ is the covariance matrix across the channels (note that Σ is the same across pixels for the same layer). We may remove the indices to simplify the formulation and avoid clutter. A schematic showing propagating

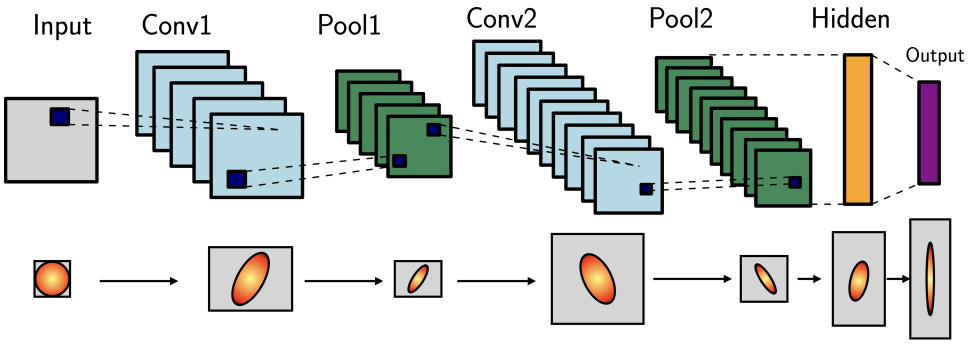

Figure 5.4: The LeNet with tracking the bounding box or the covariance matrices over each layer. The covariance matrices are denoted as the ovals. Since bounding boxes are proportional to $\|W\|_1$, while covariance matrices are proportional to $\|W\|_2$, the covariance-based upper bound will be tighter than the box-base one.

the distribution across LeNet Bengio et al. (2007) model, for simplicity, is shown in Figure 5.4 denoted by the colored ovals.

To propagate the distribution through the whole network, we need a way to propagate the moments through the layers, including commonly used network modules, such as convolution and fully connected layers. Since the batch normalization layer normally has a large Lipschitz constant, we do not include the batch normalization layer in the network. We will introduce the high-level idea, while the low-level details are in the Appendix A.1.

Convolution layer. Since the convolution layer is a linear operator, the covariance of an output pixel $\Sigma_{\mathbf{h}} \in \mathbb{R}^{N_{out} \times N_{out}}$ is defined as $\Sigma_{\mathbf{h}} = W^T \widetilde{\Sigma} W$. Here, let $\widetilde{\mathbf{x}} \in \mathbb{R}^{N_{in} k^2}$ be the vector consisting of all the independent variables inside a $k \times k$ kernel $\{\mathbf{x}_i\}$, $\widetilde{\Sigma} \in \mathbb{R}^{N_{in} k^2 \times N_{in} k^2}$ is the covariance of the concatenated $\widetilde{\mathbf{x}}$. W is the reshaped weight matrix of the shape $N_{in} k^2 \times N_{out}$.

We need to apply Theorem 5.2 to compute the upper bound of $\Sigma_{\mathbf{h}}$ as $\widehat{\Sigma_{\mathbf{h}}} = (1 + r_{max}) W^T \widetilde{\Sigma} W$ to avoid the computational costs of the dependency from cross-correlations. A pictorial description of propagating moments through the convolution layer is shown in Figure 5.5.

First (and other) linear layers. The first linear layer can be viewed as a special case

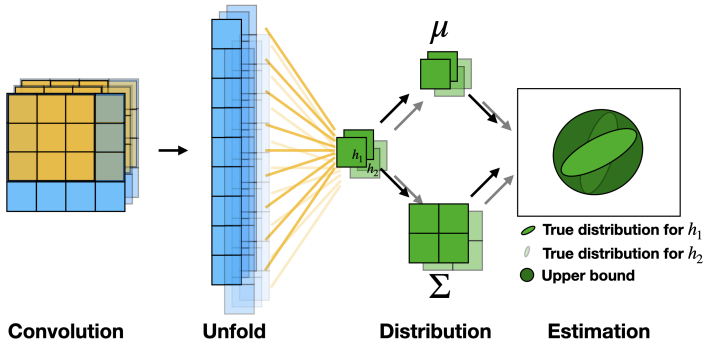

Figure 5.5: The yellow blocks are the kernel of convolution, while the blue blocks are the data. After computing the distribution, we use an upper bound to remove the dependency of two pixels h_1, h_2.

of convolution with kernel size equal to the input spatial dimension. Since there will only be one output neuron $\mathbf{h}$ (with channels), there is no need to break the cross-correlation between neurons. Thus, $\Sigma_{\mathbf{h}} = W^T \Sigma_{\tilde{\mathbf{x}}} W$ and takes a form similar to the convolution layer.

Special case: From Observation 5.1, we only need the largest two intensities to estimate the $\underline{p_{c_x}}$ in the $u_\theta(\mathbf{x})$ layer. Thus, if there is only one linear layer as the last layer in the $u_\theta(\mathbf{x})$, as in most of Resnet like models, this can be further simplified. We only need to consider the covariance matrix between c_x and $\tilde{c}$ index of $u_\theta(\mathbf{x})$. Thus, this will need calculating a 2×2 covariance matrix instead of a $C \times C$ matrix.

On the other hand, if the network consists of multiple linear layers, calculating the moments of the subsequent linear layers must be handled differently. Let $\mathbf{x} \sim \mathcal{N}\left(\boldsymbol{\mu}_{\mathbf{x}}^{(i)}, \Sigma_{\mathbf{x}}^{(i)}\right) \in \mathbb{R}^{N_i}$ be the input of the i^{th} linear layer given by $\mathbf{h} = W_i^T \mathbf{x} + \mathbf{b}_i$, then

$$\mathbf{h} \sim \mathcal{N}\left(W_i^T \boldsymbol{\mu}_{\mathbf{x}}^{(i)} + \mathbf{b}_i, W_i^T \Sigma_{\mathbf{x}}^{(i)} W_i\right).$$

Here, $W_i \in \mathbb{R}^{N_i \times N_{i+1}}$, $\mathbf{b}_i \in \mathbb{R}^{N_{i+1}}$, and $\mathbf{h} \in \mathbb{R}^{N_{i+1}}$.

Pooling layer. Recall that the input of a max pooling layer is $\{\mathbf{x}_i\}$ where each $\mathbf{x}_i \in \mathbb{R}^{N_{in}}$ and the index i varies over the spatial dimension. Observe that as we identify each $\mathbf{x}_i$ by the respective distribution $\mathcal{N}(\boldsymbol{\mu}_i, \Sigma)$, applying max pooling over $\mathbf{x}_i$ essentially requires computing the maximum over $\{\mathcal{N}(\boldsymbol{\mu}_i, \Sigma)\}$, which is not a

well-defined operation. Thus, we restrict ourselves to average pooling. This can be viewed as a special case of the convolution layer with no overlapping and the fixed kernel: $\mathbf{h} \sim \mathcal{N}\left(\frac{1}{k^2}\sum_{\mathbf{x}_i \in \mathbb{W}} \boldsymbol{\mu}_i, \frac{\Sigma}{k^2}\right)$, $\mathbb{W}$ is the kernel window.

Normalization layer. For the normalization layer, given by $\mathbf{h} = (\mathbf{x} - \mu')/\sigma'$, where μ', σ' can be computed in different ways Ioffe and Szegedy (2015); Ba et al. (2016); Ulyanov et al. (2016), we have $\mathbf{h} \sim \mathcal{N}\left((\boldsymbol{\mu}_{\mathbf{x}}^{(i)} - \mu')/\sigma', \Sigma_{\mathbf{x}}^{(i)}/\sigma'^2\right)$. However, as the normalization layers often have large Lipschitz constant Awais et al. (2020), we omit these layers in this work.

Activation layer. This is the final missing piece in efficiently tracking the moments. The overall goal is to find an identical upper bound of the second moments after the activation layer when the input vectors share identical second moments. Also, the first moments should be easier to compute, and ideally, will have a closed form. In Bibi et al. (2018); Lee et al. (2019), the authors introduced a scheme to compute the mean and variance after a ReLU operation. Since ReLU is an element-wise operation, for each element (a scalar), assume $x \sim \mathcal{N}(\mu, \sigma^2)$. After ReLU activation, the first and second moments of the output are given by:

$$\mathbb{E}(\text{ReLU}(x)) = \frac{1}{2}\mu - \frac{1}{2}\mu \,\text{erf}(\frac{-\mu}{\sqrt{2}\sigma}) + \frac{1}{\sqrt{2\pi}}\sigma \exp(-\frac{\mu^2}{2\sigma^2}),$$
$$\text{var}(\text{ReLU}(x)) < \text{var}(x)$$

Here, erf is the Error function. Since we want an identical upper bound of the covariance matrix after ReLU, as well as the closed form of the mean, we use $\text{ReLU}(\mathbf{x}) \sim \mathcal{N}(\boldsymbol{\mu}_a, \Sigma_a)$ where,

$$\boldsymbol{\mu}_a = \frac{1}{2}\boldsymbol{\mu} - \frac{1}{2}\boldsymbol{\mu} \,\text{erf}(\frac{-\boldsymbol{\mu}}{\sqrt{2}\boldsymbol{\sigma}}) + \frac{1}{\sqrt{2\pi}}\boldsymbol{\sigma} \exp(-\frac{\boldsymbol{\mu}^2}{2\boldsymbol{\sigma}^2}),$$
$$\Sigma_a \preceq \Sigma$$

$\boldsymbol{\sigma}$ is the square root of the diagonal elements of Σ, $\boldsymbol{\mu}$ is the mean of the input vector.

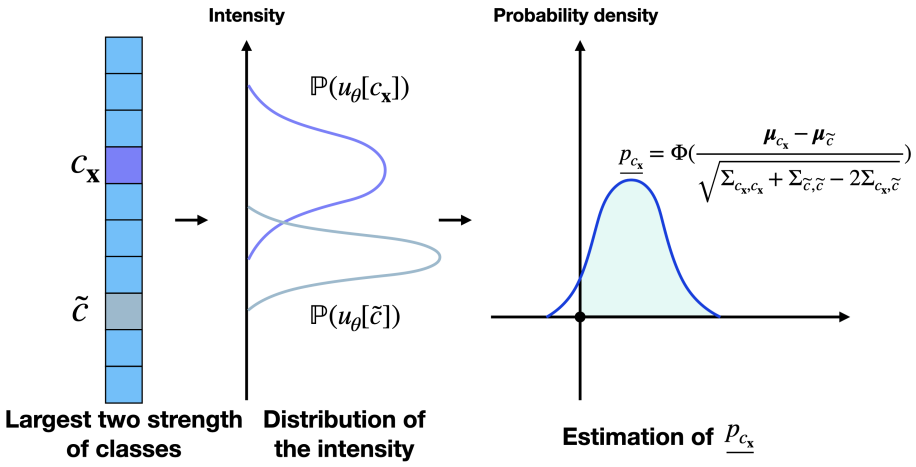

Figure 5.6: In the last layer, we first find the indexes of the largest two intensity $c_x, \widetilde{c}$. Then compute the $\underline{p_{c_x}}$.

All the operators in the first equation are element-wise operators.

Last layer/prediction. The last layer is the layer before softmax layer, which represents the "strength" of the model for a specific class (as shown in Figure 5.6). By Observation 5.1, we have the estimate

$$p_{c_x} = \underline{p_{c_x}} = \Phi\left(\frac{\mu[c_x] - \mu[\widetilde{c}]}{\sqrt{\Sigma[c_x, c_x] + \Sigma[\widetilde{c}, \widetilde{c}] - 2\Sigma[c_x, \widetilde{c}]}}\right)$$

and $p_{\widetilde{c}} = \overline{p_{\widetilde{c}}} = 1 - p_{c_x}$ as an upper bound estimation. By Theorem 5.1, the certified radius is

$$C_R = \frac{\sigma}{2}(\Phi^{-1}(\underline{p_{c_x}}) - \Phi^{-1}(\overline{p_{\widetilde{c}}})) \tag{5.4}$$

$$= \sigma \frac{\mu[c] - \mu[\widetilde{c}]}{\sqrt{\Sigma[c_x, c_x] + \Sigma[\widetilde{c}, \widetilde{c}] - 2\Sigma[c_x, \widetilde{c}]}}. \tag{5.5}$$

Network structures used. In the experiment, we applied two types of network on different dataset, LeNet Bengio et al. (2007) and PreActResnet 18 He et al. (2016b).

Table 5.1: A review of different layers. Here, μ_i, Σ_i is the mean and covariance matrix of the input channels, while μ_o, Σ_o is the mean and covariance after that layer.

	Convolution	Linear	Pooling	Activation
μ_o	$conv(\mu_i, W) + b$	$W^T\mu_i + b$	$\frac{1}{k^2}\sum \mu_i$	$\frac{1}{2}\mu_i - \frac{1}{2}\mu_i \operatorname{erf}(\frac{-\mu_i}{\sqrt{2}\sigma})$ $+\frac{1}{\sqrt{2\pi}\sigma}\exp(-\frac{\mu_i^2}{2\sigma^2})$
Σ_o	$(1 + r_{max})W^T\widetilde{\Sigma_i}W$	$W^T\Sigma_i W$	$\frac{1}{k^2}\Sigma_i$	Σ_i

LeNet requires convolution layer, average pooling layer, activation layer, and linear layer. We build the network with three convolution layers with activation and pooling after each layer, and two linear layers.

The structure of PreActResnet 18 is similar with two major differences – the residual connection and it involves only one linear layer. For the residual connection, it can be viewed as a special type of linear layer. Due to the assumption of independence, the final covariance is the addition of two inputs. Also, there is only one linear layer as the final layer. Thus, we can reduce the cost of computing the whole covariance matrix to only computing the covariance matrix of the largest two intensities.

As discussed above, we removed all batch normalization layers within the network as well as replaced all max pooling operations with the average pooling layer in the network structure. Our experiments suggest that there is minimal impact on performance.

Training Loss

In the spirit of Zhai et al. (2020), the training loss consists of two parts: the classification loss and the robustness loss, i.e., the total loss $l(g_\theta; \mathbf{x}, y) = l_C(g_\theta; \mathbf{x}, y) + \lambda l_{C_R}(g_\theta; \mathbf{x}, y)$. Similar to the literature, we use the softmax layer on the expectation to compute the cross-entropy of the prediction and the true label, given by

$$l_C(g_\theta; \mathbf{x}, y) = y \log(\text{softmax}(\mathbb{E}[u_\theta(\mathbf{x})]))$$

Here, $l_{C_R}(g_\theta; \mathbf{x}, y = c_x)$ is

$$\max(0, \Gamma - \sigma \frac{\boldsymbol{\mu}[c_x] - \boldsymbol{\mu}[\widetilde{c}]}{\sqrt{\Sigma[c_x, c_x] + \Sigma[\widetilde{c}, \widetilde{c}] - 2\Sigma[c_x, \widetilde{c}]}})$$

Thus, minimizing the loss of l_{C_R} is equivalent to maximizing C_R. Γ is the offset to control the certified radius.

5.4 Experiments

In this section, we discuss the applicability and usefulness of our proposed model in two applications namely

(1) image classification tasks to show the performance of our proposed model both in terms of performance and speed

(2) trainability of our model on data with noisy labels

Robust Training

Similar to Cohen et al. (2019), we use the approximate certified test set accuracy as our metric, which is defined as the percentage of test set whose $C_R \geqslant r$. For a fair comparison, we use the Monte Carlo method introduced in Cohen et al. (2019) Section 3.2 to compute C_R here just as our baseline model does. Recall that $C_R = 0$ if the classification is wrong. Otherwise, $C_R = \sigma\Phi^{-1}(\underline{p_A})$ (please refer to the pseudocode in Cohen et al. (2019)). In order to run certification, we used the code provided by Cohen et al. (2019). We also report the average certified radius (ACR), which is defined as $\frac{1}{m}\sum_{i=1}^{m} C_R(\mathbf{x}^i)$ over the test set.

Datasets and baselines. We evaluate our proposed model on five vision datasets: MNIST LeCun (1998), SVHN Netzer et al. (2011), Cifar-10 Krizhevsky and Hinton (2009), ImageNet Deng et al. (2009), and Places365 Zhou et al. (2018). We modify LeNet for MNIST dataset and PreActResnet 18 He et al. (2016b) for SVHN, Cifar-10, ImageNet, and Places365 datasets similarly as in Cohen et al. (2019). Our baseline

model is based on Monte Carlo samples, which requires a large number of samples to make an accurate estimation. *In the rest of the section, we will observe that our model can be at best* $5\times$ *faster than the baseline model. For the larger dataset, since MACER Zhai et al. (2020) uses a reduced number of MC samples, our model is* $1.5\times$ *faster.*

Model hyperparameters. During training, we use a similar strategy as our baseline model. We train the base classifier first and then fine-tune our model considering the aforementioned robust error. We train a total of 200 epochs with the initial learning rate to be 0.01 for MNIST, SVHN, and Cifar-10. The learning rate decays at $100, 150$ epochs respectively. The parameter λ is set to be 0 in the initial training step, and changes to 4.0 at epoch 100 for MNIST, SVHN, and Cifar-10. For ImageNet and Places365 dataset, we train 120 epochs with λ being 0.5 after 30 epochs. The initial learning rate is 0.01 and decays linearly at $30, 60, 90$ epochs.

Results. We report the numerical results in Table 5.3, where, the number reported in each column represents the ratio of the test set with the certified radius larger than the header of that column. Thus, the larger the number is, the better the performance of different models. The ACR is the average of all the certified radius on the test set. Note that the certified radius is 0 when the classification result is wrong. It is noticeable that our model strikes a balance of robustness and the training speed. We achieve $5\times$ speed-up over Zhai et al. (2020), which uses the Monte Carlo method during the training phase, as shown in Figure 5.7. On the other hand, compared with Cohen et al. (2019), our model achieves competitive accuracy and certified radius. We like to point out that although SmoothAdv Salman et al. (2019) is a powerful model, we did not compare with SmoothAdv because MACER Zhai et al. (2020) performs better than SmoothAdv Salman et al. (2019) in terms of ACR and training speed.

Separate from the quantitative performance measures, we also evaluate the validity of Gaussian assumption on the pre-activation vectors within the network. Here, we choose PreActResnet 18 on ImageNet to visualize the first two channels across different layers. The detailed results are shown in Figure 5.8 and Table 5.2. These results not only show that the assumption is reasonable along the neural network, but also demonstrates that our method can estimate the covariance matrix

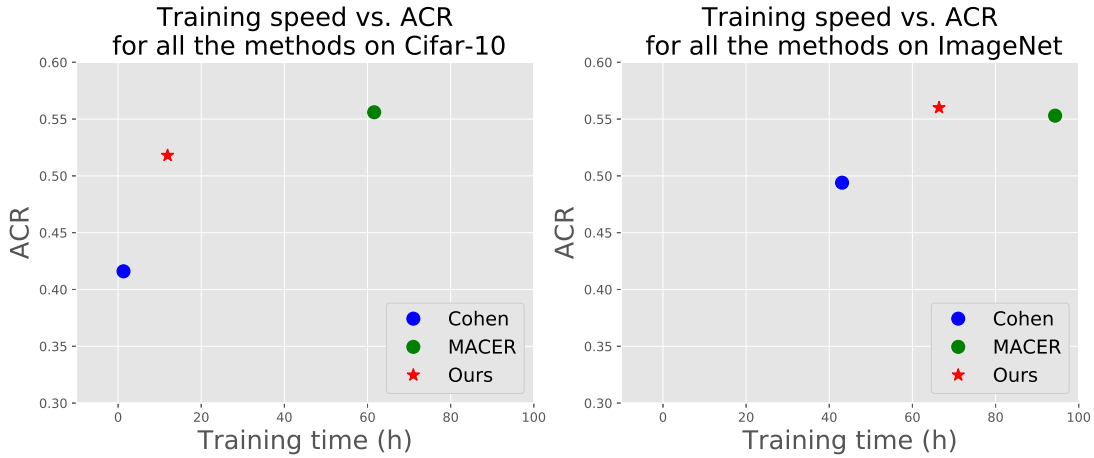

Figure 5.7: The training speed for three models on Cifar-10 and ImageNet dataset, including Cohen et al. (2019), MACER Zhai et al. (2020), and ours.

Table 5.2: Statistics for different layers of MC sampling and our upper bound tracking method.

Layer number	1	5	9	13	17
MC (1000 samples)	0.243	0.913	2.740	2.999	0.712
Upper bound	0.256	1.126	4.069	5.367	1.208

well when the depth of network is moderate.

Ablation study. We perform an ablation study on the choice of the hyperparameters for Places365 dataset. We fix σ of the perturbation to be 0.5. We first test the influence of λ which is the balance between the accuracy (first moments) and the robustness (second moments). Also, to verify the estimation of r_{max}, we tried different r_{max} estimates while fixing $\lambda = 0.5$. Detailed results are shown in Table 5.4.

Discussion. A key benefit of our method is the training time. As shown in Figure 5.7, our method can be $5\times$ faster on Cifar-10, dataset, with a comparable ACR as MACER. For larger datasets, since MACER reduces the number of MC samples in their algorithm, our method is only $1.5\times$ faster with a slightly better ACR than MACER. *Hence, our method is a cheaper substitute of the SOTA with a marginal performance compromise.*

Limitations. There are some limitations due to the simplifications incorporated in our model. When the network is extremely deep, e.g., Resnet 101, the estimation of the second moments tends to be looser as the network grows deeper. Another minor issue is when the input perturbation is large. As observed from Table 5.3, the ACR drops for $\sigma = 1.0$ from $\sigma = 0.5$. The main reason is the assumption that samples are Gaussian distributed. Hence, as the perturbation grows larger, the number of channels, by the central limit theorem, should be much larger to satisfy the Gaussian distribution. Thus, given a fixed network, there is an inherent limitation imposed on the input perturbation. We provide a more detailed discussion in Appendix A.2.

Training with Noisy Labels

As we discussed in Section 5.1, training a robust network has a side effect on smoothing the margin of the decision boundary, which enables training with noisy labels.

Problem statement. Here, we consider a challenging noise setup called "pair flipping", which can be described as follows. When noise rate is p fraction, it means p fraction of the i_{th} labels are flipped to the $(i+1)_{th}$. In this work, we test our method for a high noise rate 0.45.

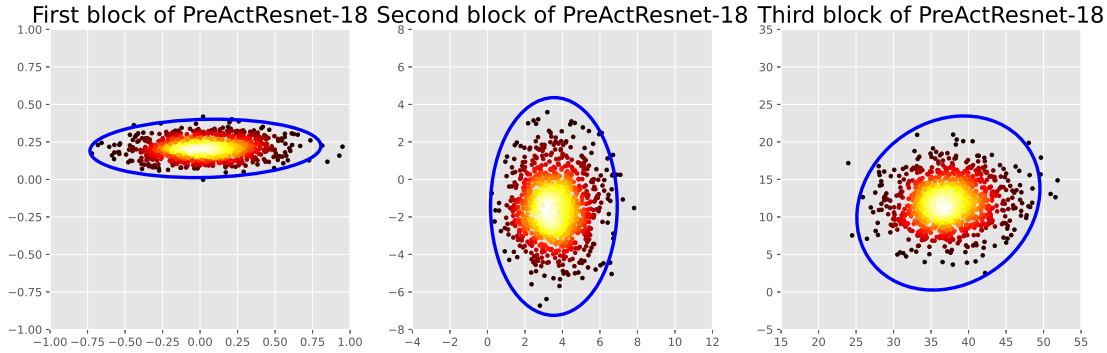

Figure 5.8: A visualization of the first two channels within the neural network across different layers. The dots are the actual MC samples and the color represents the density at that point. The blue oval is generated from the covariance matrices we are tracking.

Table 5.3: Results on MNIST, SVHN, Cifar-10, ImageNet, and Places365 with the certified robustness. The number reported in each column represents the ratio of the test set with the certified radius larger than the header of that column under the perturbation σ. ACR is the average certified radius of all the test samples. A larger value is better for all the numbers reported.

Dataset	σ	Method	0.00	0.25	0.50	0.75	1.00	1.25	1.50	1.75	ACR
MNIST	0.25	Cohen Cohen et al. (2019)	0.99	0.97	0.94	0.89	0	0	0	0	0.887
		MACER Zhai et al. (2020)	0.99	0.99	0.97	0.95	0	0	0	0	**0.918**
		Ours	0.99	0.98	0.96	0.92	0	0	0	0	0.904
	0.50	Cohen Cohen et al. (2019)	0.99	0.97	0.94	0.91	0.84	0.75	0.57	0.33	1.453
		MACER Zhai et al. (2020)	0.99	0.98	0.96	0.94	0.90	0.83	0.73	0.50	**1.583**
		Ours	0.98	0.98	0.95	0.91	0.87	0.77	0.62	0.37	1.485
SVHN	0.25	Cohen Cohen et al. (2019)	0.90	0.70	0.44	0.26	0	0	0	0	0.469
		MACER Zhai et al. (2020)	0.86	0.72	0.56	0.39	0	0	0	0	**0.540**
		Ours	0.89	0.68	0.48	0.36	0	0	0	0	0.509
	0.50	Cohen Cohen et al. (2019)	0.67	0.48	0.37	0.24	0.14	0.08	0.06	0.03	0.434
		MACER Zhai et al. (2020)	0.61	0.53	0.44	0.35	0.24	0.15	0.09	0.04	**0.538**
		Ours	0.67	0.53	0.36	0.29	0.19	0.12	0.07	0.03	0.475
Cifar-10	0.25	Cohen Cohen et al. (2019)	0.75	0.60	0.43	0.26	0	0	0	0	0.416
		MACER Zhai et al. (2020)	0.81	0.71	0.59	0.43	0	0	0	0	**0.556**
		Ours	0.80	0.72	0.55	0.37	0	0	0	0	0.518
	0.50	Cohen Cohen et al. (2019)	0.65	0.54	0.41	0.32	0.23	0.15	0.09	0.04	0.491
		MACER Zhai et al. (2020)	0.66	0.60	0.53	0.46	0.38	0.29	0.19	0.12	**0.726**
		Ours	0.58	0.56	0.43	0.36	0.27	0.15	0.08	0.01	0.543
ImageNet	0.25	Cohen Cohen et al. (2019)	0.58	0.49	0.40	0.29	0	0	0	0	0.379
		MACERZhai et al. (2020)	0.59	0.52	0.43	0.34	0	0	0	0	0.418
		Ours	0.64	0.55	0.44	0.33	0	0	0	0	**0.425**
	0.50	Cohen Cohen et al. (2019)	0.43	0.38	0.34	0.29	0.26	0.22	0.17	0.12	0.494
		MACER Zhai et al. (2020)	0.54	0.47	0.39	0.32	0.29	0.21	0.17	0.11	0.553
		Ours	0.52	0.47	0.39	0.32	0.28	0.23	0.18	0.13	**0.560**
	1.00	Cohen Cohen et al. (2019)	0.21	0.19	0.18	0.16	0.15	0.13	0.11	0.09	0.345
		MACER Zhai et al. (2020)	0.37	0.33	0.30	0.26	0.22	0.19	0.15	0.12	0.517
		Ours	0.38	0.33	0.29	0.26	0.22	0.19	0.15	0.11	**0.519**
Places365	0.25	Cohen Cohen et al. (2019)	0.45	0.42	0.36	0.29	0	0	0	0	0.340
		MACERZhai et al. (2020)	0.46	0.44	0.39	0.30	0	0	0	0	0.359
		Ours	0.50	0.46	0.40	0.33	0	0	0	0	**0.380**
	0.50	Cohen Cohen et al. (2019)	0.43	0.38	0.35	0.28	0.23	0.19	0.17	0.12	0.484
		MACER Zhai et al. (2020)	0.45	0.42	0.37	0.31	0.26	0.22	0.18	0.13	0.533
		Ours	0.46	0.43	0.39	0.35	0.31	0.28	0.23	0.16	**0.597**
	1.00	Cohen Cohen et al. (2019)	0.20	0.18	0.16	0.15	0.13	0.12	0.11	0.10	0.357
		MACER Zhai et al. (2020)	0.31	0.29	0.28	0.25	0.22	0.21	0.19	0.17	0.615
		Ours	0.32	0.30	0.29	0.26	0.24	0.21	0.19	0.16	**0.622**

Dataset. The dataset we considered for this analysis is Cifar-10. To generate noisy labels from the clean labels of the dataset, we stochastically changed p fraction of the labels using the source code provided by Han et al. (2018). We perform a comparative analysis of our method with Bootstrap Reed et al. (2015), S-model Goldberger

Table 5.4: Ablation experiment on Places365 with $\sigma = 0.5$. We perform the choice of λ and r_{max} as the hyper-parameters.

Parameters	Value	0.00	0.25	0.50	0.75	1.00	1.25	1.50	1.75	ACR
λ	0.0	0.43	0.38	0.35	0.28	0.23	0.19	0.17	0.12	0.484
	0.5	0.47	0.44	0.39	0.34	0.29	0.23	0.19	0.14	**0.565**
	1.0	0.44	0.41	0.34	0.30	0.28	0.23	0.20	0.14	0.530
r_{max}	0.0	0.43	0.38	0.36	0.31	0.27	0.21	0.16	0.13	0.509
	0.1	0.47	0.44	0.39	0.34	0.29	0.23	0.19	0.14	0.565
	0.2	0.46	0.43	0.39	0.35	0.31	0.28	0.23	0.16	**0.597**
	0.3	0.46	0.44	0.41	0.35	0.29	0.23	0.19	0.15	0.573
	0.4	0.44	0.40	0.36	0.31	0.27	0.23	0.17	0.12	0.520

and Ben-Reuven (2017), Decoupling Malach and Shalev-Shwartz (2017), MentorNet Jiang et al. (2018), Co-teaching Han et al. (2018), and Trunc $\mathcal{L}_q$Zhang and Sabuncu (2018).

Model hyperparameters. Similar to training robust network with the clean labels, we first treat the noisy labels as "clean" to train our model. After 60 epochs, we remove the classification loss for the data with top 10% C_R to fine-tune the network. The initial learning rate is set to 0.01 and decays at $30, 60, 90$ epochs, respectively.

Results. The results are shown in Table 5.5, where, it is noticeable that even under this strong label corruption, our model outperforms most baseline results as well as stays stable over different epochs.

Table 5.5: Average test accuracy on pair-flipping with noise rate 45% for last 10 epochs. Results of BootstrapReed et al. (2015), S-modelGoldberger and Ben-Reuven (2017), DecouplingMalach and Shalev-Shwartz (2017), MentorNetJiang et al. (2018), Co-teachingHan et al. (2018), Trunc $\mathcal{L}_q$Zhang and Sabuncu (2018), and Ours.

Method	Bootstrap	S-model	Decoupling	MentorNet	Co-teaching	Trunc $\mathcal{L}_q$	Ours
mean	0.501	0.482	0.488	0.581	0.726	**0.828**	0.808
std	$3.0e-3$	$5.5e-3$	$0.4e-3$	$3.8e-3$	$1.5e-3$	$6.7e-3$	$0.2e-3$

5.5 Conclusions

Developing mechanisms that enable training certifiably robust neural networks nicely complements the rapidly evolving body of literature on adversarial training. While certification schemes, in general, have typically been limited to small sized networks, recent proposals related to randomized smoothing have led to a significant expansion of the type of models where these ideas can be used. Our proposal here takes this line of work forward and shows that, with extension from Euclidean data representation to manifold-valued data representation, bound propagation ideas together with some meaningful approximations can provide an efficient method to maximize the certified radius – a measure of robustness of the model. We show that the strategy achieves competitive results to other baselines with faster training speed. We also investigate a potential use case for training with noisy labels where the behavior of such ideas has not been investigated, but appears to be promising. While this work uses maximum perturbation at the input layer, these ideas may have some use within formulations where perturbations are the feature-level internal to the network, see Meng et al. (2021b). The code for the algorithms described in this chapter is available at `https://github.com/zhenxingjian/Propagating_Covariance`.

6 ON THE VERSATILE USES OF PARTIAL DISTANCE CORRELATION IN DEEP LEARNING

In Chapters 3 to 5, we mainly operated with manifold-valued data such as SPD matrices. We discussed why the metric (distance) induced by the geometric information of such space is important, and how the manifold operations such as wFM help. In this chapter, we will study the concept of distance. The description builds upon the brief introduction in Section 2.2.

Comparing the functional behavior of neural network models, whether it is a single network over time or two (or more networks) during or post-training, is an essential step in understanding what they are learning (and what they are not), and for identifying strategies for regularization or efficiency improvements. Despite recent progress, e.g., comparing vision transformers to CNNs, systematic comparison of function, especially across different networks, remains difficult and is often carried out layer by layer. Approaches such as canonical correlation analysis (CCA) are applicable in principle, but have been sparingly used so far. In this chapter, we revisit a (less widely known) from statistics in Section 6.2, called distance correlation (and its partial variant), designed to evaluate correlation between feature spaces of different dimensions. We describe the steps necessary to carry out its deployment for large scale models – this opens the door to a surprising array of applications ranging from optimizing diverse models that would directly be more robust to adversarial attacks, learning disentangled representations as well as conditioning one deep model w.r.t. another. We will discuss each application in detail in Sections 6.4, 6.5, and 6.6 respectively. Our experiments suggest a versatile regularizer (or constraint) with many advantages, which avoids some of the common difficulties one faces in such analyses. While distance correlation has also been shown to work for manifold-valued data Pan et al. (2017), this general setting will not be immediately needed for the tasks tackled here, but could be interesting to pursue in the future. The work presented in this chapter appeared as a conference paper in ECCV 2022 Zhen et al. (2022).

6.1 Introduction

Consider the following hypothetical scenario. Let us say that a fully functional *computational* model of the visual system – perhaps a modern version of the Neocognitron Fukushima et al. (1983) – was somehow provided to us. And we wished to "compare" its behavior to modern CNN models Iandola et al. (2014); He et al. (2016a). To do so, two options appear sensible. The first – inspired by analogies between computational vision and biological vision – would draw a correspondence between how simple/complex cells in the visual cortex process scenes and their induced receptive fields with those of activations of units/blocks in a modern deep neural network architecture Selvaraju et al. (2017). While this process is often difficult to carry out systematically, it is powerful and, in some ways, has contributed to interest in biologically inspired deep learning, see Wozniak et al. (2020). Updated forms of this intuition – associating different subsets of cells (or neural network units) to different semantic/visual concepts – remains the default approach we use in debugging and interpretation. The second option for tackling the hypothetical setting above is to pose it in an information theoretic setting. That is, for two models Θ_X and Θ_Y, we ask the following question: *what has* Θ_X *learned that* Θ_Y *has not? Or vice versa*. The asymmetry is intentional because if we consider two random variables (r.v.) X, Y, the question simply takes the form of "conditioning", i.e., compare $\mathbb{P}(X)$ versus $\mathbb{P}(X|Y)$. This form suffices if our interest is restricted to the *predictions* of the two models. If we instead wish to capture the model's behavior more globally – when X and Y denote the full set of feature responses – we can use divergence measures on high dimensional probability measures given by the two models (Θ_X and Θ_Y) responses on the training samples. Importantly, notice that our description assumes that, at least, the probability measures are defined on the same domain.

More general use cases. While the above discussion was cast as comparing two networks, it is representative of a broad basket of tasks in deep learning. (**a**) Consider the problem of learning fair representations Zemel et al. (2013); Feldman et al. (2015); Zafar et al. (2017); Lokhande et al. (2020a) where the model must

be invariant to one (or more) sensitive attributes. We seek latent representations, say $\Psi_{\mathrm{pred}}(X)$ for the prediction task, which minimizes mutual information w.r.t. the latent representation relevant for predicting the sensitive attribute $\Psi_{\mathrm{sens}}(X)$. Indeed, if information regarding the sensitive attribute is partially preserved or leaks into $\Psi_{\mathrm{pred}}(X)$, the relative entropy will be low Moyer et al. (2018). Observe that this calculation is possible partly because the latent space specifies the *same probability space* for the two distributions. (**b**) The setting is identical in common approaches for learning disentangled representations, where disentanglement is measured via various information theoretic measures Chen et al. (2018); Achille and Soatto (2018); Gabbay et al. (2021); Shu et al. (2020). If we now segue back to comparing two different networks, but without the convenience of a common coordinate system to measure divergence, the options turn out to be limited. (**c**) Recently, in trying to understand whether vision Transformers "see" similar to convolutional neural networks Raghu et al. (2021), one option utilized recently was a kernel-based representation similarity, in a layer-by-layer manner. What we may actually want is a mechanism for conditioning – for example, if one of the models is thought of a "nuisance variable", we wish to check the residual in the other after the first has been controlled for (or marginalized out). Importantly, this should be possible without assuming that the probability distributions live in the same space (or networks Θ_X and Θ_Y are the same).

A direct application of CCA? As we discussed in Section 2.2, CCA is able to deal with the different dimensionality of two random variables. Consider two different feature spaces ($\mathcal{X}$ and $\mathcal{Y}$), say in dimensions $\mathbb{R}^n$ and $\mathbb{R}^m$, pertaining to feature activations from two different models. Comparison of these two feature spaces *is* possible. One natural choice is canonical correlation analysis (CCA) Bach and Jordan (2005), a generalization of correlation, specifically suited when $n \neq m$. The idea has been utilized for studying representation similarity in deep neural network models Morcos et al. (2018), albeit in a post-training setting for reasons that will be clear shortly, as well as for identifying more efficient training regimes (i.e., can lower layers be sequentially frozen after a certain number of timesteps). CCA has also been shown to be implementable within DNN pipelines for multi-

view training, called DeepCCA Andrew et al. (2013), although efficiency can be a bottleneck limiting its broader deployment. A stochastic version of CCA suitable for DNN training with mini-batches has been proposed very recently, and strong experimental evidence was presented Meng et al. (2021a), also see Gemp et al. (2022). Given that a stochastic CCA is now available, its extensions to the partial CCA setting are not yet available. If successful, this may eventually provide a scheme, suitable for deep learning, for controlling the influence of one model (or a set of variables) with respect to another model.

This work. The starting point of this work is a less widely used statistical concept to measure the correlation between two different feature spaces $(\mathcal{X}, \mathcal{Y})$ of *different dimensions*, called distance correlation (and the method of dissimilarities). In shallow settings, CCA and distance correlation offers very similar functionality – for the most part, they can be used interchangeably although distance correlation would *also need* specification of distances (or dissimilarities). In other words, CCA may be easier to deploy. On the other hand, deep variants of CCA involve specialized algorithms Andrew et al. (2013); Meng et al. (2021a). Further, deep versions of partial CCA have not been reported. In contrast, as long as feature distances *can* be calculated, the differences between the shallow and the deep versions of distance correlation are minimal at best, and adjustments needed are quite minor. These advantages carry over to partial distance correlation, directly enabling conditioning one model w.r.t. another (or using such a term as a regularizer). The main **contribution** of this chapter is to study distance correlation (and partial distance correlation) as a powerful measure in a broad suite of tasks in vision. We review the relevant technical steps which enable its instantiation in deep learning settings and show its broad applications ranging from learning disentangled representations to understanding the differences between what two (or more) networks are learning to training "mutually distinct" deep models (akin to earlier works on M best solutions to MAP estimation in graphical models Fromer and Globerson (2009); Batra et al. (2012)) or training M diverse models for foreground-background segmentation as well as other tasks Guzmán-Rivera et al. (2014).

Related Works

Four distinct lines of work are related to our development, which we review next.
Similarity between networks. Understanding the similarity between different networks is an active topic Kornblith et al. (2019b); Geirhos et al. (2019); Neyshabur et al. (2020) also relevant in adversarial models Demontis et al. (2019); Cheng et al. (2019). Early attempts to compare neural network representations were approached via linear regression Ramsay et al. (1984), whose applicability to nonlinear models is limited. As noted above, canonical correlation analysis (CCA) Anderson (1958); Hotelling (1936) is a suitable off-the-shelf method for model comparisons. To this end, singular vector CCA (SVCCA) Raghu et al. (2017), Projection-Weighted CCA Morcos et al. (2018), DeepCCA Andrew et al. (2013), and stochastic CCA Gao et al. (2019) are all potentially useful. Recently, Kornblith et al. (2019a) studied the invariance properties for a good similarity measurement and proposed the centered kernel alignment (CKA). CKA offers invariance to invertible linear transformations, orthogonal transformations, and isotropic scaling. Separately, Nguyen et al. (2021); Raghu et al. (2021) used CKA to study similarities between deep and wide neural networks and also between different network structures.
Information theoretic divergence measures. Another body of related work pertains to approximately measuring the mutual information Cover and Thomas (2012) to remove this information, mainly in the context of fair representation learning. Here, mutual information (MI) is measured between features and the sensitive attribute Moyer et al. (2018). In Song et al. (2019), another information theoretic bound for learning maximally expressive representations subject to the given attributes is presented. In Cho et al. (2020), MI between prediction and the sensitive attributes is used to train a fair classifier whereas Akash et al. (2021) describes the use of inverse contrastive loss. Group-theoretic approaches have also been described in Cohen and Welling (2016a); Lokhande et al. (2022). The work in Lample et al. (2017) gives an empirical solution to remove specific visual features from the latent variables using adversarial training.
Repulsion/diversity. If we consider the ensemble of neural networks, there are

several different strategies to maintain functional diversity between ensemble members – we acknowledge these results here because they are loosely related to one of the use cases we evaluate later. SVGD D'Angelo et al. (2021) shows the benefits of choosing the kernel to measure the similarity between ensemble members. In D'Angelo and Fortuin (2021), the authors introduce a kernelized repulsive term in the training loss, which endows deep ensembles with Bayesian convergence properties. The so-called quality diversity (QD) is interesting: Pugh et al. (2016) tries to maximize a given objective function with diversity to a set of pre-defined measure functions Gaier et al. (2020); Rakicevic et al. (2021). When both the objective and measure functions in QD are differentiable, Fontaine and Nikolaidis (2021) offers an efficient way to explore the latent space of the objective w.r.t. the measure functions.

Distance correlation (DC). The central idea of this chapter is distance correlation, which was introduced in Székely et al. (2007) and has been used to analyze nonlinear dependence in time-series Zhou (2012) and ultra high-dimensional data analysis tasks Li et al. (2012). Recent research has proposed new measures, such as conditional local distance correlation (cLDC) Pan et al. (2017) and distance covariance analysis (DCA) Cowley et al. (2017), that capture both linear and nonlinear dependencies and consider the local structure of the data. Additionally, Liu et al. (2021) applies distance correlation to measure the disentanglement of content and style in natural language text using deep learning models. A detailed review of the concept of distance correlation will be presented later in this chapter.

6.2 Review: Distance (and Partial Distance) Correlation

We discussed different types of correlation, especially distance correlation in Section 2.2. In this section, we will introduce the partial distance correlation which can be useful when computing the correlation between two random variables with consideration of the third random variable. To start, let us review distance

correlation.

Given two random variables $X, Y \in \mathbb{R}$ (in the same domain), correlation (say, the Pearson correlation) helps measure their association. One can derive meaningful conclusions by statistical testing. As noted in Section 6.1, one generalization of correlation to a higher dimension is CCA, which seeks to find projection matrices such that correlation among the projected data is maximized, see Bach and Jordan (2005).

Benefits of distance correlation. In many applications, the notion of distances or dissimilarities appears quite naturally. Motivated by the need for a scheme that can capture both linear and non-linear correlations when provided with such dissimilarity information, in Székely et al. (2007), the authors proposed a new measure of dependence between random vectors, called **distance correlation**. The key benefits of distance correlation are:

(1) The distance correlation $\mathcal{R}$ satisfies $0 \leqslant \mathcal{R} \leqslant 1$, and $\mathcal{R} = 0$ if and only if X, Y are independent.

(2) $\mathcal{R}(X, Y)$ is defined for X and Y in **arbitrary dimensions**, e.g., $\mathcal{R}(X, Y)$ is well-defined when X is of dimension p while Y is of dimension q for $p \neq q$.

We focus on empirical distance correlation for n samples drawn from the unknown joint distribution, and review its calculation.

For an observed random sample $(x, y) = \{(X_i, Y_i) : i = 1, \cdots, n\}$ from the joint distribution of random vectors X in $\mathbb{R}^p$ and Y in $\mathbb{R}^q$, define:

$$a_{k,l} = \|X_k - X_l\|, \quad \bar{a}_{k,\cdot} = \frac{1}{n}\sum_{l=1}^{n} a_{k,l}, \quad \bar{a}_{\cdot,l} = \frac{1}{n}\sum_{k=1}^{n} a_{k,l},$$
$$\bar{a}_{\cdot,\cdot} = \frac{1}{n^2}\sum_{k,l=1}^{n} a_{k,l}, \quad A_{k,l} = a_{k,l} - \bar{a}_{k,\cdot} - \bar{a}_{\cdot,l} + \bar{a}_{\cdot,\cdot} \tag{6.1}$$

where $k, l \in \{1, \cdots, n\}$. Similarly, we can define $b_{k,l} = \|Y_k - Y_l\|$, and $B_{k,l} = b_{k,l} - \bar{b}_{k,\cdot} - \bar{b}_{\cdot,l} + \bar{b}_{\cdot,\cdot}$, and based on these quantities we have.

Definition 6.1. (*Distance correlation*) *Székely et al. (2007). The empirical distance correlation* $\mathcal{R}_n(x, y)$ *is the square root of*

$$\mathcal{R}_n^2(x, y) = \begin{cases} \frac{\mathcal{V}_n^2(x,y)}{\sqrt{\mathcal{V}_n^2(x,x)\mathcal{V}_n^2(y,y)}} & , \mathcal{V}_n^2(x, x)\mathcal{V}_n^2(y, y) > 0 \\ 0 & , \mathcal{V}_n^2(x, x)\mathcal{V}_n^2(y, y) = 0 \end{cases} \tag{6.2}$$

where the empirical distance covariance (variance) $\mathcal{V}_n(x, y), \mathcal{V}_n(x, x)$ *are defined as* $\mathcal{V}_n^2(x, y) = \frac{1}{n^2}\sum_{k,l=1}^{n} A_{k,l}B_{k,l}, \mathcal{V}_n^2(x, x) = \frac{1}{n^2}\sum_{k,l=1}^{n} A_{k,l}^2$, *with* A *in* (6.1).

Examples. We show a few simple 2D examples to contrast Pearson Correlation and Distance Correlation in Figure 6.1. Notice that if the relationship between the two random variables is not linear, Pearson Correlation might be small while Distance Correlation remains meaningful.

Extensions to conditioning. Given three random variables X, Y, and Z, we want to measure the correlation between X and Y but "controlling for" Z (thinking of it as a nuisance variable), i.e., we want to estimate $\mathcal{R}(X|Z, Y|Z) = \mathcal{R}^*(X, Y; Z)$. Such a quantity is key in existing approaches in disentangled learning, deriving invariant representations and understanding what one or more networks are learning after concepts learned by another network have been accounted for. Consider how this task would be accomplished in linear regression. We would project X and Y into the space of Z, and only use the residuals to measure the correlation. Nonetheless,

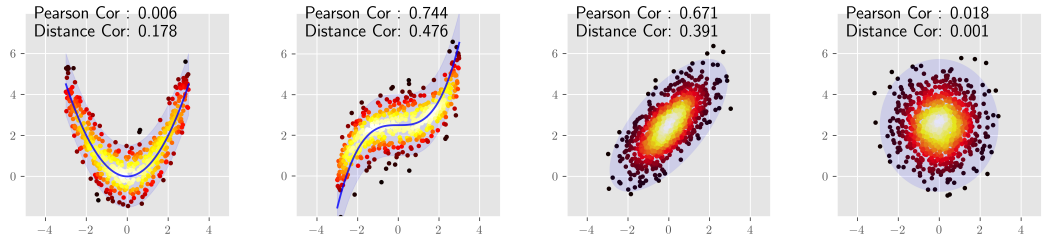

Figure 6.1: Examples of Pearson Correlation and Distance Correlation in different settings. (a): $y = 0.5x^2 + 0.75n, n \sim \mathcal{N}(0, 1)$; (b): $y = 0.15x^3 + 0.75n + 2.5, n \sim \mathcal{N}(0, 1)$; (c): $\begin{bmatrix} x \\ y \end{bmatrix} \sim \mathcal{N}\left(\begin{bmatrix} 0 \\ 2.5 \end{bmatrix}, \begin{bmatrix} 1 & 0.75 \\ 0.75 & 1.25 \end{bmatrix}\right)$; (d): $\begin{bmatrix} x \\ y \end{bmatrix} \sim \mathcal{N}\left(\begin{bmatrix} 0 \\ 2.5 \end{bmatrix}, \begin{bmatrix} 1 & 0 \\ 0 & 1.25 \end{bmatrix}\right)$

defining partial distance correlation is more involved – in Székely and Rizzo (2014), the authors introduced a new Hilbert space where we can define the projection of distance matrix. To do so, the authors calculate a $\mathcal{U}$-centered matrix $\tilde{A}$ from the distance matrix $(a_{k,l})$ so that the inner product of the $\mathcal{U}$-centered matrices will be the distance covariance.

Definition 6.2. *Let $A = (a_{k,l})$ be a symmetric, real valued $n \times n$ matrix $(n > 2)$ with zero diagonal. Define the $\mathcal{U}$-centered matrix $\tilde{A} = (\tilde{a}_{k,l})$ as follows.*

$$\tilde{a}_{k,l} = \begin{cases} a_{k,l} - \frac{1}{n-2}\sum_{i=1}^{n} a_{i,l} - \frac{1}{n-2}\sum_{j=1}^{n} a_{k,j} + \frac{1}{(n-1)(n-2)}\sum_{i,j=1}^{n} a_{i,j} & , k \neq l \\ 0 & , k = l \end{cases} \tag{6.3}$$

Further, the inner product between $\tilde{A}, \tilde{B}$ is defined as $(\tilde{A} \cdot \tilde{B}) := \frac{1}{n(n-3)}\sum_{k \neq l} \tilde{A}_{k,l}\tilde{B}_{k,l}$, and is an unbiased estimator of squared population distance covariance $\mathcal{V}^2(x, y)$.

Before defining partial distance covariance formally, we recall the definition of orthogonal projection on these matrices.

Definition 6.3. *Let $\tilde{A}, \tilde{B}, \tilde{C}$ corresponding to samples x, y, z respectively, and let*

$$P_{z^\perp}(x) = \tilde{A} - \frac{(\tilde{A} \cdot \tilde{C})}{(\tilde{C} \cdot \tilde{C})}\tilde{C}$$
$$P_{z^\perp}(y) = \tilde{B} - \frac{(\tilde{B} \cdot \tilde{C})}{(\tilde{C} \cdot \tilde{C})}\tilde{C}$$

denote the orthogonal projection of $\tilde{A}(x)$ onto $(\tilde{C}(z))^\perp$ and the orthogonal projection of $\tilde{B}(y)$ onto $(\tilde{C}(z))^\perp$.

Now, we are ready to define the partial distance covariance and the partial distance correlation.

Definition 6.4. *Let (x,y,z) be a random sample observed from the joint distribution of (X, Y, Z). The sample partial distance covariance is defined by:*

$$pdCov(x, y; z) = (P_{z^\perp}(x) \cdot P_{z^\perp}(y)) = \frac{1}{n(n-3)} \sum_{i \neq j} (P_{z^\perp}(x))_{i,j} (P_{z^\perp}(y))_{i,j} \qquad (6.4)$$

And the partial distance correlation is defined as:

$$\mathcal{R}^{*2}(x, y; z) := \frac{(P_{z^\perp}(x) \cdot P_{z^\perp}(y))}{\|P_{z^\perp}(x)\| \|P_{z^\perp}(y)\|}$$

where $\|P_{z^\perp}(x)\| = (P_{z^\perp}(x) \cdot P_{z^\perp}(x))^{1/2}$ is the norm.

Partial distance correlation enables asking various interesting questions. By projecting the original $\mathcal{U}$-centered matrix $\tilde{A}$ onto $\tilde{C}$, the correlation between the residual and $\tilde{B}$ will be a measure of what does X learn that Z does not.

6.3 Optimizing Distance Correlation in Neural Networks

While distance correlation can be implemented in a differentiable way, and thereby used as an appropriate loss function in a neural network, we must take efficiency into account. For two p dimensional random variables, let the number of samples for the empirical estimate of DC be n. Observe that the total cost for computing $(a_{k,l})$ is $O(n^2 p)$, and the memory to store the intermediate matrices is also $O(n^2)$. So, we use a stochastic estimate of DC by averaging over minibatches, with each minibatch containing m samples. We describe why this approximation is sensible.
Notation. We use Θ_X, Θ_Y to denote the parameters of the neural networks, and X, Y as features extracted by the respective neural networks. Let the minibatch size be m, and the dataset $\mathcal{D} = (\mathcal{D}_x, \mathcal{D}_y)$ be of size n. We use $(x_t, y_t)_{t=1}^T, x_t \subset \mathcal{D}_x, y_t \subset \mathcal{D}_y$ to represent the data samples at step t, T is the total number of training steps. The distance matrices A_t, B_t are computed when given X_t, Y_t using (6.1), which is of

dimension $m \times m$ for each minibatch. Further, we use $(X_t)_k$ to represent the k^{th} element in X_t. And $(A_t)_{k,l}$ is the k^{th} row and l^{th} column element in the matrix A_t. The inner-product between two matrices A, B is defined as $\langle A, B\rangle = \sum_{i,j}^{m}(A)_{i,j}(B)_{i,j}$.

Objective function. Consider the case where we minimize DC between two networks Θ_X, Θ_Y. Since the parameters between Θ_X, Θ_Y are separable, we can use the block stochastic gradient iteration in Xu and Yin (2015) with some simple modifications.

To minimize the distance correlation, we need to solve the following problem

$$\min_{\Theta_X,\Theta_Y} \frac{\langle A(\Theta_X; x), B(\Theta_Y; y)\rangle}{\sqrt{\langle A(\Theta_X; x), A(\Theta_X; x)\rangle\langle B(\Theta_Y; y), B(\Theta_Y; y)\rangle}} \tag{6.5}$$
$$(A)_{k,l} = \|(X)_k - (X)_l\|_2,\ X = \Theta_X(x), (B)_{k,l} = \|(Y)_k - (Y)_l\|_2,\ \ Y = \Theta_Y(y)$$

We slightly abuse the notation of $\Theta_X(x)$ as applying the network Θ_X onto data x, and reuse A to simplify the notation $A(\Theta_X; x)$ and the distance matrix. We can rewrite the expression (with A, B defined above) using:

$$\min_{\Theta_X,\Theta_Y} \langle A, B\rangle \ \text{ s.t. } \ \max_{x\subset\mathcal{D}_X}\langle A, A\rangle \leqslant m;\ \ \max_{y\subset\mathcal{D}_y}\langle B, B\rangle \leqslant m \tag{6.6}$$

where (x, y) are the minibatch of samples from the data space $(\mathcal{D}_X, \mathcal{D}_y)$.

We can rewrite the above into the following equation similar to (1) in Xu and Yin (2015).

$$\min_{\Theta_X,\Theta_Y} \Phi(\Theta_X, \Theta_Y) = \mathbb{E}_{x,y} f(\Theta_X, \Theta_Y; x, y) + \gamma(\Theta_X) + \gamma(\Theta_Y) \tag{6.7}$$

where $f(\Theta_X, \Theta_Y; x, y)$ is $\langle A, B\rangle$ and $\gamma(\Theta_X)$ encodes the convex constraint of network Θ_X: $\max_{x\subset\mathcal{D}_X}\langle A, A\rangle \leqslant m$. Similarly, $\gamma(\Theta_Y)$ encodes $\max_{y\subset\mathcal{D}_y}\langle B, B\rangle \leqslant m$. $\Phi(\Theta_X, \Theta_Y)$ is the constrained objective function to be optimized.

Block stochastic gradient iteration. We adjust Algorithm 1 from Xu and Yin (2015) to our case in Algorithm 6.1. Since we will need the entire minibatch (x_t, y_t) to compute the objective function, there will be no mean term when computing the

sample gradient $\tilde{\mathbf{g}}_X^t$. Further, since both blocks (Θ_X, Θ_Y) are constrained, line 3, 5 will use (5) from Xu and Yin (2015). The detailed algorithm is presented in Algorithm 6.1.

Algorithm 6.1: Block Stochastic Gradient for Updating Distance Correlation

Input: Two neural network with starting point Θ_X^1, Θ_Y^1. Training data $\{(x_t, y_t)\}_{t=1}^T$, step size η_X, η_Y, and batch size m.

Output: $\tilde{\Theta}_X^T, \tilde{\Theta}_Y^T$

1: **for** $t = 1, \cdots, T$ **do**
2: Compute sample gradient for Θ_X
$\tilde{\mathbf{g}}_X^t = \nabla_{\Theta_X} f(\Theta_X^t, \Theta_Y^t; x_t, y_t)$
3: $\Theta_X^{t+1} = \arg\min_{\Theta_X} \langle \tilde{\mathbf{g}}_X^t + \tilde{\nabla}\gamma_X(\Theta_X^t), \Theta_X - \Theta_X^t \rangle + \frac{1}{2\eta_X}\|\Theta_X - \Theta_X^t\|^2$
4: Compute sample gradient for Θ_Y
$\tilde{\mathbf{g}}_Y^t = \nabla_{\Theta_Y} f(\Theta_X^{t+1}, \Theta_Y^t; x_t, y_t)$
5: $\Theta_Y^{t+1} = \arg\min_{\Theta_Y} \langle \tilde{\mathbf{g}}_Y^t + \tilde{\nabla}\gamma_Y(\Theta_Y^t), \Theta_Y - \Theta_Y^t \rangle + \frac{1}{2\eta_Y}\|\Theta_Y - \Theta_Y^t\|^2$
6: **end for**
7: $\tilde{\Theta}_X^T = \frac{1}{T}\sum_{t=1}^T \Theta_X^t$
8: $\tilde{\Theta}_Y^T = \frac{1}{T}\sum_{t=1}^T \Theta_Y^t$

Proposition 6.5. *After* T *iterations of Algorithm 6.1 with step size* $\eta_X = \eta_Y = \frac{\eta}{\sqrt{T}} < \frac{1}{L}$, *for some positive constant* $\eta < \frac{1}{L}$, *where* L *is the Lipschitz constant of the partial gradient of* f, *by Theorem 6 in Xu and Yin (2015), we know there exists an index subsequence* $\mathcal{T}$ *such that:*

$$\lim_{t\to\infty, t\in\mathcal{T}} \mathbb{E}[dist(\mathbf{0}, \nabla\Phi(\Theta_X^t, \Theta_Y^t))] = 0 \tag{6.8}$$

where $dist(\mathbf{y}, \mathcal{X}) = \min_{\mathbf{x}\in\mathcal{X}} \|\mathbf{x} - \mathbf{y}\|$.

But empirically, we find that simply applying Stochastic Gradient Decent (SGD) is sufficient, but this choice is available to the user.

6.4 Independent Features Help Robustness

Goal. We show how distance correlation can help us train multiple deep networks that learn **mutually independent** features, roughly similar to finding diverse M-best solutions in structured SVM models Schiegg et al. (2016). We describe how such an approach can lead to better robustness against adversarial attacks.

Rationale. Recently, several efforts have explored generating of adversarial examples that can transfer to different networks and how to defend against such attacks Demontis et al. (2019); Shumailov et al. (2019); Chan et al. (2020). It is often observed that an adversarial sample for one trained network is relatively easy to transfer to another network with the same architecture Demontis et al. (2019). Here, we show that even for as few as two networks (same architecture; trained on the same data), we can, to some extent, prevent adversarial examples from transferring between them by seeking independent features. Xu et al. (2022) shows a similar discussion that the use of the orthogonal classifier leads to various benefits, such as controlled style transfer, enhanced alignment methods for domain adaptation, and a reduced degree of unfairness.

Setup. We formulate the problem considering a classification task as an example. Given two deep neural networks with the same architecture denoted as $f_1(\cdot), f_2(\cdot)$, we train them using image-label pairs (x, y) using the cross-entropy loss Loss_{CE}. If we train f_1 and f_2 using only the cross-entropy loss, the adversarial examples generated on f_1 can relatively easily transfer to f_2 (see the performance of "Baseline" in Table 6.1). To enforce f_1 and f_2 to learn independent features, let the extracted feature of x in some intermediate layer of f be given as $g(x)$ (in this section we use the feature before the last fully connected layer as an example). We can still train f_1 using Loss_{CE}, and then, we train f_2 using,

$$\text{Loss}_{\text{total}} = \text{Loss}_{\text{CE}}(f_2(x), y) + \alpha \cdot \text{Loss}_{\text{DC}}(g_1(x), g_2(x)) \tag{6.9}$$

where α is a constant scalar and Loss_{DC} is the distance correlation from Definition 6.1. Note that we do not require $g_1(x)$ and $g_2(x)$ to be in the same dimension, so in principle we could easily use features from different layers for these two networks.

Experimental settings. We first conduct experiments on Cifar-10 Krizhevsky and Hinton (2009) using Resnet 18 He et al. (2016a). We then use four different architectures (mobilenet-v3-small Howard et al. (2019), efficientnet-B0 Tan and Le (2019), Resnet 34, and Resnet 152) and train them on ImageNet Krizhevsky et al. (2012). For each network architecture, we first train two networks using only Loss_{CE}. Next, we train a network using only Loss_{CE} before training a second network using the loss in (6.9). On Cifar-10, we utilize the SGD optimizer with momentum 0.9 and train for 200 epochs using an initial learning rate 0.1 with a cosine learning rate scheduler Paszke et al. (2019). The mini-batch size is set to 128. On ImageNet Krizhevsky et al. (2012), we train for 40 epochs using an initial learning rate 0.1, which decays by 0.1 every 10 epochs. The mini-batch size is 512. Our α in (6.9) is set to 0.05 for all cases. For each combination of the dataset and the network architecture, we train two networks f_1 and f_2, after which we generate adversarial examples on f_1 and use them to attack f_2 and measure its classification accuracy. We construct a baseline by training f_1 and $f_{2\mathtt{Baseline}}$ without constraints. And train $f_{2\mathtt{Our}}$ using (6.9) to learn independent features w.r.t. f_1. We report performance under two widely used attack methods: fast gradient sign method (FGM) Goodfellow et al. (2015) and projected gradient descent method (PGD) Madry et al. (2018), where the latter is considered among the strongest attacks. The scale ϵ of the adversarial perturbation is chosen from $\{0.03, 0.05, 0.1\}$ and the maximum number of iterations of PGD is set to 40.

Results. The results are shown in Table 6.1. We see that we get significant improvement in accuracy over the baseline under adversarial attacks, with comparable performance on clean inputs. Notably, our method achieves more than 10% absolute improvement in accuracy under PGD attack on Resnet 18 and Mobilenet-v3-small. This provides evidence supporting the benefits of enforcing the networks to learn independent features using our distance correlation loss.

In Figure 6.2, we show correlation results using Picasso Henderson and Rothe (2017); Chari and Pachter (2022) to lower the dimension of features for each network. The embedding dimension is 2 for visualization. In Figure 6.2(a), we show the embedding of different networks. f_1 represents the network to generate the

Table 6.1: The test accuracy (%) of a model f_2 on the adversarial examples generated using f_1 with the same architecture. "Baseline": train without constraint. "Ours": f_2 is independent to f_1. "Clean": test accuracy without adversarial examples.

Dataset	Network	Method	Clean	$FGM_{\epsilon=0.03}$	$PGD_{\epsilon=0.03}$	$FGM_{\epsilon=0.05}$	$PGD_{\epsilon=0.05}$	$FGM_{\epsilon=0.1}$	$PGD_{\epsilon=0.1}$
Cifar-10	Resnet 18	Baseline	89.14	72.10	66.34	62.00	49.42	48.23	27.41
Cifar-10	Resnet 18	Ours	87.61	**74.76**	**72.85**	**65.56**	**59.33**	**50.24**	**36.11**
ImageNet	Mobilenet-v3-small	Baseline	47.16	29.64	30.00	23.52	24.81	13.90	17.15
ImageNet	Mobilenet-v3-small	Ours	42.34	**34.47**	**36.98**	**29.53**	**33.77**	**19.53**	**28.04**
ImageNet	Efficientnet-B0	Baseline	57.85	26.72	28.22	18.96	19.45	12.04	11.17
ImageNet	Efficientnet-B0	Ours	55.82	**30.42**	**35.99**	**22.05**	**27.56**	**14.16**	**17.62**
ImageNet	Resnet 34	Baseline	64.01	52.62	56.61	45.45	51.11	33.75	41.70
ImageNet	Resnet 34	Ours	63.77	**53.19**	**57.18**	**46.50**	**52.28**	**35.00**	**43.35**
ImageNet	Resnet 152	Baseline	66.88	56.56	59.19	50.61	53.49	40.50	44.49
ImageNet	Resnet 152	Ours	68.04	**58.34**	**61.33**	**52.59**	**56.05**	**42.61**	**47.17**

adversarial examples. $f_{2Baseline}$ denotes the baseline network, trained without distance correlation constraint. Also, f_{2Ours} is the same network trained to be independent to f_1. In Figure 6.2(b), we visualize the correlation between f_1 and $f_{2Baseline}$ for each dimension, and the correlation between f_1 and f_{2Ours}. If the scatter plot looks circle-like, we can infer that the two models are independent. We see that in different networks, the use of DC shows stronger independence. From Figure 6.2/Table 6.1, we also see that the more independent the models are, the better is the gain for transferred attack robustness.

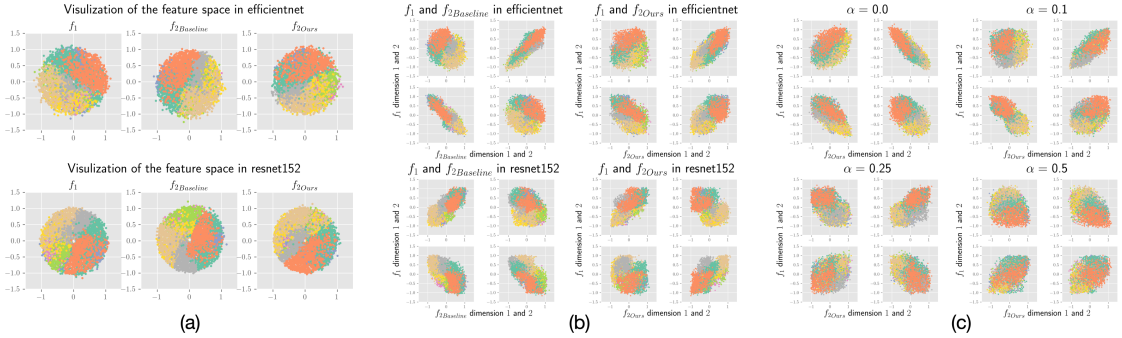

Figure 6.2: Picasso visualization of features space and the correlation between different models. (**a**) Feature space distribution. (**b**) Cross-correlation between the feature space of f_1 and f_2 trained with/without DC. We get better independence. (**c**) By increasing the balance parameter α of DC loss, Mobilenet is more independent to f_1.

6.5 Informative Comparisons between Networks

Overview. As discussed in Section 6.1, there is much interest in understanding whether two different models learn similar concepts from the data – for example, whether vision Transformers "see" similar to convolutional neural networks Raghu et al. (2021). Here, we first follow Raghu et al. (2021) and discuss similarities between different layers of ViT and Resnets using distance correlation. Next, we investigate that after taking out the influence of Resnets from ViT (or vice versa), what are the residual learned concepts remaining in the network.

Measure Similarity between Neural Networks

Goal. We first want to understand whether ViTs represent features across all layers differently from CNNs (such as Resnets). However, analyzing the features in the hidden layers can be challenging, because the features are spread across neurons. Also, different layers have different numbers of neurons. Recently, Raghu et al. (2021) applied the Centered Kernel Alignment (CKA) for this task. CKA is effective because it involves no constraint on the number of neurons. It is also independent to the orthogonal transformations of representations. Here, we want to demonstrate that distance correlation is a reasonable alternative for CKA in these settings.
Experimental settings. First, as described in Raghu et al. (2021), we show that similarity between layers within a single neural network can be assessed using distance correlation (see Figure 6.3). We pick ViT Base with patch 16, and three commonly used Resnets. All networks are pretrained on ImageNet. For ViT, we pick the embedding layer and all the normalization, attention, and fully connected layers within each block. The total number of layers is 63. For Resnets, we use all convolutional layers and the last fully connected layer. The total number of layers is the depth of the given Resnet.
Results (a). Our findings add to those from Raghu et al. (2021). Using distance correlation, we find that the ViT layers can be split into small blocks and the similarity between different blocks from shallow layers to the deeper layers is higher. For most Resnets, the feature similarity shows that there are a few large blocks in

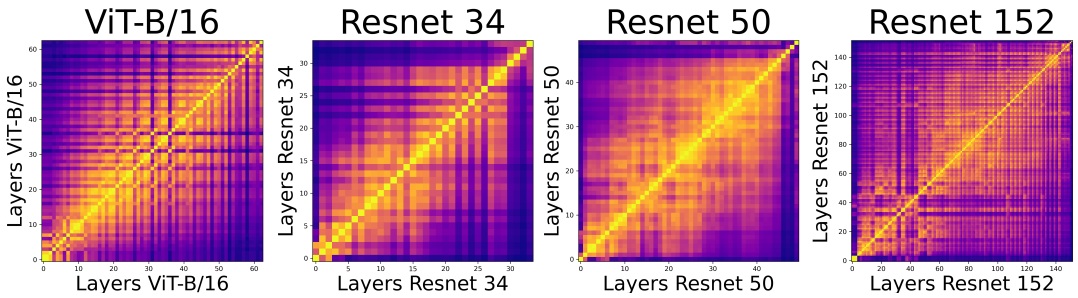

Figure 6.3: Similarity between layers within one single model. ViT can be split into small blocks and the similarity from shallow layers to the deeper layers is higher. Most Resnet models show few large blocks in the network, and the last few layers share minimal similarity with the shallow layers.

the network, which contains more than 30 layers each, and the last few layers share minimal similarity with the shallow layers.

Results (b). After within-model distance correlation, we perform across-model distance correlation comparisons between ViT and Resnets, see Figure 6.4. We notice that in the initial 1/6 layers, the two networks share high similarities. But later on, the similarity spreads across all different layers between ViT and Resnets. Notably, the last few layers share the least similarity between two networks.

By using the distance correlation to calculate the heatmap of the similarity matrices, we can qualitatively describe the difference between the patterns of the

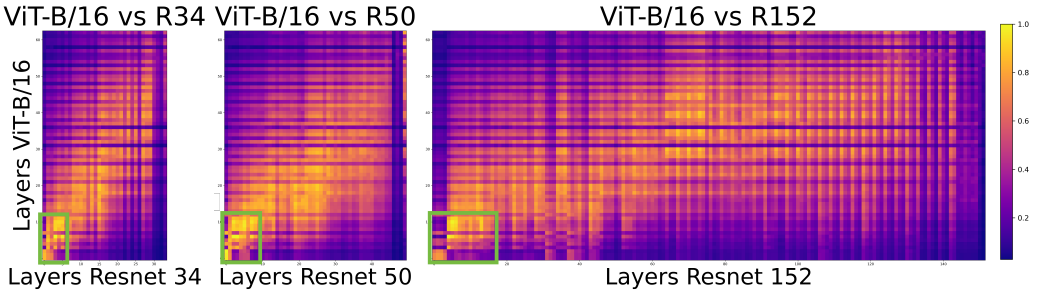

Figure 6.4: Similarity between layers across ViT and Resnets. In the initial 1/6 layers (highlighted in green), the two networks share high similarity. And the last few layers share the least similarity

features in different layers from different networks. What is even more interesting is to quantitatively show the difference, for example, to answer which network contains more information for the ground truth classes. We discuss this next.

What Remains When "Taking out" Y from X

Goal. Even measuring information contained in one neural network is challenging, and often tackled by measuring the accuracy on the test dataset. But the association between accuracy and the information contained in a network may be weak. Based on existing literature, conditioning one network w.r.t. another remains unresolved. Despite the above challenges, we can indeed measure the similarity between the features of the network X and the ground truth labels. If the similarity is higher, we can say that the feature space of X contains more information regarding the true labels. Distance correlation enables this. Interestingly, partial distance correlation extends this idea to multiple networks allowing us to approach the "conditioning" question posed above.

Rationale/setup. Here, we choose the last layer before the final fully-connected layer as the feature layer similar to the setup in Section 6.4. Our first attempt involved directly applying the distance correlation measurement to feature X and the one-hot ground truth embedding. However, the one-hot embedding for the label contains very little information, e.g., it does not show the difference between "cat" vs. "dog" and "cat" vs. "airplane". So, we use the pretrained BERT Devlin et al. (2019) to linguistically embed the class labels into the hidden space. We then measure the distance correlation between the feature space of X and the pretrained hidden space GT. $\mathcal{R}^2(X, GT) = \frac{m}{n}\sum_{t=1}^{n/m} dCor(x_t, gt_t)$ where x_t is the feature for one minibatch, and gt_t is the BERT embedding vector of the corresponding label. To further extend this metric to measure the "remaining" or residual information, we apply the partial distance correlation calculation by removing Y out of X, or say X conditioned on Y. Then, we have $\mathcal{R}^2\left((X|Y), GT\right) = \frac{m}{n}\sum_{t=1}^{n/m} dCor\left((x_t|y_t), gt_t\right)$ using (6.4). This capability has not been shown before.

Experimental settings. In order to measure the information remaining when con-

Table 6.2: Partial DC between the network Θ_X conditioned on the network Θ_Y, and the ImageNet class name embedding. The higher value indicates the more information.

Network Θ_X	Network Θ_Y	$\mathcal{R}^2(X, GT)$	$\mathcal{R}^2(Y, GT)$	$\mathcal{R}^2((X\|Y), GT)$	$\mathcal{R}^2((Y\|X), GT)$
ViT[1]	Resnet 18[2]	0.042	0.025	0.035	0.007
ViT	Resnet 50[3]	0.043	0.036	0.028	0.017
ViT	Resnet 152[4]	0.044	0.020	0.040	0.009
ViT	VGG 19 BN[5]	0.042	0.037	0.026	0.015
ViT	Densenet121[6]	0.043	0.026	0.035	0.007
ViT large[7]	Resnet 18	0.046	0.027	0.038	0.007
ViT large	Resnet 50	0.046	0.037	0.031	0.016
ViT large	Resnet 152	0.046	0.021	0.042	0.010
ViT large	ViT	0.045	0.043	0.019	0.013
ViT+Resnet 50[8]	Resnet 18	0.044	0.024	0.037	0.005
Resnet 152	Resnet 18	0.019	0.025	0.013	0.020
Resnet 152	Resnet 50	0.021	0.037	0.003	0.030
Resnet 50	Resnet 18	0.036	0.025	0.027	0.008
Resnet 50	VGG 19 BN	0.036	0.036	0.020	0.019

Accuracy: 1. 84.40%; 2. 69.76%; 3. 79.02%; 4. 82.54%; 5. 74.22%; 6. 75.57%; 7. 85.68%; 8. 84.13%

ditioning network Θ_Y out of Θ_X, we first use pretrained networks on ImageNet. We use the validation set of the ImageNet for evaluation. We want to evaluate which network contains the richest information regarding linguistic embedding. Interestingly, we can go beyond such an evaluation, instead, asking *the network Θ_X to learn concepts above and beyond what the network Θ_Y has learned*. To do so, we include the partial distance correlation into the loss. Unlike the experiment discussed above (minimizing distance correlation), in this setup, we seek to maximize partial distance correlation. The $\text{Loss}_{\text{total}}$ is

$$\text{Loss}_{\text{CE}}(f_1(x), y) - \alpha \cdot \text{Loss}_{\text{PDC}}\left((g_1(x)|g_2(x)), gt\right) \tag{6.10}$$

We take pretrained networks Θ_X, Θ_Y and then finetune Θ_X using (6.10). The learning rate is set to be $1e-5$ and α in the loss term is 1. To check the benefits of partial DC, we use Grad-CAM Selvaraju et al. (2017) to highlight the areas that

each network is looking at, together with what Θ_X conditioned on Θ_Y sees then.
Results (a). We first show information comparison between two networks. The details of DC and partial DC are shown in Table 6.2. The reader will notice that since ViT achieves the best test accuracy, it also contains the most information. Additionally, although better test accuracy normally coincides with more information, this is not always true. Resnet 50 contains more linguistic information than the much deeper Resnet 152, perhaps a compensation mechanism. For Resnet 152, the network is deep enough to focus on local structures that overwhelm the linguistic information (or this information is unnecessary). This experiment suggests a new strategy to compare two networks beyond test accuracy.
Results (b). After using a pretrained network, we can also check that by including the partial distance correlation in the loss, which regions does the model pay attention to, using Grad-CAM. We replace the loss term of Grad-CAM with the partial distance correlation. The results are shown in Figure 6.5. We see that the pretrained ViT sees across the whole image in different locations, while the Resnet (VGG) tends to focus on only one area of the image. After training, ViT (conditioned on Resnet) pays more attention to the subjects, especially locations outside the Resnet focus. Such experiments help understand how ViT learns *beyond* Resnets (CNNs).

6.6 Disentanglement

Overview. This experiment studies disentanglement Higgins et al. (2017); Kim and Mnih (2018); Chen et al. (2018); Locatello et al. (2020); Gabbay et al. (2021). It is believed that the image data are generated from low dimensional latent variables – but isolating and disentangling the latent variables is challenging. A key in disentangled latent variable learning is to make the factors in the latent variables independent Akash et al. (2021). Distance correlation fits perfectly and can handle a variety of dimensions for the latent variables. When the distance correlation is 0, we know that the two variables are independent.

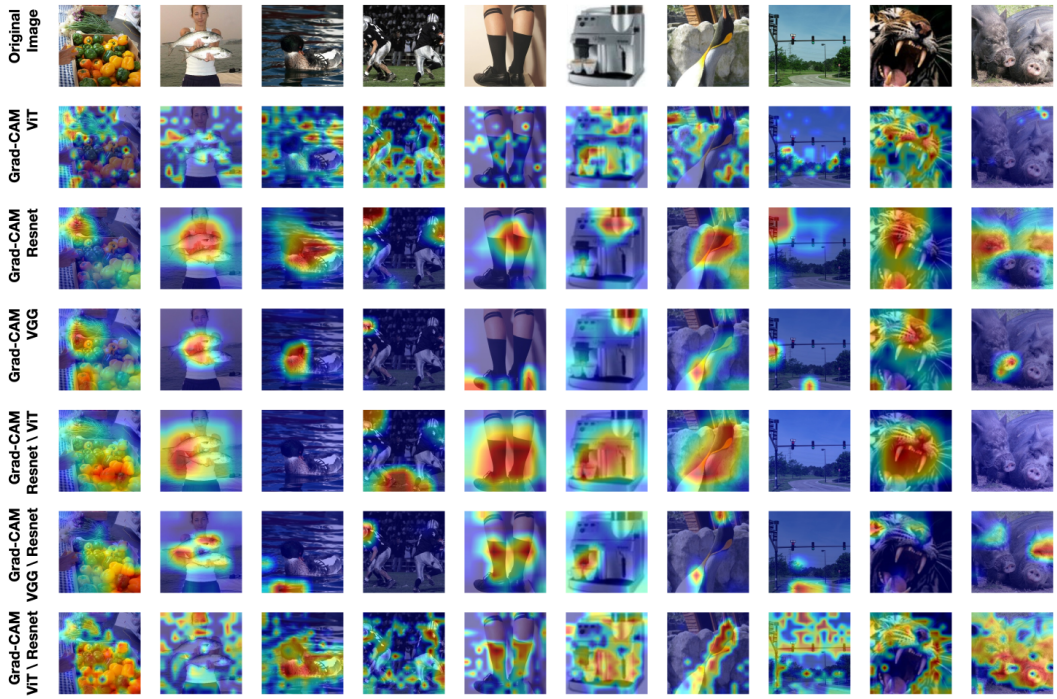

Figure 6.5: Grad-CAM results on ImageNet using ViT, Resnet 18 and VGG 16. After using Partial DC to remove the information learned by another network, ViT can focus on detail places and Resnet can only look in major spots. Similar issue happens to VGG.

Experimental settings. We follow Gabbay et al. (2021) which focuses on semi-supervised disentanglement to generate high-resolution images. In Gabbay et al. (2021), one divides the latent variables into two categories: (a) attributes of interest – a set of semantic and interpretable attributes, e.g. hair color and age; (b) residual attributes – the remaining information. Formally, $x_i = G(f_i^1, ..., f_i^k, r_i)$, where G is the generator that uses the factors of interest f_i^l and the residual to generate image x_i.

In order to enforce the condition that the information regarding the attributes of interest is not leaking into the residual representations, the authors of Gabbay et al. (2021) introduced the loss $L_{res} = \sum_{i=1}^{n} ||r_i||^2$ to limit the residual information. This is sub-optimal as there can be cases where r_i is not 0 but still independent to

the factors of interest $(f_i^l)_{l=1}^k$. Thus, we use distance correlation to replace this loss:

$$L_{res} = dCor([f^1; f^2; ...; f^k], r) \tag{6.11}$$

We use the same structure proposed in Gabbay et al. (2021), while the generator architecture is adopted from StyleGAN2 Karras et al. (2020). The dataset is the human face dataset FFHQ Karras et al. (2019), and the attributes are: age, gender, etc. We use CLIP Radford et al. (2021) to partially label the attributes to generate the semi-supervised dataset for training. All losses from Gabbay et al. (2021) are used, except that L_{res} is replaced by (6.11). Liu et al. (2021) also discusses a nice demonstration of distance correlation in terms of how it helps content style disentanglement.

Results. (Shown in Figure 6.6) Our model shows the ability to change specific

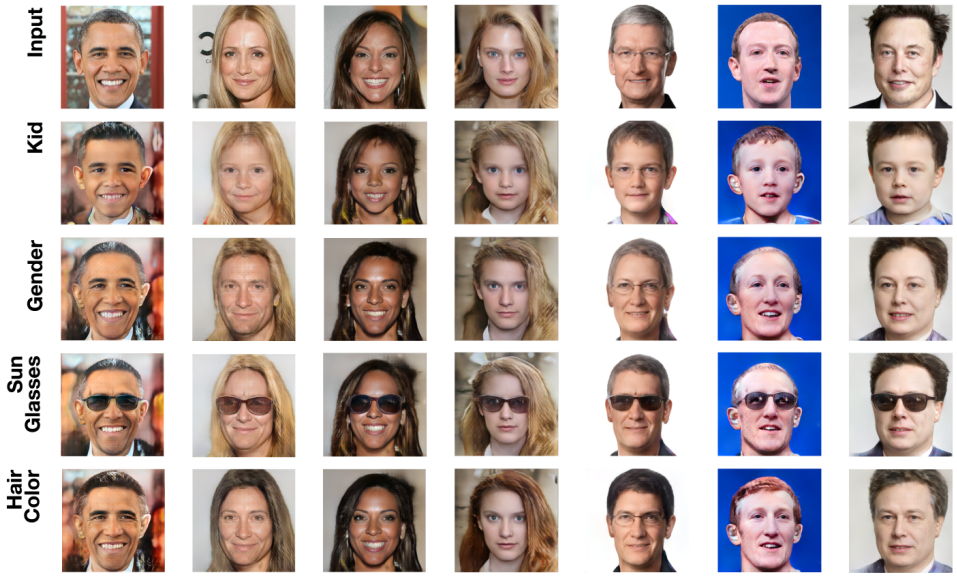

Figure 6.6: Representative generated images using our training on FFHQ. Note that these results only use semi-supervised dataset by CLIP. Our methods shows the ability to disentangle the attributes of interest and the remaining information.

attributes without affecting residual features, such as posture.

6.7 Conclusions

In this chapter, we studied how distance correlation (and partial distance correlation) has a wide variety of uses in deep learning tasks in vision. The measure offers various properties that are often enforced using alternative means, that are often far more involved. Further, it is extremely simple to incorporate in contrast to various divergence-based measures often used in invariant representation learning. Notably, the use of partial distance correlation offers the ability of conditioning, which is under-explored in the community. We showcase three very different settings, ranging from network comparison to training distinct/different models to disentanglement where the idea is immediately beneficial, and expect that numerous other applications will emerge in short order. The code for the algorithms described in this chapter is available at `https://github.com/zhenxingjian/Partial_Distance_Correlation`.

7 CONCLUSIONS

In this thesis, we explored various types of modifications required for deep learning models to operate on Riemannian manifold-valued data and structured matrices. We demonstrated that with these modifications, several applications can benefit, including analysis of 1D sequential SPD matrices along brain fiber bundles, 3D $\mathrm{SPD}/\mathbf{S}^{n-1}$ valued voxels in brain images, and covariance matrices that represent the pixel distribution after perturbation. We also studied the use of distance correlation in various settings in deep learning. We showed that in these different applications, the models can leverage inherent geometric information of the data to improve efficiency, test accuracy, generative quality, and more. Specifically,

1. In Chapter 3, we modified the convolutional operators and residual connections using a weighted Fréchet mean (wFM) on manifold-valued data. Additionally, we replaced the bias term with a group operation. With these modifications, we showed that our model can more accurately detect pre-AD signals in dMRI scans. Furthermore, our modified model also demonstrated improved efficiency on vision datasets.

2. In Chapter 4, we incorporated the exponential map and logarithm map of $\mathrm{SPD}(3)$ and $\mathbf{S}^{n-1}$ into an invertible generative model called GLOW. We demonstrated that by using two manifold GLOW models in parallel, we can generate an ODF based on a given DTI while preserving meaningful statistical differences across different groups.

3. In Chapter 5, we introduced a tracking-based method to estimate the distribution of every pixel in each layer when given a random perturbation. We demonstrated that the distribution can be expressed as a normal distribution, while the covariance can be estimated using SPD matrices with some additional adjustment. Our experimental results showed that this approach can improve the robustness of a given model and enable more efficient training compared to sampling methods.

4. In Chapter 6, we demonstrated how geometric information, specifically distances, can be used to estimate the correlation between two random vectors. We showed that this simple idea, called distance correlation, can offer several benefits in different deep learning applications. For example, it can improve robustness against transferable attacks, facilitate disentanglement in generative models, and enable quantitative comparison of information between two different structured neural networks.

7.1 Future Direction and Ongoing Projects

We now discuss some ongoing and future research directions building off of our work described in Chapters 3 - 6.

Hyperbolic Space Embedding for Patches with Hierarchy in 3D Brain Scan Images

Throughout Chapters 3 to 6, we focused on a small subset of structured data, treating each pixel or patch or fiber in the image as equivalent without considering its hierarchical structure. However, real-world visual data often exhibits inherent hierarchies, such as the relationship between a cat, its ears, and its eyes in an image. Representing such structures effectively requires hyperbolic spaces, as demonstrated by recent work on hyperbolic neural networks Chami et al. (2020b); Fan et al. (2022). Yet, implementing these networks often demands extensive knowledge of hyperbolic spaces and careful design and implementation of operations, limiting generalization and deployment of rapidly developed formulations. In this ongoing collaborative project, we aim to explore the use of hyperbolic spaces while only imposing some basic constraints and using distances induced by given hyperbolic spaces. Our preliminary experiments show that this method can provide a meaningful hyperbolic embedding for learning representations from 3D brain imaging data and improved rotation invariance without requiring extensive design.

Rationale for Hierarchy in Representation Learning

In machine learning, accurately representing and manipulating data is becoming increasingly important as models become more complex and datasets grow larger. Traditionally, Euclidean geometry has been the default choice for representing data due to its intuitive vector space and stable closed-form formulas Bdeir et al. (2023). However, recent research by Dhingra et al. (2018); Lensink et al. (2019) has demonstrated that hyperbolic geometry may be better suited for capturing hierarchical structures in images. This is because the negative curvature of hyperbolic manifolds allows distances to scale exponentially with respect to the radius, which matches the exponential scaling of tree distances between graph nodes Sarkar (2011). Consequently, hyperbolic geometry can better retain graph information while preventing spatial distortion.

The tree structure (or hierarchical structure), is prevalent in Natural Language Processing (NLP) tasks Peng et al. (2022), where multiple words form a sentence, multiple sentences form a paragraph, and multiple paragraphs form a document. As such, documents should be at a higher level of hierarchy than paragraphs, while words should be at the lowest. However, it is important to note that hierarchical structures are not only limited to textual data, but also appear in images. The concept of part-whole relationships within object representations and classes establishes a clear notion of hierarchy in images Khrulkov et al. (2020). Recent works have incorporated hyperbolic geometry into neural network architectures for vision in response to the hierarchical structures in vision data Liu et al. (2020); Bdeir et al. (2023). Specifically, they utilize the Poincaré ball to describe hyperbolic space and formalize hyperbolic translations of Euclidean machine learning operations Mishne et al. (2022). However, at present, most Hyperbolic Neural Network (HNN) components are only available in the Poincaré ball setup due to its support for gyrovector space and basic vector operations Hsu et al. (2021). However, the Poincaré ball is known to be numerically unstable Yu and Sa (2019). Additionally, the high computational cost has limited HNNs in computer vision to a mixture of Euclidean backbones and hyperbolic task heads Ganea et al. (2018).

Another limitation of existing methods involving hyperbolic neural networks is that it requires extensive knowledge of the geometry of different types of hyperbolic spaces. This can pose a challenge for practitioners who may want to adapt an existing method to their specific use case. However, since such spaces can be captured by an appropriate metric, which only requires minimal knowledge of the geometry of hyperbolic spaces, it may be possible to utilize only the distance information in hyperbolic spaces and extend neural network architectures from Euclidean space to hyperbolic spaces and assess if empirical behavior is satisfactory.

In this section, we investigate the feasibility of such a method, which involves a contrastive learning setup where representations from the same hierarchical branch have smaller distances in the embedding space, while those from different branches have larger distances. To meaningfully evaluate this idea, we utilize 3D brain scan images due to the lack of mature auto-encoder models for 3D brain scan reconstruction where this idea can provide improvements. We demonstrate that our method can embed various patches of different sizes and hierarchies from the same brain scan image into a hyperbolic space in almost linear time during training while the brute force way would be $O(N^2)$ where N is the number of patches. Our experiments show that this method effectively reconstructs the brain scan and maintains a more rotation-invariant embedding space compared to the normal Euclidean auto-encoder model. This property is advantageous in various vision tasks and highlights the robustness of our proposed method in the context of medical imaging.

Efficiently Finding Hyperbolic Embedding with Metric

Hyperbolic formulation. In Example 1.7, we discussed some use cases. In this section, we will delve deeper into the mathematical background of hyperbolic spaces.

The hyperbolic space, also known as a negatively curved space, is non-Euclidean similar to examples discussed in previous chapters. There are several models used to describe a hyperbolic space. One such model is the Poincaré ball model, which is

commonly used in many deep learning model setups, as seen in Hsu et al. (2021). The Poincaré ball model (of curvature $c = -1$) is defined as the open ball of radius 1 centered at the origin, equipped with the metric tensor $\mathfrak{g}_p = (\lambda_x)^2 \mathfrak{g}_e$, where the conformal factor $\lambda_x = \frac{2}{1-|x|^2}$ and $\mathfrak{g}_e$ is the Euclidean tensor. The distance between two points in this model is given by:

$$d(x_1, x_2) = \operatorname{arcosh}\left(1 + 2\frac{\|x_1 - x_2\|^2}{(1 - \|x_1\|^2)(1 - \|x_2\|^2)}\right)$$

It should be noted that this model may not be numerically stable when the magnitudes of $\|x_1\|$ and $\|x_2\|$ approach 1.

On the other hand, another type of hyperbolic space is the Poincaré half-plane model Tung (2018), where features can be represented as (x, y) with $x \in \mathbb{R}^n$ and $y \in \mathbb{R}^+$. In this model, the distance between two points is defined as:

$$d((x_1, y_1), (x_2, y_2)) = \operatorname{arsinh}\left(\frac{|(x_1, y_1) - (x_2, y_2)|}{2\sqrt{y_1 y_2}}\right) \tag{7.1}$$

It is worth noting that when features are extracted using neural networks, the Poincaré half-plane model is more numerically stable than the Poincaré ball model from the experimental results. This is because we can easily ensure that y is away from zero using a nonlinear function, while modifications to the Poincaré ball model would require greater care to push x away from the boundary. Additionally, the Poincaré half-plane model is easier to extend from Euclidean spaces; we can use a nonlinear function to project the last dimension to $\mathbb{R}^+$ to create an embedding that resembles a Poincaré half-plane embedding. Therefore, in this section, we will focus on the Poincaré half-plane model and explore its capabilities.

Unsupervised hyperbolic representation learning. We introduce a hyperbolic auto-encoder that utilizes a 3D convolutional encoder and decoder to handle 3D input. Our encoder maps a sampled patch from a given 3D brain scan image to a Euclidean vector space. Next, we apply a simple projection to the last dimension

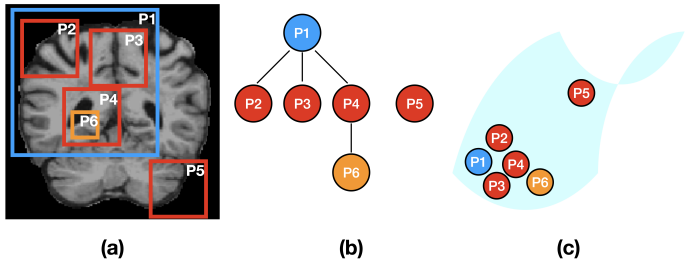

Figure 7.1: We show the random generated patches in (a). The corresponding tree structures are shown in (b). The potential goal of the unsupervised learning is that the embeddings of the same branch are close to each other, while different branches are far apart, see (c).

to convert it to a positive value, resulting in features embedded in a Poincaré half-plane model. In the preliminary experiments we tried, the projection function is set as SoftPlus function:

$$\text{SoftPlus}(x) = \log(1 + \exp(x))$$

The decoder then reconstructs the input patch from this embedded feature, without any hyperbolic operations such as Gyrovector operations. It is important to note that the entire neural network is optimized similar to a Euclidean neural network, except the simple projection layer. We aim to minimize the computational cost associated with handling a hyperbolic neural network.

To capture the hierarchical structures present in an image in an unsupervised manner, we randomly sample patches of different sizes at different locations. We define the hierarchical structure based on the containment relationship between the patches. Specifically, if one patch P_1 is contained within another patch P_2, we consider P_1 to be under P_2 in the same branch of the tree generated by our random sampling method. Conversely, if two patches P_1 and P_3 have little overlap or are far apart, we consider them to be in different branches and do not know their hierarchy. An example is shown in Figure 7.1.

For all the patches of the given image, we try to optimize the following equation:

$$\ell_{\mathcal{H}} = \sum_{i} \sum_{j,s.t.(x_i,x_j)\in T_k} d((x_1, y_1), (x_2, y_2)) \tag{7.2}$$

where T_k is the k_{th} tree, (x_i, y_i) is the hyperbolic embedding of P_i, $d(\cdot,\cdot)$ is the distance metric of the space, and in the this section, we apply (7.1). Thus, the embedding of the patches from the same branch will be close to each other, (see Figure 7.1(c) as an example).

Efficiency using a Hash function on a hyperbolic space. One can notice that in the loss function described in (7.2), the major difficulty is to identify all the nodes in the same tree. Assume that we have N patches in the entire image, and we do not contain an explicit expression of the tree structures inherent in these patches. The brute force way to find the hierarchical information is by computing the distances (either among the physical locations or among the embeddings), which would require a time cost of $O(N^2)$. One possible approach to addressing this challenge is to use hash functions to efficiently identify nodes or embeddings that belong to the same tree with a time cost of $O(N)$. This can be achieved by extending locality-sensitive hashing (LSH) Chen et al. (2020); Jafari et al. (2021) from Euclidean space into hyperbolic space with some modifications. LSH has been shown to be useful in nearest neighbor search.

To better understand LSH in the Euclidean setting, let us briefly review it.

Definition 7.1 (Locality-sensitive hashing (LSH))**.** *A family $\mathcal{H}$ is called (S_0, cS_0, p_1, p_2)-sensitive if for any two points $x_1, x_2 \in \mathbb{R}^D$ and h chosen uniformly from H satisfies the following:*

- *if $Sim(x_1, x_2) \geqslant S_0$ then $Pr(h(x_1) = h(x_2)) \geqslant p_1$*
- *if $Sim(x_1, x_2) \leqslant cS_0$ then $Pr(h(x_1) = h(x_2)) \leqslant p_2$*

For approximate nearest neighbor search, we need $p_1 > p_2$ and $c < 1$ to hold.

There are multiple choices for LSH families, such as p-stable distributions Datar et al. (2004), random projections Charikar (2002), and E2 LSH Zhang et al. (2014).

In this work, we propose to use the random projection method, which has been used in deep learning in the context of hashing Zeng et al. (2021); Chen et al. (2021) and easy to extend to hyperbolic spaces.

Definition 7.2 (Random Projections)**.** *Given an input vector* $x \in \mathbb{R}^n$ *and a hyperplane defined by* $r = (r_1, r_2, \cdots, r_n)$ *with each* $r_i \sim \mathcal{N}(0,1)$*, the hash function is defined as* $h(x) = r \cdot x$*. To further quantize* $h(x)$ *into a set of hash buckets, we have the following hash function*

$$h_{r,b}(x) = \lfloor \frac{x \cdot r + b}{w} \rfloor$$

where w *is the length of each quantization bucket and* b *is a random variable* $b \sim \mathcal{N}(0,1)$*.*

By performing a projection on K random directions in parallel, we obtain $h(x) = [h_1(x), h_2(x), \cdots h_K(x)] \in \mathbb{R}^K$. h_K here are the hash functions defined in Definition 7.2. We say that x_1 and x_2 are candidates for neighbors if $\forall i,\ h_i(x_1) = h_i(x_2)$.

In Euclidean random projection, the key component is the hyperplane, which cannot be directly extended into hyperbolic spaces. Thus we will need to make some modification in hyperbolic spaces to enable the LSH model to operate well. We know that LSH explores the relationship between similarity (induced by the distance) and the collision probability. So if we can modify the setup such that we can use, from a computational perspective, hyperbolic distances similar to how we work with Euclidean distances, we may be able to apply LSH in our setup. We can notice that the major difference between the distance in the Poincaré half-plane model and the distance in the Euclidean space is that the Poincaré half-plane distance is scaled by y. If we can remove the dependency of y, we can extend the LSH from Euclidean space to Hyperbolic spaces.

Therefore, we first divide the entire Poincaré half-plane space into multiple stripes along the y-axis, see Figure 7.2. We say that two points (x_1, y_1) and (x_2, y_2) are candidates for neighbors if they fall into the same stripe and have the same hash code within the stripe. In this setting, $\sqrt{y_1 y_2}$ is bounded, so that $d((x_1, y_1), (x_2, y_2))$

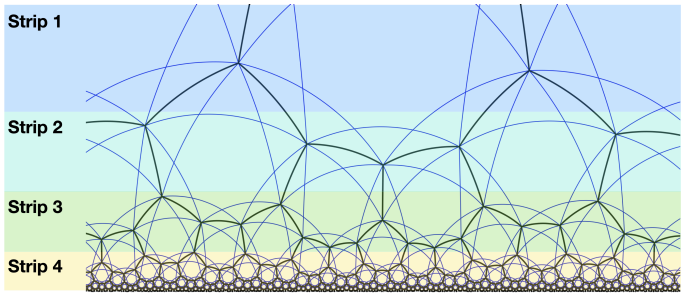

Figure 7.2: We divide the entire Poincaré half-plane space into multiple stripes along the y-axis. We only consider points within the same strip being potential near neighbor candidates and apply Euclidean LSH inside each strip.

can be rewritten as $\text{arsinh}(C\|(x_1, y_1) - (x_2, y_2)\|)$, which is a monotonic function of the Euclidean distance.

Note that for different hyperbolic embeddings, the hash function h is shared. We only sample r_i and b_i once.

We empirically verify that Definition 7.1 can be satisfied with the hyperbolic hash function defined in Algorithm 7.1. The results are shown in Figure 7.3.

Algorithm 7.1: Hyperbolic Hash Code

Input: Hyperbolic embedding (x, y), y axis stripe scale α_y, number of hash code N, and the length of each quantization bucket w.

Output: Hash code of $h(x, y)$

1: Compute the stripe index of y, $h_y(y) = \lfloor \alpha_y \log(y) \rfloor$
2: **for** $i = 1, \cdots, N$ **do**
3: Sample hyperplane $r_i \sim \mathcal{N}(\mathbf{0}, I)$ and bias $b_i \sim \mathcal{N}(0, 1)$
4: $h_i(x, y) = \lfloor \frac{(x,y) \cdot r_i + b_i}{w} \rfloor$
5: **end for**
6: Hash code for (x, y) is $h(x, y) = [h_y(y), h_1(x, y), \cdots, h_N(x, y)]$

Preliminary Experimental Results

As discussed above, the main focus of this work is to find meaningful hyperbolic embeddings for different patches inside 3D brain scan data using autoencoder-

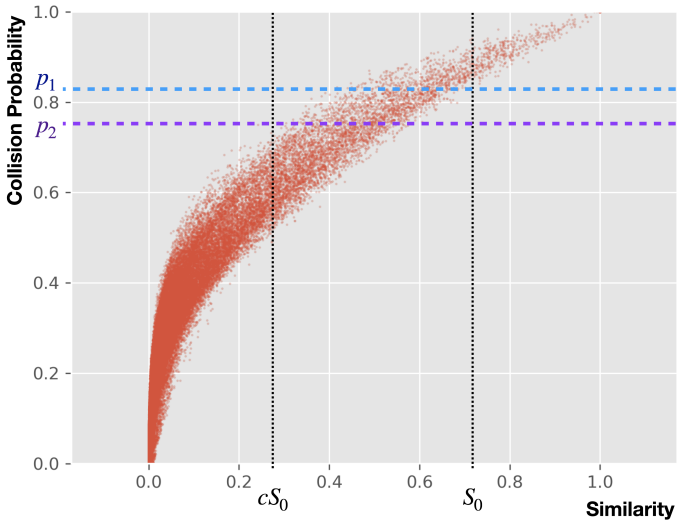

Figure 7.3: We empirically verify that the hyperbolic hash function satisfies the requirement in Definition 7.1. x axis shows the similarity between two given points, which is defined as $e^{-d((x_1,y_1),(x_2,y_2))}$ and y axis shows the collision probability. We can notice that $c < 1$ and $p_1 > p_2$. cS_0 here serves as the lower threshold that if the similarity is lower than cS_0, it is very unlikely to collide (in terms of hashing). S_0 shows that if the similarity is higher than S_0, it is more likely to collide.

based models. We first apply the proposed method to the Alzheimer's Disease Neuroimaging Initiative 3 (ADNI3) dataset. Detailed demographics can be found in Table 7.1.

Experimental setup. The encoder in this experiment consists of multiple layers of 3D convolution with ReLU activation functions in between. We further utilize an average pooling layer and two linear layers to embed the features of the given input. To project these features into the Poincaré half-plane model, we apply the

Table 7.1: The demographics used in the study.

Age	**Gender (male)**	AD	MCI	CN
71.42 (6.92)	300 (46.2%)	39	270	340

softplus activation function on the last dimension to make it a positive scalar value (as required for y). For the decoder model, we use the inverse structure of the encoder model.

During the training phase, for each 3D brain scan image, we randomly crop 1000 patches with 3 different sizes ranging from 8 to 16. In the testing phase, we fix the patch size to evaluate the performance of our model. We train two models: one with hyperbolic loss (our method) and the other without hyperbolic loss (which serves as the baseline model). We first evaluate the reconstruction error on the hold-out test dataset of our method and the baseline method. The reconstruction error ranges between $5.0e-3$ to $6.4e-3$ for different patch sizes, which shows the ability of the proposed autoencoder model to meaningfully extract features in the latent space. Some examples of the reconstruction results for both the baseline model and our model can be found in Figure 7.4. Also, we would like to point out that our model takes only slightly longer to train per epoch compared to the baseline model. Specifically, it takes 4.77 minutes to train one epoch, while the baseline model takes 4.71 minutes.

We then further evaluate the invariance of rotation even if we do not provide rotation data samples during the training phase. Since our method works on the inherent structural relationship between patches, we hypothesize that the features extracted by our method can be more invariant to rotation. This property indicates the robustness of our method. The results are shown in Figure 7.5.

To visualize the hyperbolic embedding space of our proposed method, we show patches of a fixed size of 8. Since the dimension of x is relatively high (127 in this experiment), we apply a random projection vector $v \sim \mathcal{N}(\mathbf{0}, I)$ onto the x axis and show the projected x alongside the original y in Figure 7.6. To demonstrate the hierarchical information inside this hyperbolic space, we also present features of patches with different sizes in Figure 7.7.

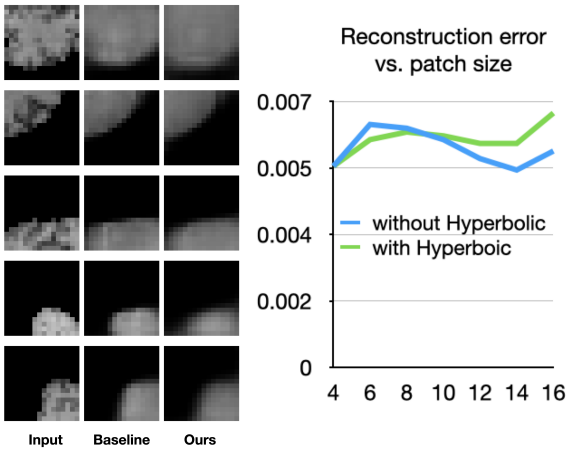

Figure 7.4: Some examples of the reconstruction results are shown on the left. Due to the fact that our proposed model deals with patches, the images shown here are different patches instead of the entire brain. On the right, we show the reconstruction error range for both the baseline model and our model. We can see that the reconstruction is comparable for our method and the baseline model.

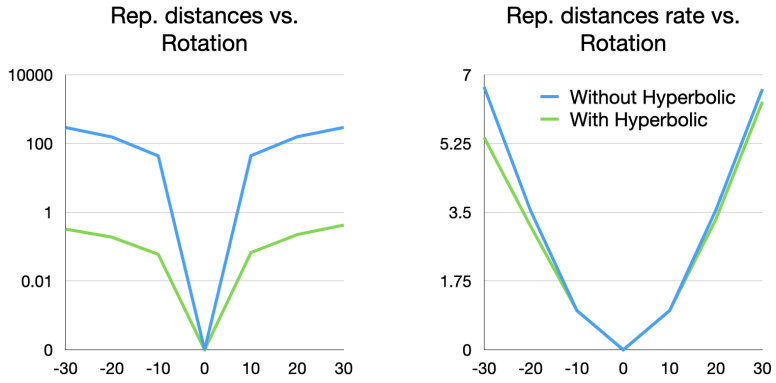

Figure 7.5: We evaluated the invariance of rotation of our method and the baseline method. The left figure shows the distance between feature representations before and after rotation. Our method shows much smaller distance in the latent embedding space. The right figure shows the ratio between the distance of different rotation angles and the distance of a 10° rotation. Our method outperforms the baseline model in terms of invariance to rotation.

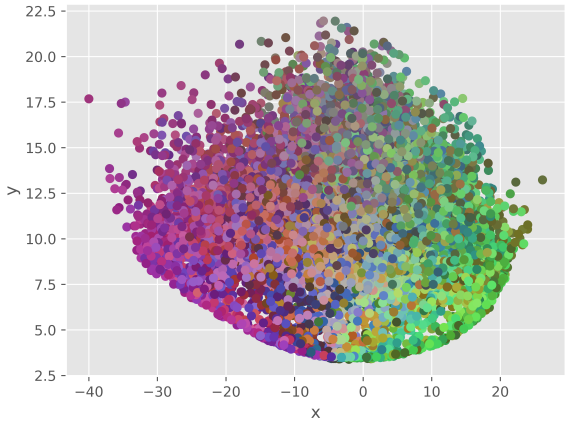

Figure 7.6: We visualize the embedding space of our proposed method, where the color indicates the patch location in the 3D brain scan image. Similar colors indicate similar spatial locations. This indicates that our method can preserve similarities in the patches, i.e., patches from a similar location tend to be together because we see that patches from the same branch tend to be together in the embedding space.

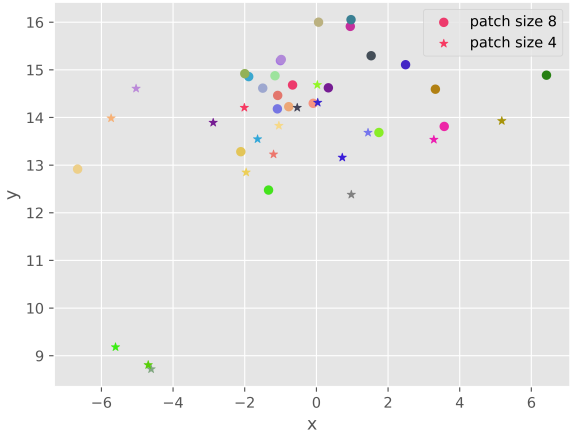

Figure 7.7: We present the features in the hyperbolic space with different patch sizes. We can observe that patches with size 4 generally exhibit lower hierarchy than those with size 8.

Distance Correlation for Self Supervised Learning

In Chapter 6, we explored three distinct use cases for distance correlation and partial distance correlation. Now, we will examine the potential advantages of

using distance correlation within a self-supervised learning framework.

In recent years, there has been significant progress in the field of computer vision with regards to unsupervised and self-supervised learning Jaiswal et al. (2020). Notable works include Grill et al. (2020); Chen and He (2021), which are based on the concept of Siamese networks Bromley et al. (1993), which are weight sharing networks for two or more inputs.

For instance, BYOL Grill et al. (2020) generates two augmented views $v = t(x)$ and $v' = t'(x)$ from the same image x, and trains an online network f_θ. During training, the parameters θ are smoothed using an exponential moving average to form the target network parameters ξ, which are not trained through back propagation. The objective is to maximize the similarity between the projected online network feature $h_\zeta \circ f_\theta(v)$ and the target network feature $f_\xi(v')$. The projection network, or predictor h_ζ, serves as an alignment method and typically involves the use of a multilayer perceptron (MLP). Another example is SimSiam Chen and He (2021), which simplifies the network structure of the BYOL model by removing the momentum encoder f_ξ. In SimSiam, a single network f_θ and predictor h_ζ are trained. The network f_θ is applied to both v and v', and the objective is to maximize the similarity between $h_\zeta \circ f_\theta(v)$ and $f_\theta(v')$. It is worth noting that the loss does not back propagate through the v' branch. A diagram of this process is illustrated in Figure 7.8(a).

One of the major benefits of BYOL and SimSiam is that they do not require hard assignment of negative pairs, which can be difficult to define when there is no label information available. Instead, positive pairs are defined as the different views v and v' of the same image x, and the similarity between the two views is defined as the cosine similarity between the corresponding feature vectors:

$$\frac{\langle h_\zeta \circ f_\theta(v), f_\theta(v') \rangle}{\|h_\zeta \circ f_\theta(v)\| \|f_\theta(v')\|}$$

The similarity function measures the agreement between the projections of the two views in this latent space. The cosine similarity is commonly used in these methods because it is a normalized measure of similarity, and it is invariant to scaling and

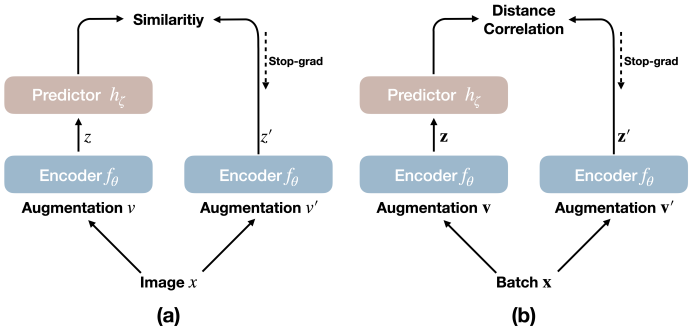

Figure 7.8: (a) SimSiam architecture. Two augmented views of one image passing through the same encode network f_θ and try to align the latent representation via a prediction network h_ζ. To measure the alignment, it uses the cosine similarity. (b) Our architecture. Our model takes the entire batch into consideration and try to maximize the distance correlation between the two views of the same batch images.

translation of the feature vectors.

However, using correlation instead of cosine similarity can provide additional benefits by considering all the samples from the same batch (both positive and negative pairs) even without label information. Correlation is also invariant to scaling and translation, and can be used to incorporate information not only from the positive pairs defined by multiple augmentations of the same input sample, but also from other samples in the batch.

To achieve this, we can apply the distance correlation to this setup by replacing the cosine similarity. Unlike other self-supervised learning methods, such as Wu et al. (2018); Caron et al. (2020), which treat all different input samples as negative pairs, our approach does not require hard assignment of positive and negative pairs in the batch. In this setting, the loss function is defined as:

$$\begin{aligned} \ell(\mathbf{x}) &= -\mathcal{R}(h_\zeta(\mathbf{z}), \mathbf{z}') \\ \mathbf{z} &= f_\theta(\mathbf{v}) \\ \mathbf{z}' &= f_\theta(\mathbf{v}') \end{aligned}$$

where bold symbols represent a batch of input samples, $\mathcal{R}$ is the distance correlation

Table 7.2: The test accuracy on Cifar-10 dataset with different training epochs using SimSiam-based network with cosine similarity and our distance correlation.

Training Epochs	100			800
Backbone Network	Resnet 18	Resnet 34	Resnet 50	Resnet 18
Cosine Similarity	42.08%	40.60%	42.90%	83.78%
Distance Correlation	45.89%	44.45%	43.97%	80.68%

function defined in (6.2), and $\mathbf{v}$ and $\mathbf{v}'$ are two different augmentations of the same input image x. A diagram of this process is illustrated in Figure 7.8(b).

Our proposed method was evaluated on the Cifar-10 dataset as a preliminary test of the idea. To assess the quality of the learned latent representation, we trained a linear classification model using the features extracted by f_θ and the true labels. Preliminary results show that our method achieved 45.89% accuracy on Cifar-10, outperforming SimSiam, which achieved 42.08% accuracy after 100 epochs of training. After a total of 800 epochs of training, our method achieved 80.68% accuracy, while SimSiam showed 83.78% accuracy. The training time is comparable between our method and SimSiam. We would like to note that the reported accuracy here is slightly lower than that of the original SimSiam model due to the unavailability of detailed training code for the Cifar-10 dataset in the original paper. But as a fair comparison, we use the exactly same structure and training strategy for both SimSiam and our distance correlation method. We use batch size 512, initial learning rate 0.05, and a cosine decay schedule for 100 epochs (or 800 epochs). The results for different structures can be found in Table 7.2. These results suggest that our method could be a promising warm-up strategy for different self-supervised learning methods and architectures.

A SIMPLER CERTIFIED RADIUS MAXIMIZATION BY PROPAGATING COVARIANCES: APPENDIX

A.1 Tracking Distributions through Layers

In Section 5.3, we briefly described the different layers used in our model. Here, we will provide more details about the layers.

We consider the i^{th} pixel, after perturbation, to be drawn from a Gaussian distribution $\mathbf{x}_i \sim \mathcal{N}(\boldsymbol{\mu}_i, \Sigma)$, where $\boldsymbol{\mu}_i \in \mathbb{R}^N$ and Σ is the covariance matrix across the N channels (note that Σ is same for a layer across all pixels). In the following sections, we will remove the indices to simplify the formulation.

Several commonly-used basic blocks such as convolution and fully connected layers are used to setup our network architecture. In order to propagate the distribution through the entire network, we need a way to propagate the moments through these layers.

Convolution Layer

Let the pixels $\{\mathbf{x}_i\}$ inside a $k \times k$ convolution kernel window be independent. Let $\widetilde{\mathbf{x}} \in \mathbb{R}^{N_{in}k^2}$ be the vector which consists of $\{\mathbf{x}_i\}$, where N_{in} is the number of input channels. Let $\Sigma \in \mathbb{R}^{N_{in} \times N_{in}}$ be the covariance matrix of each pixel within the $k \times k$ window. Let W be the weight matrix of the convolution layer, which is of the shape $N_{in} \times k \times k \times N_{out}$. With a slight abuse of notation, let W be the reshaped weight matrix of the shape $N_{in}k^2 \times N_{out}$. Further, concatenate Σ from each $\{\mathbf{x}_i\}$ inside the $k \times k$ window to get a block diagonal $\widetilde{\Sigma} \in \mathbb{R}^{N_{in}k^2 \times N_{in}k^2}$. Then, we get the covariance of an output pixel to be $\Sigma_{\mathbf{h}} \in \mathbb{R}^{N_{out} \times N_{out}}$ defined as

$$\Sigma_{\mathbf{h}} = W^T \widetilde{\Sigma} W$$

We need to apply Theorem 5.2 to compute the upper bound of $\Sigma_{\mathbf{h}}$. We get the upper bound covariance matrix (covariance matrix of independent random variable

$\widehat{\mathbf{x}}$ used in Theorem 5.2) as

$$\widehat{\Sigma_{\mathbf{h}}} = (1 + r_{max}) W^T \widetilde{\Sigma} W$$

Summary: Given each input pixel, $\mathbf{x}_i \in \mathbb{R}^{N_{in}}$ following $\mathcal{N}(\boldsymbol{\mu}_i, \Sigma)$ and convolution kernel matrix W, the output distribution of each pixel is $\mathcal{N}(\boldsymbol{\mu}_{h_i}, (1 + r_{max}) W^T \widetilde{\Sigma} W)$, where, $\widetilde{\Sigma}$ is the block diagonal covariance matrix as mentioned before.

First Linear Layer

For the first linear layer, we reshape the input in a vector by flattening both the channel and spatial dimensions. Similar to the convolution layer, we concatenate $\{\mathbf{x}_i\}$ to be $\widetilde{\mathbf{x}}$, whose covariance matrix has a block-diagonal structure. Thus, the covariance matrix of the output pixel is

$$\Sigma_{\mathbf{h}} = W^T \Sigma_{\widetilde{\mathbf{x}}} W$$

where W is the learnable parameter and $\Sigma_{\widetilde{\mathbf{x}}}$ is the block-diagonal covariance matrix of $\widetilde{\mathbf{x}}$ similar to the convolution layer.
Special case: From Observation 5.1, we only need the largest two intensities to estimate the $\underline{p_{c_x}}$. Thus if there is only one linear layer as the last layer in the entire network (as in most Resnet like models), this can be further simplified: it needs computing a 2×2 covariance matrix instead of the $C \times C$ covariance matrix.

Linear Layer

If the network consists of multiple linear layers, calculating the moments of the subsequent linear layers is performed differently. Since it contains only 1-D inputs, we can either treat it spatially or channel-wise. In our setup, we consider it as along the channels.

Let $\mathbf{x} \in \mathbb{R}^{N_i}$ be the input of the i^{th} linear layer, where $i > 1$. Assume, $\mathbf{x} \sim \mathcal{N}\left(\boldsymbol{\mu}_{\mathbf{x}}^{(i)}, \Sigma_{\mathbf{x}}^{(i)}\right)$. Given the i^{th} linear layer with parameter $(W_i, \mathbf{b}_i)$, with the output

given by

$$\mathbf{h} = W_i^T \mathbf{x} + \mathbf{b}_i$$

$$\mathbf{h} \sim \mathcal{N}\left(W_i^T \boldsymbol{\mu}_x^{(i)} + \mathbf{b}_i, W_i^T \Sigma_x^{(i)} W_i\right)$$

Here, $W_i \in \mathbb{R}^{N_i \times N_{i+1}}$ and $\mathbf{b}_i \in \mathbb{R}^{N_{i+1}}$ and $\mathbf{h} \in \mathbb{R}^{N_{i+1}}$.

Pooling Layer

Recall that the input of a max pooling layer is $\{\mathbf{x}_i\}$ where each $\mathbf{x}_i \in \mathbb{R}^{N_{in}}$ and the index i varies over the spatial dimension. Observe that as we identify each $\mathbf{x}_i$ by the respective distribution $\mathcal{N}(\boldsymbol{\mu}_i, \Sigma)$, applying max pooling over $\mathbf{x}_i$ essentially requires computing the maximum over $\{\mathcal{N}(\boldsymbol{\mu}_i, \Sigma)\}$. Thus, we restrict ourselves to average pooling. To be precise, with a kernel window $\mathbb{W}$ of size $k \times k$ with stride k used for average pooling, the output of average pooling, denoted by

$$\mathbf{h} \sim \mathcal{N}\left(\frac{1}{k^2} \sum_{\mathbf{x}_i \in \mathbb{W}} \boldsymbol{\mu}_i, \frac{\Sigma}{k^2}\right)$$

Normalization Layer

For the normalization layer, given by $\mathbf{h} = (\mathbf{x} - \mu)/\sigma$, where μ, σ can be computed in different ways Ioffe and Szegedy (2015); Ba et al. (2016); Ulyanov et al. (2016), we have

$$\mathbf{h} \sim \mathcal{N}\left(\frac{\boldsymbol{\mu}_x^{(i)} - \mu}{\sigma}, \frac{\Sigma_x^{(i)}}{\sigma^2}\right)$$

However, as the normalization layers often have large Lipschitz constant Awais et al. (2020), we remove those layers in this work.

Batch normalization. For the batch normalization, the mean and variance are computed within each mini-batch Ioffe and Szegedy (2015).

$$\mu = \frac{1}{m} \sum_{i=1}^{m} x_i, \sigma^2 = \frac{1}{m} \sum_{i=1}^{m} (x_i - \mu)^2$$

where m is the size of the mini-batch. One thing to notice that $\mu, \sigma \in \mathbb{R}^N$ when $x_i \in \mathbb{R}^N$, N is the channel size. In this setting, the way to compute the bounded distribution of output will need to be modified as following:

$$\mathbf{h} \sim \mathcal{N}\left(\frac{\boldsymbol{\mu}_{\mathbf{x}}^{(i)} - \mu}{\sigma}, \frac{\Sigma_{\mathbf{x}}^{(i)}}{\sigma\sigma^T}\right)$$

where "/" is the element-wise divide. $\sigma\sigma^T \in \mathbb{R}^{N\times N}$. And μ, σ will be computed dynamically as the new data being fed into the network.

Act normalization. As the performance for batch normalization being different between training and testing period, there are several other types of normalization layers. Kingma and Dhariwal (2018) proposed an act normalization layer where μ, σ are trainable parameters. These two parameters are initialized during warm-up period to compute the mean and variance of the training dataset. After initialization, these parameters are trained freely.

In this case, the update rule is the same as above. The only difference is that μ, σ do not depend on the data being fed into the network.

Activation Layer

ReLU. In Bibi et al. (2018); Lee et al. (2019), the authors introduced a way to compute the mean and variance after ReLU. Since ReLU is an element-wise operation, assume $x \sim \mathcal{N}(\mu, \sigma^2)$. After ReLU activation, the first and second moments of the output are given by:

$$\mathbb{E}(\text{ReLU}(x)) = \frac{1}{2}\mu - \frac{1}{2}\mu\,\text{erf}(\frac{-\mu}{\sqrt{2}\sigma}) + \frac{1}{\sqrt{2\pi}}\sigma\exp(-\frac{\mu^2}{2\sigma^2})$$
$$\text{var}(\text{ReLU}(x)) < \text{var}(x)$$

Here, erf is the Error function. We need to track an upper bound of the covariance matrix, so we use $\mathrm{ReLU}(x) \sim \mathcal{N}(\mu_a, \Sigma_a)$, where,

$$\mu_a = \frac{1}{2}\mu - \frac{1}{2}\mu \,\mathrm{erf}(\frac{-\mu}{\sqrt{2}\sigma}) + \frac{1}{\sqrt{2\pi}}\sigma \exp(-\frac{\mu^2}{2\sigma^2}),$$
$$\Sigma_a \preceq \Sigma$$

Last layer/prediction: Here, the last layer is the layer before softmax layer, which represents the "strength" of the model for the label l. By Observation 5.1, we have the estimation of

$$p_{c_x} = \underline{p_{c_x}} = \Phi(\frac{\boldsymbol{\mu}[c_x] - \boldsymbol{\mu}[\widetilde{c}]}{\sqrt{\Sigma[c_x, c_x] + \Sigma[\widetilde{c}, \widetilde{c}] - 2\Sigma[c_x, \widetilde{c}]}})$$

and

$$p_{\widetilde{c}} = \overline{p_{\widetilde{c}}} = 1 - p_{c_x}$$

By Theorem 1, the certified radius is

$$\begin{aligned} C_R &= \frac{\sigma}{2}(\Phi^{-1}(\underline{p_{c_x}}) - \Phi^{-1}(\overline{p_{\widetilde{c}}})) \\ &= \sigma \frac{\boldsymbol{\mu}[c] - \boldsymbol{\mu}[\widetilde{c}]}{\sqrt{\Sigma[c_x, c_x] + \Sigma[\widetilde{c}, \widetilde{c}] - 2\Sigma[c_x, \widetilde{c}]}} \end{aligned}$$

Other activation functions. Other than ReLU, there is no closed form to compute the mean after the activation function. One simplification can be: use the local linear function to approximate the activation function, as Gowal et al. (2018) did. The good piece of this approximation is that the output covariance matrix Σ_a can be bounded by $\alpha^2\Sigma$, where α is the largest slope of the linear function.

Another direction is to apply the Hermite expansion Lokhande et al. (2020b) which will add more parameters but is theoretically sound.

As ReLU is the most well-used activation function as well as the elegant closed form mean after the activation function, in this paper, we only focus on the ReLU layer and hold other activation functions into future work.

A.2 Strengths and Limitations of Our Method

The biggest benefit of our method is the training time, which is also the main focus of Chapter 5. As in Section 5.4, we have shown that our methods can be $5\times$ faster on Cifar-10, etc. dataset, with a comparable ACR as MACER. In real-world applications, training speed is an important consideration. Thus, our method is a cheaper substitute of MACER with a marginal performance compromise.

On the other hand, we also need to discuss that under which circumstances our method does not perform well. The first case is when the network is extremely deep, e.g., Resnet 101. Due to the nature of upper bound, the estimation of the second moments tends to become looser as the network grows deeper. Thus, this will lead to a looser estimation of the distribution of the last layer and the robustness estimation would be less meaningful. Another minor weakness is when the input perturbation is large, for example $\sigma = 1.0$. As shown in the main paper, the ACR drops from 0.56 to 0.52 on ImageNet when the noise perturbation increases from $\sigma = 0.5$ to $\sigma = 1.0$. The main reason relates to the assumption of a Gaussian distribution. As the perturbation grows larger, the number of channels, by the central limit theorem, should also be larger to satisfy the Gaussian distribution. Thus, when the network structure is fixed, there is an inherent limit on the input perturbation.

We note that to perform a fair comparison, we run Cohen et al. (2019), MACER Zhai et al. (2020), and our method based on the PreActResnet 18 for Table 5.3. Since the network is shallower than the original MACER paper, the performance numbers reported here are lower. As described above, a large perturbation $\sigma = 1.0$ leads to small drop in performance. Thus, almost for all three methods, the ACR for $\sigma = 1.0$ is worse than the one for $\sigma = 0.5$ on ImageNet. For Places365 dataset, the results are slightly better on $\sigma = 1.0$ than $\sigma = 0.5$.

Also, similar as Chapter 5, we statistically test the variance based on MC sampling and our upper bound tracking method. The results are shown in Table A.1. As one can see that when the network gets deeper, the upper bound tends to be looser. But in most case, the upper bound is around 3 times higher than the sample-based

variance, which is affordable in the real-world application.

Table A.1: Statistics for different layers of MC sampling and our upper bound tracking method for deeper network.

Layer number	1	9	17	25	33
MC (1000 samples)	0.595	0.782	2.751	7.692	0.712
Upper bound	0.685	3.231	5.583	22.546	3.960

REFERENCES

Achille, Alessandro, and Stefano Soatto. 2018. Emergence of invariance and disentanglement in deep representations. *J. Mach. Learn. Res.* 19:50:1–50:34.

Afsari, Bijan, Rizwan Chaudhry, Avinash Ravichandran, and René Vidal. 2012. Group action induced distances for averaging and clustering linear dynamical systems with applications to the analysis of dynamic scenes. In *Proceedings of Computer Vision and Pattern Recognition, CVPR 2012*, 2208–2215. IEEE.

Aganj, Iman, Christophe Lenglet, and Guillermo Sapiro. 2009. ODF reconstruction in q-ball imaging with solid angle consideration. In *Proceedings of IEEE International Symposium on Biomedical Imaging, ISBI 2009*, 1398–1401. IEEE.

Ahmed, Mahtab, Muhammad Rifayat Samee, and Robert E. Mercer. 2019. Improving tree-lstm with tree attention. In *Proceedings of IEEE International Conference on Semantic Computing, ICSC 2019*, 247–254. IEEE.

Akash, Aditya Kumar, Vishnu Suresh Lokhande, Sathya N. Ravi, and Vikas Singh. 2021. Learning invariant representations using inverse contrastive loss. In *Proceedings of the 35th AAAI Conference on Artificial Intelligence, AAAI 2021*, 6582–6591. AAAI Press.

Akhtar, Nadeem, and Mohd Vasim Ahamad. 2021. Graph tools for social network analysis. In *Research anthology on digital transformation, organizational change, and the impact of remote work*, 485–500. IGI Global.

Alexander, Andrew L., Jee Eun Lee, Mariana Lazar, and Aaron S. Field. 2007. Diffusion tensor imaging of the brain. *Neurotherapeutics* 4(3):316–329.

Alger, Jeffry R. 2012. The diffusion tensor imaging toolbox. *The Journal of Neuroscience* 32(22):7418–7428.

Amari, Shun-ichi, and Si Wu. 1999. Improving support vector machine classifiers by modifying kernel functions. *Neural Networks* 12(6):783–789.

Anctil-Robitaille, Benoit, Christian Desrosiers, and Herve Lombaert. 2020. Manifold-aware cyclegan for high resolution structural-to-dti synthesis. *CoRR* abs/2004.00173. 2004.00173.

Anderson, Brandon M., Truong-Son Hy, and Risi Kondor. 2019. Cormorant: Covariant molecular neural networks. In *Advances in Neural Information Processing Systems 32, NeurIPS 2019*, 14510–14519.

Anderson, Theodore W. 1958. An introduction to multivariate statistical analysis. Tech. Rep.

Andersson, Jesper L. R., and Stamatios N. Sotiropoulos. 2016. An integrated approach to correction for off-resonance effects and subject movement in diffusion MR imaging. *NeuroImage* 125:1063–1078.

Andrew, Galen, Raman Arora, Jeff A. Bilmes, and Karen Livescu. 2013. Deep canonical correlation analysis. In *Proceedings of the 30th International Conference on Machine Learning, ICML 2013*, vol. 28 of *JMLR Workshop and Conference Proceedings*, 1247–1255. JMLR.org.

Angluin, Dana, and Philip D. Laird. 1987. Learning from noisy examples. *Mach. Learn.* 2(4):343–370.

Antoniou, Antreas, Amos J. Storkey, and Harrison Edwards. 2017. Data augmentation generative adversarial networks. *CoRR* abs/1711.04340. 1711.04340.

Arjovsky, Martín, Soumith Chintala, and Léon Bottou. 2017. Wasserstein generative adversarial networks. In *Proceedings of the 34th International Conference on Machine Learning, ICML 2017*, vol. 70 of *Proceedings of Machine Learning Research*, 214–223. PMLR.

Ashburner, John, and Karl J. Friston. 2000. Voxel-based morphometry—The methods. *NeuroImage* 11(6):805–821.

Athalye, Anish, and Nicholas Carlini. 2018. On the robustness of the CVPR 2018 white-box adversarial example defenses. *CoRR* abs/1804.03286. 1804.03286.

Awais, Muhammad, Fahad Shamshad, and Sung-Ho Bae. 2020. Towards an adversarially robust normalization approach. *CoRR* abs/2006.11007. 2006.11007.

Ba, Lei Jimmy, Jamie Ryan Kiros, and Geoffrey E. Hinton. 2016. Layer normalization. *CoRR* abs/1607.06450. 1607.06450.

Bach, Francis R, and Michael I Jordan. 2005. A probabilistic interpretation of canonical correlation analysis.

Bai, Shaojie, J. Zico Kolter, and Vladlen Koltun. 2018. Convolutional sequence modeling revisited. In *6th International Conference on Learning Representations, ICLR 2018, workshop track proceedings*. OpenReview.net.

Balunovic, Mislav, Maximilian Baader, Gagandeep Singh, Timon Gehr, and Martin T. Vechev. 2019. Certifying geometric robustness of neural networks. In *Advances in Neural Information Processing Systems 32, NeurIPS 2019*, 15287–15297.

Basser, P.J., J. Mattiello, and D. LeBihan. 1994. MR diffusion tensor spectroscopy and imaging. *Biophysical Journal* 66(1):259–267.

Batra, Dhruv, Payman Yadollahpour, Abner Guzmán-Rivera, and Gregory Shakhnarovich. 2012. Diverse m-best solutions in markov random fields. In *Proceedings of the European Conference on Computer Vision, ECCV 2012*, vol. 7576, 1–16. Springer.

Bdeir, Ahmad, Kristian Schwethelm, and Niels Landwehr. 2023. Hyperbolic geometry in computer vision: A novel framework for convolutional neural networks. *CoRR* abs/2303.15919. 2303.15919.

Bengio, Yoshua, Yann LeCun, et al. 2007. Scaling learning algorithms towards ai. *Large-scale kernel machines* 34(5):1–41.

Bibi, Adel, Modar Alfadly, and Bernard Ghanem. 2018. Analytic expressions for probabilistic moments of PL-DNN with gaussian input. In *Proceedings of Computer Vision and Pattern Recognition, CVPR 2018*, 9099–9107. IEEE.

Biggio, Battista, Igino Corona, Davide Maiorca, Blaine Nelson, Nedim Srndic, Pavel Laskov, Giorgio Giacinto, and Fabio Roli. 2013. Evasion attacks against machine learning at test time. In *Machine Learning and Knowledge Discovery in Databases: European Conference, ECML PKDD 2013*, vol. 8190, 387–402. Springer.

Bissacco, Alessandro, Alessandro Chiuso, Yi Ma, and Stefano Soatto. 2001. Recognition of human gaits. In *Proceedings of Computer Vision and Pattern Recognition CVPR 2001*, 52–57. IEEE.

Bojarski, Mariusz, Davide Del Testa, Daniel Dworakowski, Bernhard Firner, Beat Flepp, Prasoon Goyal, Lawrence D. Jackel, Mathew Monfort, Urs Muller, Jiakai Zhang, Xin Zhang, Jake Zhao, and Karol Zieba. 2016. End to end learning for self-driving cars. *CoRR* abs/1604.07316. `1604.07316`.

Boothby, William M. 1986. *An introduction to differentiable manifolds and Riemannian geometry*. 2nd ed. Pure and applied mathematics v. 120, Academic Press.

Bottou, Léon, and Olivier Bousquet. 2007. The tradeoffs of large scale learning. In *Advances in Neural Information Processing Systems 20, NIPS 2007*, 161–168.

Brehmer, Johann, and Kyle Cranmer. 2020. Flows for simultaneous manifold learning and density estimation. In *Advances in Neural Information Processing Systems 33, NeurIPS 2020*.

Brody, Dorje C., and Lane P. Hughston. 1998. Statistical geometry in quantum mechanics. *Proceedings of the Royal Society of London. Series A: Mathematical, Physical and Engineering Sciences* 454(1977):2445–2475.

Bromley, Jane, James W. Bentz, Léon Bottou, Isabelle Guyon, Yann LeCun, Cliff Moore, Eduard Säckinger, and Roopak Shah. 1993. Signature verification using A "siamese" time delay neural network. *Int. J. Pattern Recognit. Artif. Intell.* 7(4): 669–688.

Bronstein, Michael M., Joan Bruna, Yann LeCun, Arthur Szlam, and Pierre Vandergheynst. 2017. Geometric deep learning: Going beyond euclidean data. *IEEE Signal Process. Mag.* 34(4):18–42.

Carlini, Nicholas, and David A. Wagner. 2017a. Adversarial examples are not easily detected: Bypassing ten detection methods. In *Proceedings of the 10th ACM Workshop on Artificial Intelligence and Security, AISec@CCS 2017*, 3–14. ACM.

———. 2017b. Towards evaluating the robustness of neural networks. In *IEEE Symposium on Security and Privacy, SP 2017*, 39–57. IEEE.

Carmo, Manfredo Perdigão Do. 1992. *Riemannian geometry*. Boston, MA: Birkhäuser Boston.

Caron, Mathilde, Ishan Misra, Julien Mairal, Priya Goyal, Piotr Bojanowski, and Armand Joulin. 2020. Unsupervised learning of visual features by contrasting cluster assignments. In *Advances in Neural Information Processing Systems 33, NeurIPS 2020*.

Carrazza, Stefano, and Juan Cruz-Martinez. 2019. Towards a new generation of parton densities with deep learning models. *The European Physical Journal C* 79(8): 676.

Casella, George, and Roger L. Berger. 2002. *Statistical inference*. 2nd ed. Australia ; Pacific Grove, CA: Thomson Learning.

Chakraborty, Rudrasis, Monami Banerjee, and Baba C. Vemuri. 2018a. A CNN for homogneous Riemannian manifolds with applications to Neuroimaging. *CoRR* abs/1805.05487. `1805.05487`.

———. 2018b. H-cnns: Convolutional neural networks for riemannian homogeneous spaces. *CoRR* abs/1805.05487. `1805.05487`.

Chakraborty, Rudrasis, Jose Bouza, Jonathan H. Manton, and Baba C. Vemuri. 2022. Manifoldnet: A deep neural network for manifold-valued data with applications. *IEEE Trans. Pattern Anal. Mach. Intell.* 44(2):799–810.

Chakraborty, Rudrasis, Chun-Hao Yang, Xingjian Zhen, Monami Banerjee, Derek Archer, David E. Vaillancourt, Vikas Singh, and Baba C. Vemuri. 2018c. A statistical recurrent model on the manifold of symmetric positive definite matrices. In *Advances in Neural Information Processing Systems 31, NeurIPS 2018*, 8897–8908.

Chami, Ines, Albert Gu, Vaggos Chatziafratis, and Christopher Ré. 2020a. From trees to continuous embeddings and back: Hyperbolic hierarchical clustering. In *Advances in Neural Information Processing Systems 33, NeurIPS 2020*.

Chami, Ines, Adva Wolf, Da-Cheng Juan, Frederic Sala, Sujith Ravi, and Christopher Ré. 2020b. Low-dimensional hyperbolic knowledge graph embeddings. In *Proceedings of Association for Computational Linguistics, ACL 2020*, 6901–6914. Association for Computational Linguistics.

Chan, Alvin, Yi Tay, and Yew-Soon Ong. 2020. What it thinks is important is important: Robustness transfers through input gradients. In *Proceedings of Computer Vision and Pattern Recognition, CVPR 2020*, 329–338. IEEE.

Chari, Tara, and Lior Pachter. 2022. The specious art of single-cell genomics.

Charikar, Moses. 2002. Similarity estimation techniques from rounding algorithms. In *Proceedings on 34th Annual ACM Symposium on Theory of Computing*, 380–388. ACM.

Chen, Beidi, Zichang Liu, Binghui Peng, Zhaozhuo Xu, Jonathan Lingjie Li, Tri Dao, Zhao Song, Anshumali Shrivastava, and Christopher Ré. 2021. MONGOOSE: A learnable LSH framework for efficient neural network training. In *9th International Conference on Learning Representations, ICLR 2021*. OpenReview.net.

Chen, Beidi, Tharun Medini, James Farwell, Sameh Gobriel, Tsung-Yuan Charlie Tai, and Anshumali Shrivastava. 2020. SLIDE : In defense of smart algorithms over hardware acceleration for large-scale deep learning systems. In *Proceedings of Machine Learning and Systems, MLSys 2020*. mlsys.org.

Chen, Tian Qi, Xuechen Li, Roger B. Grosse, and David Duvenaud. 2018. Isolating sources of disentanglement in variational autoencoders. In *Advances in Neural Information Processing Systems 31, NeurIPS 2018*, 2615–2625.

Chen, Xinlei, and Kaiming He. 2021. Exploring simple siamese representation learning. In *Proceedings of Computer Vision and Pattern Recognition, CVPR 2021*, 15750–15758. IEEE.

Cheng, Guang, Hesamoddin Salehian, and Baba C. Vemuri. 2012. Efficient recursive algorithms for computing the mean diffusion tensor and applications to DTI segmentation. In *Proceedings of the European Conference on Computer Vision, ECCV 2012*, vol. 7578, 390–401. Springer.

Cheng, Shuyu, Yinpeng Dong, Tianyu Pang, Hang Su, and Jun Zhu. 2019. Improving black-box adversarial attacks with a transfer-based prior. In *Advances in Neural Information Processing Systems 32, NeurIPS 2019*, 10932–10942.

Chirikjian, Gs, Ab Kyatkin, and Ac Buckingham. 2001. Engineering applications of noncommutative harmonic analysis: with emphasis on rotation and motion groups. *Applied Mechanics Reviews* 54(6):B97–B98.

Cho, Jaewoong, Gyeongjo Hwang, and Changho Suh. 2020. A fair classifier using mutual information. In *IEEE International Symposium on Information Theory, ISIT 2020*, 2521–2526. IEEE.

Cohen, Jeremy M., Elan Rosenfeld, and J. Zico Kolter. 2019. Certified adversarial robustness via randomized smoothing. In *Proceedings of the 36th International Conference on Machine Learning, ICML 2019*, vol. 97 of *Proceedings of Machine Learning Research*, 1310–1320. PMLR.

Cohen, Taco, and Max Welling. 2016a. Group equivariant convolutional networks. In *Proceedings of the 33nd International Conference on Machine Learning, ICML 2016*, vol. 48 of *JMLR Workshop and Conference Proceedings*, 2990–2999. JMLR.org.

Cohen, Taco S., Mario Geiger, Jonas Köhler, and Max Welling. 2018. Spherical cnns. In *6th International Conference on Learning Representations, ICLR 2018*. OpenReview.net.

Cohen, Taco S., and Max Welling. 2016b. Steerable cnns. *CoRR* abs/1612.08498. 1612.08498.

Cover, Thomas M., and Joy A. Thomas. 2012. *Elements of Information Theory*. John Wiley & Sons.

Cowley, Benjamin, João D. Semedo, Amin Zandvakili, Matthew A. Smith, Adam Kohn, and Byron M. Yu. 2017. Distance covariance analysis. In *Proceedings of the 20th International Conference on Artificial Intelligence and Statistics, AISTATS 2017*, vol. 54 of *Proceedings of Machine Learning Research*, 242–251. PMLR.

D'Angelo, Francesco, and Vincent Fortuin. 2021. Repulsive deep ensembles are bayesian. In *Advances in Neural Information Processing Systems 34, NeurIPS 2021*, 3451–3465.

D'Angelo, Francesco, Vincent Fortuin, and Florian Wenzel. 2021. On stein variational neural network ensembles. *CoRR* abs/2106.10760. 2106.10760.

Datar, Mayur, Nicole Immorlica, Piotr Indyk, and Vahab S. Mirrokni. 2004. Locality-sensitive hashing scheme based on p-stable distributions. In *Proceedings of the 20th ACM Symposium on Computational Geometry*, 253–262. ACM.

David, Laurianne, Amol Thakkar, Rocío Mercado, and Ola Engkvist. 2020. Molecular representations in ai-driven drug discovery: a review and practical guide. *J. Cheminformatics* 12(1):56.

DeGroot, Morris H., and Mark J. Schervish. 2002. *Probability and statistics*. 3rd ed. Addison Wesley.

Demontis, Ambra, Marco Melis, Maura Pintor, Matthew Jagielski, Battista Biggio, Alina Oprea, Cristina Nita-Rotaru, and Fabio Roli. 2019. Why do adversarial attacks

transfer? explaining transferability of evasion and poisoning attacks. In *28th USENIX Security Symposium, USENIX Security 2019*, 321–338. USENIX Association.

Deng, Jia, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. 2009. Imagenet: A large-scale hierarchical image database. In *Proceedings of Computer Vision and Pattern Recognition CVPR 2009*, 248–255. IEEE.

Devlin, Jacob, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2019. BERT: pre-training of deep bidirectional transformers for language understanding. In *Proceedings of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, NAACL-HLT 2019*, 4171–4186. Association for Computational Linguistics.

Dhingra, Bhuwan, Christopher J. Shallue, Mohammad Norouzi, Andrew M. Dai, and George E. Dahl. 2018. Embedding text in hyperbolic spaces. In *Proceedings of the 12th Workshop on Graph-Based Methods for Natural Language Processing, TextGraphs@NAACL-HLT 2018*, 59–69. Association for Computational Linguistics.

Dinh, Laurent, Jascha Sohl-Dickstein, and Samy Bengio. 2017. Density estimation using real NVP. In *5th International Conference on Learning Representations, ICLR 2017*. OpenReview.net.

Doersch, Carl. 2016. Tutorial on variational autoencoders. *CoRR* abs/1606.05908. `1606.05908`.

Dominici, F. 2002. On the use of generalized additive models in time-series studies of air pollution and health. *American Journal of Epidemiology* 156(3):193–203.

Duvenaud, David, Dougal Maclaurin, Jorge Aguilera-Iparraguirre, Rafael Gómez-Bombarelli, Timothy Hirzel, Alán Aspuru-Guzik, and Ryan P. Adams. 2015. Convolutional networks on graphs for learning molecular fingerprints. In *Advances in Neural Information Processing Systems 28, NIPS 2015*, 2224–2232.

Elman, Jeffrey L. 1990. Finding structure in time. *Cogn. Sci.* 14(2):179–211.

Elsaid, Nahla M. H., and Yu-Chien Wu. 2019. Super-resolution diffusion tensor imaging using SRCNN: A feasibility study. In *Proceedings of IEEE Engineering in Medicine and Biology Society, EMBC 2019*, 2830–2834. IEEE.

Essen, David C. Van, Stephen M. Smith, Deanna M. Barch, Timothy Edward John Behrens, Essa Yacoub, and Kâmil Ugurbil. 2013. The WU-Minn human connectome project: An overview. *NeuroImage* 80:62–79.

Fan, Wenfei. 2012. Graph pattern matching revised for social network analysis. In *15th International Conference on Database Theory, ICDT 2012*, 8–21. ACM.

Fan, Wenqi, Yao Ma, Qing Li, Yuan He, Yihong Eric Zhao, Jiliang Tang, and Dawei Yin. 2019. Graph neural networks for social recommendation. In *The World Wide Web Conference, WWW 2019*, 417–426. ACM.

Fan, Xiran, Chun-Hao Yang, and Baba C. Vemuri. 2022. Nested hyperbolic spaces for dimensionality reduction and hyperbolic NN design. In *Proceedings of Computer Vision and Pattern Recognition, CVPR 2022*, 356–365. IEEE.

Fathi, Albert, and Alessio Figalli. 2010. Optimal transportation on non-compact manifolds. *Israel Journal of Mathematics* 175(1):1–59.

Feldman, Michael, Sorelle A. Friedler, John Moeller, Carlos Scheidegger, and Suresh Venkatasubramanian. 2015. Certifying and removing disparate impact. In *Proceedings of the 21th ACM International Conference on Knowledge Discovery and Data Mining, SIGKDD 2015*, 259–268. ACM.

Fontaine, Matthew C., and Stefanos Nikolaidis. 2021. Differentiable quality diversity. In *Advances in Neural Information Processing Systems 34, NeurIPS 2021*, 10040–10052.

Fromer, Menachem, and Amir Globerson. 2009. An LP view of the m-best MAP problem. In *Advances in Neural Information Processing Systems 22, NIPS 2009*, 567–575.

Fukushima, Kunihiko, Sei Miyake, and Takayuki Ito. 1983. Neocognitron: A neural network model for a mechanism of visual pattern recognition. *IEEE Trans. Syst. Man Cybern.* 13(5):826–834.

Gabbay, Aviv, Niv Cohen, and Yedid Hoshen. 2021. An image is worth more than a thousand words: Towards disentanglement in the wild. In *Advances in Neural Information Processing Systems 34, NeurIPS 2021*, 9216–9228.

Gaier, Adam, Alexander Asteroth, and Jean-Baptiste Mouret. 2020. Discovering representations for black-box optimization. In *GECCO '20: Genetic and evolutionary computation conference, cancún mexico, july 8-12, 2020*, 103–111. ACM.

Ganea, Octavian-Eugen, Gary Bécigneul, and Thomas Hofmann. 2018. Hyperbolic neural networks. In *Advances in Neural Information Processing Systems 31, NeurIPS 2018*, 5350–5360.

Gao, Chao, Dan Garber, Nathan Srebro, Jialei Wang, and Weiran Wang. 2019. Stochastic canonical correlation analysis. *J. Mach. Learn. Res.* 20:167:1–167:46.

Garyfallidis, Eleftherios, Matthew Brett, Bagrat Amirbekian, Ariel Rokem, Stéfan van der Walt, Maxime Descoteaux, and Ian Nimmo-Smith. 2014. Dipy, a library for the analysis of diffusion MRI data. *Frontiers Neuroinformatics* 8:8.

Geirhos, Robert, Patricia Rubisch, Claudio Michaelis, Matthias Bethge, Felix A. Wichmann, and Wieland Brendel. 2019. Imagenet-trained cnns are biased towards texture; increasing shape bias improves accuracy and robustness. In *7th International Conference on Learning Representations, ICLR 2019*. OpenReview.net.

Gemp, Ian M., Charlie Chen, and Brian McWilliams. 2022. The generalized eigenvalue problem as a nash equilibrium. *CoRR* abs/2206.04993. 2206.04993.

Glorot, Xavier, and Yoshua Bengio. 2010. Understanding the difficulty of training deep feedforward neural networks. In *Proceedings of the 13th International Conference on Artificial Intelligence and Statistics, AISTATS 2010*, vol. 9 of *JMLR Proceedings*, 249–256. JMLR.org.

Goldberger, Jacob, and Ehud Ben-Reuven. 2017. Training deep neural-networks using a noise adaptation layer. In *5th International Conference on Learning Representations, ICLR 2017*. OpenReview.net.

Good, Phillip I. 2000. *Permutation tests: a practical guide to resampling methods for testing hypotheses*. 2nd ed. New York: Springer. OCLC: 681912126.

Goodfellow, Ian J., Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron C. Courville, and Yoshua Bengio. 2014. Generative adversarial nets. In *Advances in Neural Information Processing Systems 27, NIPS 2014*, 2672–2680.

Goodfellow, Ian J., Jonathon Shlens, and Christian Szegedy. 2015. Explaining and harnessing adversarial examples. In *3rd International Conference on Learning Representations, ICLR 2015*.

Gowal, Sven, Krishnamurthy Dvijotham, Robert Stanforth, Rudy Bunel, Chongli Qin, Jonathan Uesato, Relja Arandjelovic, Timothy A. Mann, and Pushmeet Kohli. 2018. On the effectiveness of interval bound propagation for training verifiably robust models. *CoRR* abs/1810.12715. `1810.12715`.

Grill, Jean-Bastien, Florian Strub, Florent Altché, Corentin Tallec, Pierre H. Richemond, Elena Buchatskaya, Carl Doersch, Bernardo Ávila Pires, Zhaohan Guo, Mohammad Gheshlaghi Azar, Bilal Piot, Koray Kavukcuoglu, Rémi Munos, and Michal Valko. 2020. Bootstrap your own latent - A new approach to self-supervised learning. In *Advances in Neural Information Processing Systems 33, NeurIPS 2020*.

Groisser, David. 2004. Newton's method, zeroes of vector fields, and the riemannian center of mass. *Adv. Appl. Math.* 33(1):95–135.

Guo, Zhiwei, and Heng Wang. 2021. A deep graph neural network-based mechanism for social recommendations. *IEEE Trans. Ind. Informatics* 17(4):2776–2783.

Guzmán-Rivera, Abner, Pushmeet Kohli, Dhruv Batra, and Rob A. Rutenbar. 2014. Efficiently enforcing diversity in multi-output structured prediction. In *Proceedings*

of the 17th International Conference on Artificial Intelligence and Statistics, AISTATS 2014, vol. 33 of *JMLR Workshop and Conference Proceedings*, 284–292. JMLR.org.

Han, Bo, Quanming Yao, Xingrui Yu, Gang Niu, Miao Xu, Weihua Hu, Ivor W. Tsang, and Masashi Sugiyama. 2018. Co-teaching: Robust training of deep neural networks with extremely noisy labels. In *Advances in Neural Information Processing Systems 31, NeurIPS 2018*, 8536–8546.

Hanebeck, U.D., K. Briechle, and J. Horn. 2001. A tight bound for the joint covariance of two random vectors with unknown but constrained cross-correlation. In *Conference Documentation International Conference on Multisensor Fusion and Integration for Intelligent Systems. MFI 2001 (Cat. No.01TH8590)*, 85–90. Baden-Baden, Germany: VDI/VDE Soc. Meas. & Autom. Control.

Hardoon, David R., Sándor Szedmák, and John Shawe-Taylor. 2004. Canonical correlation analysis: An overview with application to learning methods. *Neural Comput.* 16(12):2639–2664.

He, Jun, Laura Balzano, and Arthur Szlam. 2012. Incremental gradient on the grassmannian for online foreground and background separation in subsampled video. In *Proceedings of Computer Vision and Pattern Recognition, CVPR 2012*, 1568–1575. IEEE.

He, Kaiming, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. 2016a. Deep residual learning for image recognition. In *Proceedings of Computer Vision and Pattern Recognition, CVPR 2016*, 770–778. IEEE.

———. 2016b. Identity mappings in deep residual networks. In *Proceedings of the European Conference on Computer Vision, ECCV 2016*, vol. 9908, 630–645. Springer.

Henderson, Ryan, and Rasmus Rothe. 2017. Picasso: A modular framework for visualizing the learning process of neural network image classifiers. *Journal of Open Research Software* 5(1):22.

Hess, Christopher P., Pratik Mukherjee, Eric T. Han, Duan Xu, and Daniel B. Vigneron. 2006. Q-ball reconstruction of multimodal fiber orientations using the spherical harmonic basis. *Magnetic Resonance in Medicine* 56(1):104–117.

Higgins, Irina, Loïc Matthey, Arka Pal, Christopher P. Burgess, Xavier Glorot, Matthew M. Botvinick, Shakir Mohamed, and Alexander Lerchner. 2017. beta-vae: Learning basic visual concepts with a constrained variational framework. In *5th International Conference on Learning Representations, ICLR 2017*. OpenReview.net.

Hinton, Geoffrey E. 1989. Connectionist learning procedures. *Artif. Intell.* 40(1-3): 185–234.

Ho, Jeffrey, Guang Cheng, Hesamoddin Salehian, and Baba C. Vemuri. 2013. Recursive karcher expectation estimators and geometric law of large numbers. In *Proceedings of the 16th International Conference on Artificial Intelligence and Statistics, AISTATS 2013*, vol. 31 of *JMLR Workshop and Conference Proceedings*, 325–332. JMLR.org.

Hochreiter, Sepp, and Jürgen Schmidhuber. 1997. Long short-term memory. *Neural Comput.* 9(8):1735–1780.

Hotelling, Harold. 1936. Relations between two sets of variates. *Biometrika* 28(3/4): 321.

Howard, Andrew, Ruoming Pang, Hartwig Adam, Quoc V. Le, Mark Sandler, Bo Chen, Weijun Wang, Liang-Chieh Chen, Mingxing Tan, Grace Chu, Vijay Vasudevan, and Yukun Zhu. 2019. Searching for mobilenetv3. In *Proceedings of International Conference on Computer Vision, ICCV 2019*, 1314–1324. IEEE.

Hsu, Joy, Jeffrey Gu, Gong Her Wu, Wah Chiu, and Serena Yeung. 2021. Capturing implicit hierarchical structure in 3d biomedical images with self-supervised hyperbolic representations. In *Advances in Neural Information Processing Systems 34, NeurIPS 2021*, 5112–5123.

Hua, Xue, Alex D. Leow, Neelroop Parikshak, Suh Lee, Ming-Chang Chiang, Arthur W. Toga, Clifford R. Jack Jr., Michael W. Weiner, and Paul M. Thompson. 2008. Tensor-based morphometry as a neuroimaging biomarker for Alzheimer's disease: An MRI study of 676 ad, mci, and normal subjects. *NeuroImage* 43(3): 458–469.

Huang, Ruitong, Bing Xu, Dale Schuurmans, and Csaba Szepesvári. 2015. Learning with a strong adversary. *CoRR* abs/1511.03034. `1511.03034`.

Huang, Zhichun, Rudrasis Chakraborty, and Vikas Singh. 2022. Forward operator estimation in generative models with kernel transfer operators. In *Proceedings of the 39th International Conference on Machine Learning, ICML 2022*, vol. 162 of *Proceedings of Machine Learning Research*, 9148–9172. PMLR.

Huang, Zhiwu, and Luc Van Gool. 2017. A riemannian network for SPD matrix learning. In *Proceedings of the 31st AAAI Conference on Artificial Intelligence, AAAI, 2017*, 2036–2042. AAAI Press.

Huang, Zhiwu, Jiqing Wu, and Luc Van Gool. 2018. Building deep networks on grassmann manifolds. In *Proceedings of the 32nd AAAI Conference on Artificial Intelligence, AAAI 2018*, 3279–3286. AAAI Press.

———. 2019. Manifold-valued image generation with wasserstein generative adversarial nets. In *Proceedings of the 33rd AAAI Conference on Artificial Intelligence, AAAI 2019*, 3886–3893. AAAI Press.

Härdle, Wolfgang Karl, and Léopold Simar. 2015. *Applied multivariate statistical analysis*. Berlin, Heidelberg: Springer Berlin Heidelberg.

Iandola, Forrest N., Matthew W. Moskewicz, Sergey Karayev, Ross B. Girshick, Trevor Darrell, and Kurt Keutzer. 2014. Densenet: Implementing efficient convnet descriptor pyramids. *CoRR* abs/1404.1869. `1404.1869`.

Ikonomovic, Milos D., William E. Klunk, Eric E. Abrahamson, Chester A. Mathis, Julie C. Price, Nicholas D. Tsopelas, Brian J. Lopresti, Scott Ziolko, Wenzhu Bi,

William R. Paljug, Manik L. Debnath, Caroline E. Hope, Barbara A. Isanski, Ronald L. Hamilton, and Steven T. DeKosky. 2008. Post-mortem correlates of in vivo PiB-PET amyloid imaging in a typical case of Alzheimer's disease. *Brain* 131(6):1630–1645.

Ilyas, Andrew, Logan Engstrom, Anish Athalye, and Jessy Lin. 2018. Black-box adversarial attacks with limited queries and information. In *Proceedings of the 35th International Conference on Machine Learning, ICML 2018*, vol. 80 of *Proceedings of Machine Learning Research*, 2142–2151. PMLR.

Ioffe, Sergey, and Christian Szegedy. 2015. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *Proceedings of the 32nd International Conference on Machine Learning, ICML 2015*, vol. 37 of *JMLR Workshop and Conference Proceedings*, 448–456. JMLR.org.

Jack, Clifford R, David A Bennett, Kaj Blennow, Maria C Carrillo, Billy Dunn, Samantha Budd Haeberlein, David M Holtzman, William Jagust, Frank Jessen, Jason Karlawish, et al. 2018. NIA-AA research framework: Toward a biological definition of Alzheimer's disease. *Alzheimer's & Dementia* 14(4):535–562.

Jafari, Omid, Preeti Maurya, Parth Nagarkar, Khandker Mushfiqul Islam, and Chidambaram Crushev. 2021. A survey on locality sensitive hashing algorithms and their applications. *CoRR* abs/2102.08942. 2102.08942.

Jaiswal, Ashish, Ashwin Ramesh Babu, Mohammad Zaki Zadeh, Debapriya Banerjee, and Fillia Makedon. 2020. A survey on contrastive self-supervised learning. *CoRR* abs/2011.00362. 2011.00362.

Jian, Bing, Baba C. Vemuri, Evren Özarslan, Paul R. Carney, and Thomas H. Mareci. 2007. A novel tensor distribution model for the diffusion-weighted MR signal. *NeuroImage* 37(1):164–176.

Jiang, Lu, Zhengyuan Zhou, Thomas Leung, Li-Jia Li, and Li Fei-Fei. 2018. Mentornet: Learning data-driven curriculum for very deep neural networks on corrupted

labels. In *Proceedings of the 35th International Conference on Machine Learning, ICML 2018*, vol. 80 of *Proceedings of Machine Learning Research*, 2309–2318. PMLR.

Jost, Jürgen. 2017. *Riemannian Geometry and Geometric Analysis*. Universitext, Cham: Springer International Publishing.

Jørgensen, Peter Bjørn, and Arghya Bhowmik. 2022. Equivariant graph neural networks for fast electron density estimation of molecules, liquids, and solids. *npj Computational Materials* 8(1):183.

Kanaan, Richard A., Matthew Allin, Marco Picchioni, Gareth J. Barker, Eileen Daly, Sukhwinder S. Shergill, James Woolley, and Philip K. McGuire. 2012. Gender differences in white matter microstructure. *PLoS ONE* 7(6):e38272.

Karras, Tero, Samuli Laine, and Timo Aila. 2019. A style-based generator architecture for generative adversarial networks. In *Proceedings of Computer Vision and Pattern Recognition, CVPR 2019*, 4401–4410. IEEE.

Karras, Tero, Samuli Laine, Miika Aittala, Janne Hellsten, Jaakko Lehtinen, and Timo Aila. 2020. Analyzing and improving the image quality of stylegan. In *Proceedings of Computer Vision and Pattern Recognition, CVPR 2020*, 8107–8116. IEEE.

Kauderer-Abrams, Eric. 2018. Quantifying translation-invariance in convolutional neural networks. *CoRR* abs/1801.01450. `1801.01450`.

Khrulkov, Valentin, Leyla Mirvakhabova, Evgeniya Ustinova, Ivan V. Oseledets, and Victor S. Lempitsky. 2020. Hyperbolic image embeddings. In *Proceedings of Computer Vision and Pattern Recognition, CVPR 2020*, 6417–6427. IEEE.

Kim, Hyunjik, and Andriy Mnih. 2018. Disentangling by factorising. In *Proceedings of the 35th International Conference on Machine Learning, ICML 2018*, vol. 80 of *Proceedings of Machine Learning Research*, 2654–2663. PMLR.

Kim, Hyunwoo J., Nagesh Adluru, Heemanshu Suri, Baba C. Vemuri, Sterling C. Johnson, and Vikas Singh. 2017. Riemannian nonlinear mixed effects models:

Analyzing longitudinal deformations in neuroimaging. In *Proceedings of Computer Vision and Pattern Recognition, CVPR 2017*, 5777–5786. IEEE.

Kingma, Diederik P., and Jimmy Ba. 2015. Adam: A method for stochastic optimization. In *3rd International Conference on Learning Representations, ICLR 2015*.

Kingma, Diederik P., and Prafulla Dhariwal. 2018. Glow: Generative flow with invertible 1x1 convolutions. In *Advances in Neural Information Processing Systems 31, NeurIPS 2018*, 10236–10245.

Kingma, Diederik P., and Max Welling. 2014. Auto-encoding variational bayes. In *2nd International Conference on Learning Representations, ICLR 2014*.

Kondor, Risi, Hy Truong Son, Horace Pan, Brandon M. Anderson, and Shubhendu Trivedi. 2018. Covariant compositional networks for learning graphs.

Kondor, Risi, and Shubhendu Trivedi. 2018. On the generalization of equivariance and convolution in neural networks to the action of compact groups. In *Proceedings of the 35th International Conference on Machine Learning, ICML 2018*, vol. 80 of *Proceedings of Machine Learning Research*, 2752–2760. PMLR.

Kornblith, Simon, Mohammad Norouzi, Honglak Lee, and Geoffrey E. Hinton. 2019a. Similarity of neural network representations revisited. In *Proceedings of the 36th International Conference on Machine Learning, ICML 2019*, vol. 97 of *Proceedings of Machine Learning Research*, 3519–3529. PMLR.

Kornblith, Simon, Jonathon Shlens, and Quoc V. Le. 2019b. Do better imagenet models transfer better? In *Proceedings of Computer Vision and Pattern Recognition, CVPR 2019*, 2661–2671. IEEE.

Krauskopf, Bernd, Hinke M Osinga, and Jorge Galán-Vioque. 2007. *Numerical continuation methods for dynamical systems*. Springer.

Krizhevsky, A., and G. Hinton. 2009. Learning multiple layers of features from tiny images. *Master's thesis, Department of Computer Science, University of Toronto*.

Krizhevsky, Alex, Ilya Sutskever, and Geoffrey E. Hinton. 2012. Imagenet classification with deep convolutional neural networks. In *Advances in Neural Information Processing Systems 25, NIPS 2012*, 1106–1114.

Lafferty, John D. 1988. The density manifold and configuration space quantization. *Transactions of the American Mathematical Society* 305(2):699–699.

Lample, Guillaume, Neil Zeghidour, Nicolas Usunier, Antoine Bordes, Ludovic Denoyer, and Marc'Aurelio Ranzato. 2017. Fader networks: Manipulating images by sliding attributes. In *Advances in Neural Information Processing Systems 30, NeurIPS 2017*, 5967–5976.

Lazer, David, Alex Pentland, Lada Adamic, Sinan Aral, Albert-László Barabási, Devon Brewer, Nicholas Christakis, Noshir Contractor, James Fowler, Myron Gutmann, et al. 2009. Computational social science. *Science* 323(5915):721–723.

Le Bihan, Denis, Jean-François Mangin, Cyril Poupon, Chris A Clark, Sabina Pappata, Nicolas Molko, and Hughes Chabriat. 2001. Diffusion tensor imaging: Concepts and applications. *Journal of Magnetic Resonance Imaging* 13(4):534–546.

Lebanon, Guy. 2005. *Riemannian geometry and statistical machine learning*. Carnegie Mellon University.

Lebanon, Guy, and John D. Lafferty. 2001. Boosting and maximum likelihood for exponential models. In *Advances in Neural Information Processing Systems 14, NIPS 2001*, 447–454.

LeCun, Yann. 1998. The MNIST database of handwritten digits. *http://yann. lecun. com/exdb/mnist/*.

LeCun, Yann, Yoshua Bengio, et al. 1995. Convolutional networks for images, speech, and time series. *The handbook of brain theory and neural networks* 3361(10): 1995.

LeCun, Yann, Bernhard E. Boser, John S. Denker, Donnie Henderson, Richard E. Howard, Wayne E. Hubbard, and Lawrence D. Jackel. 1989. Handwritten digit

recognition with a back-propagation network. In *Advances in Neural Information Processing Systems 2, NIPS 1989*, 396–404. Morgan Kaufmann.

Lécuyer, Mathias, Vaggelis Atlidakis, Roxana Geambasu, Daniel Hsu, and Suman Jana. 2019. Certified robustness to adversarial examples with differential privacy. In *IEEE Symposium on Security and Privacy, SP 2019*, 656–672. IEEE.

Lee, John M. 2003. *Introduction to smooth manifolds*, vol. 218 of *Graduate Texts in Mathematics*. New York, NY: Springer New York.

———. 2011. *Introduction to topological manifolds*. 2nd ed. Graduate texts in mathematics 202, New York, NY: Springer.

———. 2018. *Introduction to Riemannian manifolds*, vol. 176 of *Graduate Texts in Mathematics*. Cham: Springer International Publishing.

Lee, Joonho, Kumar Shridhar, Hideaki Hayashi, Brian Kenji Iwana, Seokjun Kang, and Seiichi Uchida. 2019. Probact: A probabilistic activation function for deep neural networks. *CoRR* abs/1905.10761. `1905.10761`.

Lee, Sang-gil, Sungwon Kim, and Sungroh Yoon. 2020. Nanoflow: Scalable normalizing flows with sublinear parameter complexity. In *Advances in Neural Information Processing Systems 33, NeurIPS 2020*.

Lenglet, Christophe, Mikaël Rousson, and Rachid Deriche. 2006. DTI segmentation by statistical surface evolution. *IEEE Trans. Medical Imaging* 25(6):685–700.

Lensink, Keegan, Eldad Haber, and Bas Peters. 2019. Fully hyperbolic convolutional neural networks. *CoRR* abs/1905.10484. `1905.10484`.

Leow, Alex D., Siwei Zhu, Katie McMahon, Greig I. de Zubicaray, Matthew Meredith, Margie Wright, and Paul M. Thompson. 2008. Probabilistic multi-tensor estimation using the tensor distribution function. In *Proceedings of Computer Vision and Pattern Recognition CVPR 2008*. IEEE.

Leporé, Natasha, Caroline C. Brun, Ming-Chang Chiang, Yi-Yu Chou, Rebecca A. Dutton, Kiralee M. Hayashi, Oscar L. Lopez, Howard Aizenstein, Arthur W. Toga, James T. Becker, and Paul M. Thompson. 2006. Multivariate statistics of the jacobian matrices in tensor based morphometry and their application to HIV/AIDS. In *Medical Image Computing and Computer-Assisted Intervention - MICCAI 2006*, vol. 4190 of *Lecture Notes in Computer Science*, 191–198. Springer.

Li, Bai, Changyou Chen, Wenlin Wang, and Lawrence Carin. 2019. Certified adversarial robustness with additive noise. In *Advances in Neural Information Processing Systems 32, NeurIPS 2019*, 9459–9469.

Li, Runze, Wei Zhong, and Liping Zhu. 2012. Feature screening via distance correlation learning. *Journal of the American Statistical Association* 107(499):1129–1139.

Lin, Hengxu, Dong Zhou, Weiqing Liu, and Jiang Bian. 2021. Deep risk model: a deep learning solution for mining latent risk factors to improve covariance matrix estimation. In *2nd ACM International Conference on AI in Finance, ICAIF 2021*, 12:1–12:8. ACM.

Liu, Jingen, Jiebo Luo, and Mubarak Shah. 2009. Recognizing realistic actions from videos. In *Proceedings of Computer Vision and Pattern Recognition CVPR 2009*, 1996–2003. IEEE.

Liu, Katherine, Kyel Ok, William Vega-Brown, and Nicholas Roy. 2018. Deep inference for covariance estimation: Learning gaussian noise models for state estimation. In *IEEE International Conference on Robotics and Automation, ICRA 2018*, 1436–1443. IEEE.

Liu, Shaoteng, Jingjing Chen, Liangming Pan, Chong-Wah Ngo, Tat-Seng Chua, and Yu-Gang Jiang. 2020. Hyperbolic visual embedding learning for zero-shot recognition. In *Proceedings of Computer Vision and Pattern Recognition, CVPR 2020*, 9270–9278. IEEE.

Liu, Xiao, Spyridon Thermos, Gabriele Valvano, Agisilaos Chartsias, Alison Q. O'Neil, and Sotirios A. Tsaftaris. 2021. Measuring the biases and effectiveness of content-style disentanglement. In *32nd British Machine Vision Conference, BMVC 2021*, 124. BMVA Press.

Locatello, Francesco, Michael Tschannen, Stefan Bauer, Gunnar Rätsch, Bernhard Schölkopf, and Olivier Bachem. 2020. Disentangling factors of variations using few labels. In *8th International Conference on Learning Representations, ICLR 2020*. OpenReview.net.

Lokhande, Vishnu Suresh, Aditya Kumar Akash, Sathya N. Ravi, and Vikas Singh. 2020a. Fairalm: Augmented lagrangian method for training fair models with little regret. In *Proceedings of the European Conference on Computer Vision, ECCV 2020*, vol. 12357, 365–381. Springer.

Lokhande, Vishnu Suresh, Rudrasis Chakraborty, Sathya N. Ravi, and Vikas Singh. 2022. Equivariance allows handling multiple nuisance variables when analyzing pooled neuroimaging datasets. In *Proceedings of Computer Vision and Pattern Recognition, CVPR 2022*, 10422–10431. IEEE.

Lokhande, Vishnu Suresh, Songwong Tasneeyapant, Abhay Venkatesh, Sathya N. Ravi, and Vikas Singh. 2020b. Generating accurate pseudo-labels in semi-supervised learning and avoiding overconfident predictions via hermite polynomial activations. In *Proceedings of Computer Vision and Pattern Recognition, CVPR 2020*, 11432–11440. IEEE.

Louizos, Christos, and Max Welling. 2016. Structured and efficient variational deep learning with matrix gaussian posteriors. In *Proceedings of the 33nd International Conference on Machine Learning, ICML 2016*, vol. 48 of *JMLR Workshop and Conference Proceedings*, 1708–1716. JMLR.org.

Ma, Justin, Alex Kulesza, Mark Dredze, Koby Crammer, Lawrence K. Saul, and Fernando Pereira. 2010. Exploiting feature covariance in high-dimensional online

learning. In *Proceedings of the 13th International Conference on Artificial Intelligence and Statistics, AISTATS 2010*, vol. 9 of *JMLR Proceedings*, 493–500. JMLR.org.

Madry, Aleksander, Aleksandar Makelov, Ludwig Schmidt, Dimitris Tsipras, and Adrian Vladu. 2018. Towards deep learning models resistant to adversarial attacks. In *6th International Conference on Learning Representations, ICLR 2018*. OpenReview.net.

Malach, Eran, and Shai Shalev-Shwartz. 2017. Decoupling "when to update" from "how to update". In *Advances in Neural Information Processing Systems 30, NeurIPS 2017, long beach, ca, USA*, 960–970.

Mansour, Romany F. 2018. Deep-learning-based automatic computer-aided diagnosis system for diabetic retinopathy. *Biomedical Engineering Letters* 8(1):41–57.

Manton, Jonathan H. 2004. A globally convergent numerical algorithm for computing the centre of mass on compact lie groups. In *8th International Conference on Control, Automation, Robotics and Vision, ICARCV 2004*, 2211–2216. IEEE.

Mao, Feng, Xiang Wu, Hui Xue, and Rong Zhang. 2018. Hierarchical video frame sequence representation with deep convolutional graph network. In *Proceedings of the European Conference on Computer Vision Workshops, ECCV Workshops 2018*, vol. 11132 of *Lecture Notes in Computer Science*, 262–270. Springer.

Maurice Fréchet. 1948. Les éléments aléatoires de nature quelconque dans un espace distancié. *Annales scientifiques de l'École normale supérieure* 65:211–237.

Meng, Zihang, Rudrasis Chakraborty, and Vikas Singh. 2021a. An online riemannian PCA for stochastic canonical correlation analysis. In *Advances in Neural Information Processing Systems 34, NeurIPS 2021*, 14056–14068.

Meng, Zihang, Vikas Singh, and Sathya N. Ravi. 2021b. Neural tmdlayer: Modeling instantaneous flow of features via SDE generators. In *Proceedings of International Conference on Computer Vision, ICCV 2021*, 11615–11624. IEEE.

Menzler, K, M Belke, E Wehrmann, K Krakow, U Lengler, A Jansen, HM Hamer, Wolfgang H Oertel, F Rosenow, and S Knake. 2011. Men and women are different: diffusion tensor imaging reveals sexual dimorphism in the microstructure of the thalamus, corpus callosum and cingulum. *Neuroimage* 54(4):2557–2562.

Miolane, Nina, and Susan P. Holmes. 2020. Learning weighted submanifolds with variational autoencoders and riemannian variational autoencoders. In *Proceedings of Computer Vision and Pattern Recognition, CVPR 2020*, 14491–14499. IEEE.

Mirman, Matthew, Timon Gehr, and Martin T. Vechev. 2018. Differentiable abstract interpretation for provably robust neural networks. In *Proceedings of the 35th International Conference on Machine Learning, ICML 2018*, vol. 80 of *Proceedings of Machine Learning Research*, 3575–3583. PMLR.

Mishne, Gal, Zhengchao Wan, Yusu Wang, and Sheng Yang. 2022. The numerical stability of hyperbolic representation learning. *CoRR* abs/2211.00181. `2211.00181`.

Morcos, Ari S., Maithra Raghu, and Samy Bengio. 2018. Insights on representational similarity in neural networks with canonical correlation. In *Advances in Neural Information Processing Systems 31, NeurIPS 2018*, 5732–5741.

Moyer, Daniel, Shuyang Gao, Rob Brekelmans, Aram Galstyan, and Greg Ver Steeg. 2018. Invariant representations without adversarial training. In *Advances in Neural Information Processing Systems 31, NeurIPS 2018*, 9102–9111.

Munkres, James R. 1984. *Elements of algebraic topology*. Addison-Wesley.

Netzer, Yuval, Tao Wang, Adam Coates, Alessandro Bissacco, Bo Wu, and Andrew Y. Ng. 2011. Reading digits in natural images with unsupervised feature learning. In *Advances in Neural Information Processing Systems Workshop on Deep Learning and Unsupervised Feature Learning, NIPS Workshop 2011*.

Neyshabur, Behnam, Hanie Sedghi, and Chiyuan Zhang. 2020. What is being transferred in transfer learning? In *Advances in Neural Information Processing Systems 33, NeurIPS 2020*.

Nguyen, Thao, Maithra Raghu, and Simon Kornblith. 2021. Do wide and deep networks learn the same things? uncovering how neural network representations vary with width and depth. In *9th International Conference on Learning Representations, ICLR 2021*. OpenReview.net.

Oliva, Junier B., Barnabás Póczos, and Jeff G. Schneider. 2017. The statistical recurrent unit. In *Proceedings of the 34th International Conference on Machine Learning, ICML 2017*, vol. 70 of *Proceedings of Machine Learning Research*, 2671–2680. PMLR.

Pan, Wenliang, Xueqin Wang, Canhong Wen, Martin Styner, and Hongtu Zhu. 2017. Conditional local distance correlation for manifold-valued data. In *Information Processing in Medical Imaging, IPMI 2017*, vol. 10265 of *Lecture Notes in Computer Science*, 41–52. Springer.

Papernot, Nicolas, Patrick D. McDaniel, Somesh Jha, Matt Fredrikson, Z. Berkay Celik, and Ananthram Swami. 2016. The limitations of deep learning in adversarial settings. In *IEEE European Symposium on Security and Privacy, euros&p 2016*, 372–387. IEEE.

Paszke, Adam, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Köpf, Edward Z. Yang, Zachary DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. 2019. Pytorch: An imperative style, high-performance deep learning library. In *Advances in Neural Information Processing Systems 32, NeurIPS 2019*, 8024–8035.

Patrini, Giorgio, Alessandro Rozza, Aditya Krishna Menon, Richard Nock, and Lizhen Qu. 2017. Making deep neural networks robust to label noise: A loss correction approach. In *Proceedings of Computer Vision and Pattern Recognition, CVPR 2017*, 2233–2241. IEEE.

Pearson, Karl. 1895. VII. Note on regression and inheritance in the case of two parents. *Proceedings of the Royal Society of London* 58(347-352):240–242.

Peng, Wei, Tuomas Varanka, Abdelrahman Mostafa, Henglin Shi, and Guoying Zhao. 2022. Hyperbolic deep neural networks: A survey. *IEEE Trans. Pattern Anal. Mach. Intell.* 44(12):10023–10044.

Perez-Nievas, Beatriz G., Thor D. Stein, Hwan-Ching Tai, Oriol Dols-Icardo, Thomas C. Scotton, Isabel Barroeta-Espar, Leticia Fernandez-Carballo, Estibaliz Lopez De Munain, Jesus Perez, Marta Marquie, Alberto Serrano-Pozo, Mathew P. Frosch, Val Lowe, Joseph E. Parisi, Ronald C. Petersen, Milos D. Ikonomovic, Oscar L. López, William Klunk, Bradley T. Hyman, and Teresa Gómez-Isla. 2013. Dissecting phenotypic traits linked to human resilience to Alzheimer's pathology. *Brain* 136(8):2510–2526.

Peterson, Leif E. 2009. K-nearest neighbor. *Scholarpedia* 4(2):1883.

Pugh, Justin K., Lisa B. Soros, and Kenneth O. Stanley. 2016. Quality diversity: A new frontier for evolutionary computation. *Frontiers Robotics AI* 3:40.

Raamana, Pradeep Reddy, and Stephen C. Strother. 2018. graynet: single-subject morphometric networks for neuroscience connectivity applications. *J. Open Source Softw.* 3(30):924.

Radford, Alec, Jong Wook Kim, Chris Hallacy, Aditya Ramesh, Gabriel Goh, Sandhini Agarwal, Girish Sastry, Amanda Askell, Pamela Mishkin, Jack Clark, Gretchen Krueger, and Ilya Sutskever. 2021. Learning transferable visual models from natural language supervision. In *Proceedings of the 38th International Conference on Machine Learning, ICML 2021*, vol. 139 of *Proceedings of Machine Learning Research*, 8748–8763. PMLR.

Radford, Alec, Luke Metz, and Soumith Chintala. 2016. Unsupervised representation learning with deep convolutional generative adversarial networks.

Raghu, Maithra, Justin Gilmer, Jason Yosinski, and Jascha Sohl-Dickstein. 2017. SVCCA: singular vector canonical correlation analysis for deep learning dynamics and interpretability. In *Advances in Neural Information Processing Systems 30, NeurIPS 2017*, 6076–6085.

Raghu, Maithra, Thomas Unterthiner, Simon Kornblith, Chiyuan Zhang, and Alexey Dosovitskiy. 2021. Do vision transformers see like convolutional neural networks? In *Advances in Neural Information Processing Systems 34, NeurIPS 2021*, 12116–12128.

Rakicevic, Nemanja, Antoine Cully, and Petar Kormushev. 2021. Policy manifold search: exploring the manifold hypothesis for diversity-based neuroevolution. In *GECCO '21: Genetic and evolutionary computation conference, lille, france, july 10-14, 2021*, 901–909. ACM.

Ramsay, J. O., Jos ten Berge, and G. P. H. Styan. 1984. Matrix correlation. *Psychometrika* 49(3):403–423.

Reed, Scott E., Honglak Lee, Dragomir Anguelov, Christian Szegedy, Dumitru Erhan, and Andrew Rabinovich. 2015. Training deep neural networks on noisy labels with bootstrapping. In *3rd International Conference on Learning Representations, ICLR 2015*.

Reiser, Patrick, Marlen Neubert, André Eberhard, Luca Torresi, Chen Zhou, Chen Shao, Houssam Metni, Clint van Hoesel, Henrik Schopmans, Timo Sommer, and Pascal Friederich. 2022. Graph neural networks for materials science and chemistry. *CoRR* abs/2208.09481. 2208.09481.

Ren, Mengye, Wenyuan Zeng, Bin Yang, and Raquel Urtasun. 2018. Learning to reweight examples for robust deep learning. In *Proceedings of the 35th International Conference on Machine Learning, ICML 2018*, vol. 80 of *Proceedings of Machine Learning Research*, 4331–4340. PMLR.

Rey, Luis A. Pérez, Vlado Menkovski, and Jim Portegies. 2020. Diffusion variational autoencoders. In *Proceedings of the 29th International Joint Conference on Artificial Intelligence, IJCAI 2020*, 2704–2710. ijcai.org.

Rezende, Danilo Jimenez, and Shakir Mohamed. 2015. Variational inference with normalizing flows. In *Proceedings of the 32nd International Conference on Machine*

Learning, ICML 2015, vol. 37 of *JMLR Workshop and Conference Proceedings*, 1530–1538. JMLR.org.

Ridgway, Gerard Robert. 2009. Statistical analysis for longitudinal mr imaging of dementia. Ph.D. thesis, UCL (University College London).

Salimans, Tim, and Diederik P. Kingma. 2016. Weight normalization: A simple reparameterization to accelerate training of deep neural networks. In *Advances in Neural Information Processing Systems 29, NIPS 2016*, 901.

Salman, Hadi, Jerry Li, Ilya P. Razenshteyn, Pengchuan Zhang, Huan Zhang, Sébastien Bubeck, and Greg Yang. 2019. Provably robust deep learning via adversarially trained smoothed classifiers. In *Advances in Neural Information Processing Systems 32, NeurIPS 2019*, 11289–11300.

Sarkar, Rik. 2011. Low distortion delaunay embedding of trees in hyperbolic plane. In *Graph Drawing, GD 2011*, vol. 7034 of *Lecture Notes in Computer Science*, 355–366. Springer.

Scaglione, Anna, Roberto Pagliari, and Hamid Krim. 2008. The decentralized estimation of the sample covariance. In *42nd Asilomar Conference on Signals, Systems and Computers, ACSSC 2008*, 1722–1726. IEEE.

Schiegg, Martin, Ferran Diego, and Fred A. Hamprecht. 2016. Learning diverse models: The coulomb structured support vector machine. In *Proceedings of the European Conference on Computer Vision, ECCV 2016*, vol. 9907, 585–599. Springer.

Selvaraju, Ramprasaath R., Michael Cogswell, Abhishek Das, Ramakrishna Vedantam, Devi Parikh, and Dhruv Batra. 2017. Grad-cam: Visual explanations from deep networks via gradient-based localization. In *Proceedings of International Conference on Computer Vision, ICCV 2017*, 618–626. IEEE.

Shafahi, Ali, Mahyar Najibi, Amin Ghiasi, Zheng Xu, John P. Dickerson, Christoph Studer, Larry S. Davis, Gavin Taylor, and Tom Goldstein. 2019. Adversarial training

for free! In *Advances in Neural Information Processing Systems 32, NeurIPS 2019*, 3353–3364.

Shlomi, Jonathan, Peter W. Battaglia, and Jean-Roch Vlimant. 2021. Graph neural networks in particle physics. *Mach. Learn. Sci. Technol.* 2(2):21001.

Shu, Rui, Yining Chen, Abhishek Kumar, Stefano Ermon, and Ben Poole. 2020. Weakly supervised disentanglement with guarantees. In *8th International Conference on Learning Representations, ICLR 2020*. OpenReview.net.

Shumailov, Ilia, Xitong Gao, Yiren Zhao, Robert D. Mullins, Ross Anderson, and Cheng-Zhong Xu. 2019. Sitatapatra: Blocking the transfer of adversarial samples. *CoRR* abs/1901.08121. `1901.08121`.

Simonyan, Karen, and Andrew Zisserman. 2015. Very deep convolutional networks for large-scale image recognition. In *3rd International Conference on Learning Representations, ICLR 2015*.

Singh, Gagandeep, Timon Gehr, Matthew Mirman, Markus Püschel, and Martin T. Vechev. 2018. Fast and effective robustness certification. In *Advances in Neural Information Processing Systems 31, NeurIPS 2018*, 10825–10836.

Son, Seong-jin, Bo-yong Park, Kyoungseob Byeon, and Hyunjin Park. 2019. Synthesizing diffusion tensor imaging from functional MRI using fully convolutional networks. *Comput. Biol. Medicine* 115.

Song, Jiaming, Pratyusha Kalluri, Aditya Grover, Shengjia Zhao, and Stefano Ermon. 2019. Learning controllable fair representations. In *Proceedings of the 22nd International Conference on Artificial Intelligence and Statistics, AISTATS 2019*, vol. 89 of *Proceedings of Machine Learning Research*, 2164–2173. PMLR.

Sonthalia, Rishi, and Anna C. Gilbert. 2020. Tree! I am no tree! I am a low dimensional hyperbolic embedding. In *Advances in Neural Information Processing Systems 33, NeurIPS 2020*.

Srivastava, Anuj, Ian Jermyn, and Shantanu H. Joshi. 2007. Riemannian analysis of probability density functions with applications in vision. In *Proceedings of Computer Vision and Pattern Recognition CVPR 2007*. IEEE.

Srivastava, Nitish, Geoffrey E. Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. 2014. Dropout: a simple way to prevent neural networks from overfitting. *J. Mach. Learn. Res.* 15(1):1929–1958.

Srivastava, Nitish, Elman Mansimov, and Ruslan Salakhutdinov. 2015. Unsupervised learning of video representations using lstms. In *Proceedings of the 32nd International Conference on Machine Learning, ICML 2015*, vol. 37 of *JMLR Workshop and Conference Proceedings*, 843–852. JMLR.org.

Straub, Julian, Jason Chang, Oren Freifeld, and John W. Fisher III. 2015. A dirichlet process mixture model for spherical data. In *Proceedings of the 18th International Conference on Artificial Intelligence and Statistics, AISTATS 2015*, vol. 38 of *JMLR Workshop and Conference Proceedings*. JMLR.org.

Sun, Haoliang, Ronak Mehta, Hao Henry Zhou, Zhichun Huang, Sterling C. Johnson, Vivek Prabhakaran, and Vikas Singh. 2019. DUAL-GLOW: conditional flow-based generative model for modality transfer. In *Proceedings of International Conference on Computer Vision, ICCV 2019*, 10610–10619. IEEE.

Szegedy, Christian, Wojciech Zaremba, Ilya Sutskever, Joan Bruna, Dumitru Erhan, Ian J. Goodfellow, and Rob Fergus. 2014. Intriguing properties of neural networks. In *2nd International Conference on Learning Representations, ICLR 2014*.

Székely, Gábor J., and Maria L. Rizzo. 2014. Partial distance correlation with methods for dissimilarities. *The Annals of Statistics* 42(6).

Székely, Gábor J., Maria L. Rizzo, and Nail K. Bakirov. 2007. Measuring and testing dependence by correlation of distances. *The Annals of Statistics* 35(6).

Tan, Mingxing, and Quoc V. Le. 2019. Efficientnet: Rethinking model scaling for convolutional neural networks. In *Proceedings of the 36th International Conference on*

Machine Learning, ICML 2019, vol. 97 of *Proceedings of Machine Learning Research*, 6105–6114. PMLR.

Terranova, N., O. Serot, P. Archier, C. De Saint Jean, and M. Sumini. 2015. Covariance Matrix Evaluations for Independent Mass Fission Yields. *Nuclear Data Sheets* 123:225–230.

Terrell, George R., and David W. Scott. 1992. Variable kernel density estimation. *The Annals of Statistics* 20(3).

Toth, Peter, Danilo J. Rezende, Andrew Jaegle, Sébastien Racanière, Aleksandar Botev, and Irina Higgins. 2020. Hamiltonian generative networks. In *8th International Conference on Learning Representations, ICLR 2020*. OpenReview.net.

Tramèr, Florian, Alexey Kurakin, Nicolas Papernot, Ian J. Goodfellow, Dan Boneh, and Patrick D. McDaniel. 2018. Ensemble adversarial training: Attacks and defenses. In *6th International Conference on Learning Representations, ICLR 2018*. OpenReview.net.

Tsay, Ruey S. 2005. *Analysis of financial time series: Tsay/Analysis of financial time series*. Wiley Series in Probability and Statistics, Hoboken, NJ, USA: John Wiley & Sons, Inc.

Tung, Michael M. 2018. Modelling acoustics on the poincaré half-plane. *Journal of Computational and Applied Mathematics* 337:366–372.

Turaga, Pavan K., Ashok Veeraraghavan, and Rama Chellappa. 2008. Statistical analysis on stiefel and grassmann manifolds with applications in computer vision. In *Proceedings of Computer Vision and Pattern Recognition CVPR 2008*. IEEE.

Uesato, Jonathan, Brendan O'Donoghue, Pushmeet Kohli, and Aäron van den Oord. 2018. Adversarial risk and the dangers of evaluating against weak attacks. In *Proceedings of the 35th International Conference on Machine Learning, ICML 2018*, vol. 80 of *Proceedings of Machine Learning Research*, 5032–5041. PMLR.

Ulyanov, Dmitry, Andrea Vedaldi, and Victor S. Lempitsky. 2016. Instance normalization: The missing ingredient for fast stylization. *CoRR* abs/1607.08022. 1607.08022.

Veraart, Jelle, Dmitry S. Novikov, Daan Christiaens, Benjamin Ades-aron, Jan Sijbers, and Els Fieremans. 2016. Denoising of diffusion MRI using random matrix theory. *NeuroImage* 142:394–406.

Wakana, Setsu, Arvind Caprihan, Martina M. Panzenboeck, James H. Fallon, Michele Perry, Randy L. Gollub, Kegang Hua, Jiangyang Zhang, Hangyi Jiang, Prachi Dubey, Ari Blitz, Peter C. M. van Zijl, and Susumu Mori. 2007. Reproducibility of quantitative tractography methods applied to cerebral white matter. *NeuroImage* 36(3):630–644.

Wang, Zhizhou, and Baba C. Vemuri. 2005. DTI segmentation using an information theoretic tensor dissimilarity measure. *IEEE Trans. Medical Imaging* 24(10):1267–1277.

Wasserthal, Jakob, Peter F. Neher, and Klaus H. Maier-Hein. 2018. Tractseg - fast and accurate white matter tract segmentation. *NeuroImage* 183:239–253.

Weng, Tsui-Wei, Huan Zhang, Hongge Chen, Zhao Song, Cho-Jui Hsieh, Luca Daniel, Duane S. Boning, and Inderjit S. Dhillon. 2018. Towards fast computation of certified robustness for relu networks. In *Proceedings of the 35th International Conference on Machine Learning, ICML 2018*, vol. 80 of *Proceedings of Machine Learning Research*, 5273–5282. PMLR.

Wishart, John. 1928. The generalised product moment distribution in samples from a normal multivariate population. *Biometrika* 20A(1-2):32–52.

Wong, Eric, and J. Zico Kolter. 2018. Provable defenses against adversarial examples via the convex outer adversarial polytope. In *Proceedings of the 35th International Conference on Machine Learning, ICML 2018*, vol. 80 of *Proceedings of Machine Learning Research*, 5283–5292. PMLR.

Wozniak, Stanislaw, Angeliki Pantazi, Thomas Bohnstingl, and Evangelos Eleftheriou. 2020. Deep learning incorporating biologically inspired neural dynamics and in-memory computing. *Nat. Mach. Intell.* 2(6):325–336.

Wu, Dan, Xin Wang, Junjie Bai, Xiaoyang Xu, Bin Ouyang, Yuwei Li, Heye Zhang, Qi Song, Kunlin Cao, and Youbing Yin. 2019. Automated anatomical labeling of coronary arteries via bidirectional tree lstms. *Int. J. Comput. Assist. Radiol. Surg.* 14(2):271–280.

Wu, Zhirong, Yuanjun Xiong, Stella X. Yu, and Dahua Lin. 2018. Unsupervised feature learning via non-parametric instance discrimination. In *Proceedings of Computer Vision and Pattern Recognition, CVPR 2018*, 3733–3742. IEEE.

Xiao, Lechao, Yasaman Bahri, Jascha Sohl-Dickstein, Samuel S. Schoenholz, and Jeffrey Pennington. 2018. Dynamical isometry and a mean field theory of cnns: How to train 10, 000-layer vanilla convolutional neural networks. In *Proceedings of the 35th International Conference on Machine Learning, ICML 2018*, vol. 80 of *Proceedings of Machine Learning Research*, 5389–5398. PMLR.

Xu, Weihua, Feifei Gao, Jianhua Zhang, Xiaoming Tao, and Ahmed Alkhateeb. 2021. Deep learning based channel covariance matrix estimation with user location and scene images. *IEEE Trans. Commun.* 69(12):8145–8158.

Xu, Yangyang, and Wotao Yin. 2015. Block stochastic gradient iteration for convex and nonconvex optimization. *SIAM J. Optim.* 25(3):1686–1716.

Xu, Yilun, Hao He, Tianxiao Shen, and Tommi S. Jaakkola. 2022. Controlling directions orthogonal to a classifier. In *10th International Conference on Learning Representations, ICLR 2022*. OpenReview.net.

Yang, Guandao, Xun Huang, Zekun Hao, Ming-Yu Liu, Serge J. Belongie, and Bharath Hariharan. 2019. Pointflow: 3d point cloud generation with continuous normalizing flows. In *Proceedings of International Conference on Computer Vision, ICCV 2019*, 4540–4549. IEEE.

Yang, Han, Xingjian Zhen, Ying Chi, Lei Zhang, and Xian-Sheng Hua. 2020. CPR-GCN: conditional partial-residual graph convolutional network in automated anatomical labeling of coronary arteries. In *Proceedings of Computer Vision and Pattern Recognition, CVPR 2020*, 3802–3810. IEEE.

Yang, Yinchong, Denis Krompass, and Volker Tresp. 2017. Tensor-train recurrent neural networks for video classification. In *Proceedings of the 34th International Conference on Machine Learning, ICML 2017*, vol. 70 of *Proceedings of Machine Learning Research*, 3891–3900. PMLR.

Yendiki, Anastasia, Kami Koldewyn, Sita Kakunoori, Nancy Kanwisher, and Bruce Fischl. 2014. Spurious group differences due to head motion in a diffusion MRI study. *NeuroImage* 88:79–90.

Yendiki, Anastasia, Patricia Panneck, Priti Srinivasan, Allison Stevens, Lilla Zöllei, Jean Augustinack, Ruopeng Wang, David H. Salat, Stefan Ehrlich, Tim E. J. Behrens, Saâd Jbabdi, Randy L. Gollub, and Bruce Fischl. 2011. Automated probabilistic reconstruction of white-matter pathways in health and disease using an atlas of the underlying anatomy. *Frontiers Neuroinformatics* 5:23.

Yendiki, Anastasia, Martin Reuter, Paul Wilkens, H. Diana Rosas, and Bruce Fischl. 2016. Joint reconstruction of white-matter pathways from longitudinal diffusion MRI data with anatomical priors. *NeuroImage* 127:277–286.

Yu, Kaicheng, and Mathieu Salzmann. 2017. Second-order convolutional neural networks. *CoRR* abs/1703.06817. `1703.06817`.

Yu, Ning, Connelly Barnes, Eli Shechtman, Sohrab Amirghodsi, and Michal Lukác. 2019. Texture mixer: A network for controllable synthesis and interpolation of texture. In *Proceedings of Computer Vision and Pattern Recognition, CVPR 2019*, 12164–12173. IEEE.

Yu, Tao, and Christopher De Sa. 2019. Numerically accurate hyperbolic embeddings using tiling-based models. In *Advances in Neural Information Processing Systems 32, NeurIPS 2019*, 2021–2031.

Zacur, Ernesto, Matías N. Bossa, and Salvador Olmos. 2014. Multivariate tensor-based morphometry with a right-invariant riemannian distance on gl+(n). *J. Math. Imaging Vis.* 50(1-2):18–31.

Zafar, Muhammad Bilal, Isabel Valera, Manuel Gomez-Rodriguez, and Krishna P. Gummadi. 2017. Fairness constraints: Mechanisms for fair classification. In *Proceedings of the 20th International Conference on Artificial Intelligence and Statistics, AISTATS 2017*, vol. 54 of *Proceedings of Machine Learning Research*, 962–970. PMLR.

Zemel, Richard S., Yu Wu, Kevin Swersky, Toniann Pitassi, and Cynthia Dwork. 2013. Learning fair representations. In *Proceedings of the 30th International Conference on Machine Learning, ICML 2013*, vol. 28 of *JMLR Workshop and Conference Proceedings*, 325–333. JMLR.org.

Zeng, Zhanpeng, Yunyang Xiong, Sathya N. Ravi, Shailesh Acharya, Glenn Moo Fung, and Vikas Singh. 2021. You only sample (almost) once: Linear cost self-attention via bernoulli sampling. In *Proceedings of the 38th International Conference on Machine Learning, ICML 2021*, vol. 139 of *Proceedings of Machine Learning Research*, 12321–12332. PMLR.

Zhai, Runtian, Chen Dan, Di He, Huan Zhang, Boqing Gong, Pradeep Ravikumar, Cho-Jui Hsieh, and Liwei Wang. 2020. MACER: attack-free and scalable robust training via maximizing certified radius. In *8th International Conference on Learning Representations, ICLR 2020*. OpenReview.net.

Zhang, Chiyuan, Samy Bengio, Moritz Hardt, Benjamin Recht, and Oriol Vinyals. 2017. Understanding deep learning requires rethinking generalization. In *5th International Conference on Learning Representations, ICLR 2017*. OpenReview.net.

Zhang, Dinghuai, Tianyuan Zhang, Yiping Lu, Zhanxing Zhu, and Bin Dong. 2019. You only propagate once: Accelerating adversarial training via maximal principle. In *Advances in Neural Information Processing Systems 32, NeurIPS 2019*, 227–238.

Zhang, Huan, Tsui-Wei Weng, Pin-Yu Chen, Cho-Jui Hsieh, and Luca Daniel. 2018. Efficient neural network robustness certification with general activation functions. In *Advances in Neural Information Processing Systems 31, NeurIPS 2018*, 4944–4953.

Zhang, Ruijie, Fushan Wei, and Bicheng Li. 2014. E2LSH based multiple kernel approach for object detection. *Neurocomputing* 124:105–110.

Zhang, Zhilu, and Mert R. Sabuncu. 2018. Generalized cross entropy loss for training deep neural networks with noisy labels. In *Advances in Neural Information Processing Systems 31, NeurIPS 2018*, 8792–8802.

Zhen, Xingjian, Rudrasis Chakraborty, and Vikas Singh. 2021a. Simpler certified radius maximization by propagating covariances. In *Proceedings of Computer Vision and Pattern Recognition, CVPR 2021*, 7292–7301. IEEE.

Zhen, Xingjian, Rudrasis Chakraborty, Nicholas Vogt, Barbara B. Bendlin, and Vikas Singh. 2019. Dilated convolutional neural networks for sequential manifold-valued data. In *Proceedings of International Conference on Computer Vision, ICCV 2019*, 10620–10630. IEEE.

Zhen, Xingjian, Rudrasis Chakraborty, Liu Yang, and Vikas Singh. 2021b. Flow-based generative models for learning manifold to manifold mappings. In *Proceedings of the 35th AAAI Conference on Artificial Intelligence, AAAI 2021*, 11042–11052. AAAI Press.

Zhen, Xingjian, Zihang Meng, Rudrasis Chakraborty, and Vikas Singh. 2022. On the versatile uses of partial distance correlation in deep learning. In *Proceedings of the European Conference on Computer Vision, ECCV 2022*, vol. 13686, 327–346. Springer.

Zhou, Bolei, Àgata Lapedriza, Aditya Khosla, Aude Oliva, and Antonio Torralba. 2018. Places: A 10 million image database for scene recognition. *IEEE Trans. Pattern Anal. Mach. Intell.* 40(6):1452–1464.

Zhou, Zhou. 2012. Measuring nonlinear dependence in time-series, a distance correlation approach. *Journal of Time Series Analysis* 33(3):438–457.