DISSERTATION

# USING MATHEMATICAL TECHNIQUES TO LEVERAGE DOMAIN KNOWLEDGE IN IMAGE ANALYSIS FOR EARTH SCIENCE

Submitted by

Lander Ver Hoef

Department of Mathematics

In partial fulfillment of the requirements

For the Degree of Doctor of Philosophy

Colorado State University

Fort Collins, Colorado

Spring 2023

ABSTRACT

USING MATHEMATICAL TECHNIQUES TO LEVERAGE DOMAIN KNOWLEDGE IN IMAGE ANALYSIS FOR EARTH SCIENCE

When presented with the power of modern machine learning techniques, there is a belief that we can simply let these algorithms loose on the data and see what they can find, unconstrained by human choice or bias. While such approaches can be useful, they are (of course) not fully free of bias or choice. Moreover, by utilizing the deep store of knowledge built up by scientific domains over decades or centuries, we can make architectural choices in our machine learning algorithms that focus the learning on features that we already know are important and informative, leading to more efficient, explainable, and interpretable methods. In this work, we present three examples of this approach. In the first project, to make use of the knowledge that texture is an important attribute of clouds, we use tools from topological data analysis focusing on the texture of satellite imagery, which leads to an effective and highly interpretable classifier of mesoscale cloud organization. This project resulted in a paper that has been published as a journal article. In the second project, we compare a rotationally invariant convolutional neural network against a conventional CNN both with and without data augmentation in their performance and behaviors on the task of predicting the major and minor axes lengths of storms in forecast data. Finally, in the third project, we explore three different techniques from harmonic analysis to enhance the signature of gravity waves in satellite imagery.

ACKNOWLEDGEMENTS

The work in Chapter 2 was supported in part by the National Science Foundation under Grant No. OAC-1934668, and under Grant No. ICER-2019758 which funds the NSF AI Institute for Research on Trustworthy AI in Weather, Climate, and Coastal Oceanography. I'd like to thank Charles H. White at CIRA (CSU) for constructive feedback and excellent suggestions for Chapter 2. I would also like to thank [62] for making their dataset publicly available.

The work in Chapter 3 was supported by the National Science Foundation under Grant No. ICER-2019758. I would like to thank the National Center for Atmospheric Research Computational and Information Systems Lab for hosting me via their visitor program for the summer of 2022, as well as the entire Machine Integration and Learning for Earth Systems group, including D.J. Gagne, John Schreck, Charlie Becker, Gabrielle Gantos, Da Fan, and Taysia Peterson for their support, advice, and insight.

The work in Chapter 4 was supported by the National Science Foundation under Grant No. OAC-1934668.

More generally, I would like to thank my co-advisors Dr. Henry Adams and Dr. Emily King and my committee member Dr. Imme Ebert-Uphoff for their encouragement, support, and help at every step of this process. I am incredibly lucky to have so many people be continuously invested in my success, so thank you.

DEDICATION

*For Shannon, with all my love.*

TABLE OF CONTENTS

LIST OF FIGURES

# Chapter 1

# Introduction

The mathematical study of images has existed for a very long time and remains a hot topic of research today. As visual creatures, humans are excellent at identifying trends, patterns, and similarities and dissimilarities in images and video, but finding algorithms to extract those same patterns automatically can be surprisingly challenging. Over the last decade, convolutional neural networks (CNNs) and their derivatives have grown to dominate the field of image analysis due to their flexibility, relative simplicity, and high performance.

However, CNNs and other deep neural networks are far from perfect. There is a tendency to tackle any problem by deploying neural networks and to attempt to improve performance by simply increasing the size of the network and feeding it more data. This approach feels "objective," as the researcher is not explicitly incorporating prior knowledge into the algorithm. However, biases will always exist within and around such algorithms, in the data itself and questions that are asked of the data if nowhere else. Moreover, domain knowledge can be incredibly useful in informing the types of algorithms used, the aspects of data those algorithms focus on, and how they are implemented. Deep neural networks are still a valid and important technique and can be modified to explicitly incorporate even more domain knowledge, as we will explore in Chapter 3. In addition to simply improving neural networks, however, there is a growing resurgence in classical and deterministic mathematical techniques in image analysis.

One field that has seen rapid growth in recent years is Topological Data Analysis (TDA). Sublevelset or superlevelset persistent homology (which we will refer to simply as *persistent homology*) is a technique that can summarize the shape and connectivity of a space. When applied to image data, this gives a descriptor of the texture and overall structure of an image. Persistent homology can then be used to describe or even classify sets of images, and it has been widely used, including emerging applications in in environmental science [28, 48, 73, 83]. Because this is a deterministic technique from a well-studied area of math, we can trace these descriptors and classifiers back through the algorithm, allowing us to gain a much deeper understanding of what attributes of the images are being used to make decisions.

An even more recent field, geometric deep learning, seeks to explicitly incorporate deeply theoretical concepts directly into sophisticated neural networks [6]. This approach uses techniques from group theory, representation theory, and geometry to build more powerful and purposeful networks by explicitly identifying symmetries of the data that are known to be important based on domain knowledge. By building a network that a priori respects these symmetries, we remove the burden on the network to learn these patterns from the data, which in theory should free it to learn other, more sophisticated patterns.

Applied harmonic analysis has continued to develop both new techniques and refinements of older methods that can give efficient, analytical descriptions of images. Wavelets, shearlets, and similar methods have proven to be extremely effective at detecting edges and periodic patterns in images without needing huge training datasets or supercomputers, and can be paired with simpler machine learning algorithms to increase their flexibility to more general problems [20, 41, 57].

The three projects described in this document cover these varying approaches to image analysis, with the underlying theme of using mathematical insights rather than automated algorithms. In our uses of TDA and applied harmonic analysis (Chapters 2 and 4 respectively), this comes in the form of using mathematical techniques to transform the data into some new form in which the features that we're interested in are more obvious and easily detectable even by simple machine learning algorithms. In Chapter 3, on the other hand, we use math to inform the machine learning algorithm itself, enforcing symmetries that we know are important in our input space. In each case, however, we are informed by domain knowledge about what aspects of the data are relevant and important, and use mathematics to tailor our approaches to emphasize those aspects.

## 1.1 Document Structure

In this document, we will discuss three projects tied together by the theme of using mathematics to inform or enhance image analysis techniques and applications.

In Chapter 2, we will present a completed project using TDA to classify types of clouds present in satellite imagery. This work was motivated by questions brought to us by researchers at the Cooperative Institute for Research in the Atmosphere (CIRA) and resulted in a paper that has been published in Artificial Intelligence for the Earth Sciences (AIES), a journal of the American Meteorological Society. The paper is available in early online release [85]. In this chapter, we'll give an introduction to persistent homology, how it can be used for image analysis, and give an example workflow in which persistent homology is paired with a simple machine learning algorithm to effectively classify satellite images of cloud organizations.

In Chapter 3, we discuss a project conducted as part of the Computational and Information Systems Lab (CISL) Visitor Program at the National Center for Atmospheric Research (NCAR). We explored an emerging topic in machine learning, *geometric deep learning*, a new framework that seeks to both provide a unifying structure for existing approaches and to give a principled method for developing new techniques. This framework is deeply mathematical in nature and fundamentally relies on topics from abstract algebra such as symmetry groups and group actions and seeks to utilize domain knowledge about what sorts of symmetries are important to enhance the power and efficiency of neural network approaches. In our work, we compare and contrast the behaviors and performance of a rotationally invariant network based on geometric deep learning against conventional neural networks on an example task on storm forecast data.

Finally, in Chapter 4, we present recent work done with CIRA focusing on the challenging task of enhancing the signature of atmospheric gravity waves in satellite imagery. These gravity waves are only faintly visible on particular moonless nights and are visible only very rarely in the complete dataset of satellite observations, even when restricted to the sensor that can detect them. This scarcity means that there are very, very few labeled samples with which to develop a detection algorithm, which makes training a neural network impracticable without some form of severe data augmentation or synthetic data generation. Instead of taking those routes, we explored a number of deterministic transformations from harmonic analysis with the aim of distinguishing gravity waves from other features. The eventual goal is to build this into an automated gravity wave detector that can start building a larger database of occurrences, but the transformations themselves proved interesting, and we present our work on them here as an example of how building a qualitative understanding of transformation properties can be key to an exploratory project.

In each of these chapters, we have included a “Narrative and Contributions” section in the introduction that gives an overview of the history of the project, who was involved, and how each member of the team contributed to the work. We hope that these are also illustrative of the process by which convergent research is conducted.

# Chapter 2

# A Primer on Topological Data Analysis to Support Image Analysis Tasks in Environmental Science

In this chapter, we will present a work that has been published as a journal paper. An early online release version can be found as [85]. The majority of the content in this chapter is identical to that of the published version, with the exception of the "Narrative and Contributions" section of the introduction (Section 2.1.6), which is original for this dissertation.

## 2.1 Introduction

Methods for image analysis have become an essential tool for many environmental science (ES) applications to automatically extract key information from satellite imagery or from gridded model output [23, 26, 70, 89]. Machine learning (ML) methods such as convolutional neural networks (CNNs) are now the dominant technique for many such tasks, where they operate as black boxes [52]. This is undesirable for high stakes applications [51, 68]. In this paper, we show how a tool that is beginning to be used in the community, namely *Topological Data Analysis (TDA)*, can be combined with ML methods for interpretable image analysis. TDA is a mathematical discipline that can quantify geometric information from an image in a predictable and well-understood way. In Section 2.4.4, we give a novel example of how we can leverage this understanding to give a strong interpretation of ML results in terms of image features.

TDA has proven highly successful to aid in the analysis of data in a variety of applications, including neuroscience [14, 28], fluid dynamics [46], and cancer histology [48]. In environmental science, TDA has recently shown potential to help identify atmospheric rivers [59], detect solar flares [19, 80], identify which wildfires are active [40], quantify the diurnal cycle in hurricanes [83], identify local climate zones [74], detect and visualize Rossby waves [53], and forecast COVID-19 spread using atmospheric data [73]. The purpose of this article is to provide an intuitive introduction to TDA for the environmental science community—using a meteorological application as guiding example—and an understanding of where TDA might be applied. This article is accompanied by easy-to-follow sample code provided as a GitHub repository (https://github.com/zyjux/sffg_tda) that we hope will be used by the community in new applications.

### 2.1.1 Guiding application - analysing the mesoscale organization of clouds

In order to provide a gentle introduction to TDA for the ES community, we illustrate its use for a practical example. We chose the application of classifying the mesoscale organization of clouds, specifically distinguishing four types of organization—*sugar*, *gravel*, *fish*, and *flowers*—identified by [78]. This task provides an ideal case study for our exploration of topological data analysis for several reasons: (1) These four organization patterns are well known from the seminal paper, [78], and meteorological experts were able to reliably identify these patterns from satellite visible imagery. (2) The task can be formulated as classification of patches of single-image monochromatic imagery, which a common TDA algorithm (*persistent homology*) is well-suited for. (3) TDA has never been applied for this application, so it is novel. (4) A well developed benchmark data set with reliable crowd sourced labels is publicly available for this task [62].

Several ML approaches have already been developed with good success for this benchmark data set to classify the four different types [62]. We emphasize that we are *not* seeking to match or exceed the performance of those ML approaches. Rather, we use this application to demonstrate TDA as an approach that can help increase transparency, decrease computational effort, and be feasible even if few labeled data samples are available; see Section 2.1.3.

### 2.1.2 Key TDA concepts discussed here

In this paper, we focus on the TDA concept that is most appropriate for image analysis, *persistent homology*. We will provide a detailed introduction in Section 2.3, but in this subsection give a short preview of key concepts to be discussed.

*Homology* is the classical study of connectivity and the presence of holes of various dimensions, giving large-scale geometric information. *Persistent homology* provides a descriptor with information on the texture of an image (how rough or smooth it is) and which can be vectorized into a format useful for machine learning. It does this by scaling through all the intensity values in an image and recording at what intensities connected components and holes appear and disappear. Particularly on images, persistent homology and its vectorizations can be efficiently computed, so for image analysis (from models or satellites), the computational effort of implementing persistent homology is small.

The results of persistent homology computations can be displayed as either *persistence diagrams* or *persistence barcodes*. We focus here on barcodes, in which each feature (connected component or hole) appears as a bar which starts at the intensity value at which the feature appears, and ends at the intensity at which it disappears. The lengths of these bars indicate the *persistence* of each feature. The raw output of persistent homology is not suitable for most machine learning tasks, as the output vector varies in length from sample to sample. While there are many proposed solutions to this, in this paper we use *persistence landscapes*, which translate a barcode into a mountain range, with the height of each mountain representing the persistence of the corresponding feature. The landscape is obtained from this mountain range by taking the $n$ highest profiles as piecewise-linear functions, where $n$ is a hyperparameter.

### 2.1.3 Advantages of TDA for image analysis tasks in environmental science

Persistent homology is a deterministic mathematical transformation (just as, say, the well known Fourier transform). We first explore the advantages that persistent homology inherits from being a deterministic algorithm.

1. **Transparency:** All the internal steps of the algorithm are known and well-understood, and the method has a high degree of theoretical interpretation, giving it far more transparency than most ML methods. In Section 2.4.4, we use this theoretical background to understand what image features are driving differences in the output of persistent homology.

2. **Known failure modes:** No technique is perfect, and there will always be situations that cause errors and incorrect results. To use a method in practice, it is important to understand in what situations it struggles and what sorts of errors can result. Because persistent homology is a deterministic method, we can both theoretically predict these failure modes and interpret experimental results in terms of the original feature space.

3. **No need for large labeled datasets:** As a deterministic algorithm, persistent homology does not require large, reliably labeled datasets. Instead, a small set of representative examples can be used to explore the different patterns that emerge in the transformed data. TDA is often used in combination with a simple ML model, and the number of labeled samples to obtain good performance is smaller than would be required to train a CNN or similar tool without TDA. This is a huge advantage for environmental science datasets, which are frequently large and detailed but almost entirely unlabeled.

4. **Environmentally friendly:** Many CNNs for image analysis tasks are known to have a surprisingly high carbon footprint due to the extensive computational resources required for model training [72, 87]. TDA is more in line with the Green AI movement [72, 87], as it enables context-driven numerical results without the environmental impact inherent in training a deep neural network.

Next we discuss the key abilities that persistent homology brings to image analysis tasks. These fall into three general categories: the incorporation of spatial context into a deterministic algorithm, the detection of texture and contrast, and invariance under certain transformations.

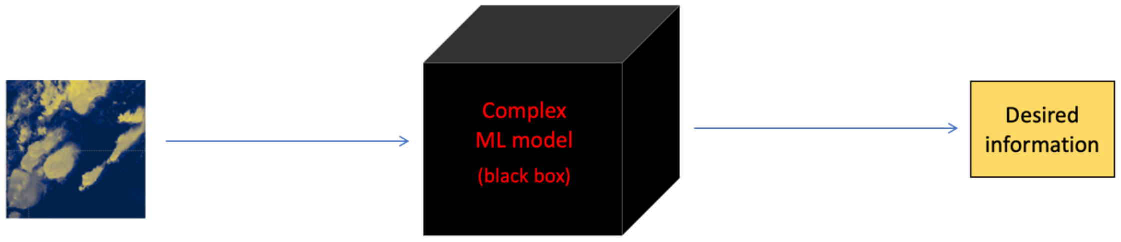
1. **Incorporating spatial context:** Many deterministic algorithms, as well as fully-connected neural networks, struggle to incorporate the spatial context inherent in satellite data. Integrating this spatial context is precisely what motivated the development of CNNs, but CNNs are costly to train and challenging to make explainable. Persistent homology, naturally incorporates spatial context, so patterns that are evident in this spatial context can be incorporated without resorting to CNNs or other spatially-informed neural network architectures.

2. **Detection of texture and contrast:** Persistent homology excels at detecting contrast differences in regions (small or large) that differ from the surrounding average, which gives a representation of the texture present in an image. This focus on texture is useful in analyzing satellite weather imagery, as texture is frequently a key distinguishing factor, even more than a cloud being a particular shape or size.

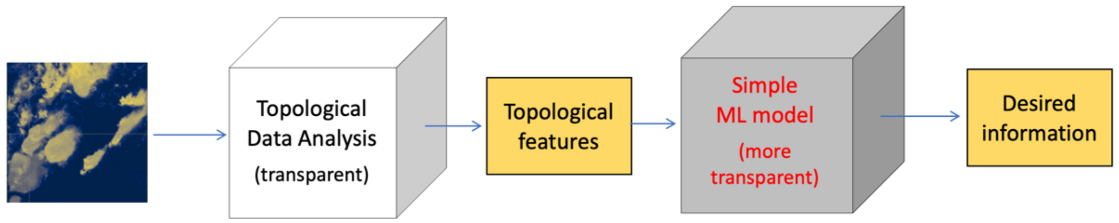
3. **Invariance to homeomorphisms:** The notion of not wanting to be constrained by a particular geometry brings us to the final advantage: invariance under a common class of transformations called *homeomorphisms*; see Section 2.3.5.

### 2.1.4 Combining TDA with simple ML algorithms

For some image analysis tasks TDA methods can be used as a stand-alone tool, but for the majority of tasks, one would first use TDA to extract topological features, then afterwards add a *simple* machine learning algorithm, as shown in Figure 2.1(b). For example, the sample application in Section 2.4 uses TDA followed by a support vector machine (SVM). TDA can thus be viewed



(a) Pure ML approach: image information extracted using a complex ML model.



(b) TDA approach: image information extracted using TDA followed by simple ML model.

**Figure 2.1:** Two different ways to extract desired information from imagery: (a) using a complex ML model, typically a deep neural network; (b) using TDA followed by a simpler machine learning method. The latter can lead to more transparent and computationally efficient approaches.

as a transparent means to construct new, physically meaningful and interpretable features that may reduce the need for black-box machine learning algorithms. Using TDA in this way can support the goals of creating ethical, responsible, and trustworthy artificial intelligence approaches for environmental science outlined in [51], since transparency is a key requirement for ML approaches to be used in tasks that affect life-and-death decision making [68], such as severe weather forecasting.

### 2.1.5 Objectives and organization of this article

As mentioned before, we are not attempting to set a new benchmark for accuracy in classification, nor are we declaring that this method renders existing techniques obsolete. Instead, we seek to raise awareness of a promising technique with significant potential for ES applications and provide the reader with a high-level understanding of how TDA works, what sorts of questions can be asked using TDA, and how the answers obtained can be interpreted and understood. The case study in Section 2.4 provides examples of the sorts of questions TDA can help to address, including reports of negative examples, i.e. situations in which persistent homology is *not* able to distinguish between classes, which are as informative as positive examples in order to understand the best use of TDA.

The remainder of this article is organized as follows. Section 2.2 discusses in detail the sample application of classifying the mesoscale organization of clouds. Section 2.3 provides an introduction to the key concepts of topological data analysis. Section 2.4 illustrates the use of these TDA concepts for the sample application from Section 2.2, in combination with a simple support vector machine. In particular, in Subsection 2.4.4, we provide a detailed and novel discussion of the characteristic image-level features that our combined TDA-SVM algorithm uses to classify. This highlights the ability to identify which learned patterns can be exposed and to discuss these in the original feature space, which is one of the greatest strengths of persistent homology and TDA. Section 2.5 provides an overview of advanced TDA concepts that are beyond the scope of this paper. Section 2.6 provides conclusions and suggests future work.

### 2.1.6 Narrative and Contributions

This project is the one that set the tone for what the rest of my Ph.D. would be. In the summer of 2020, I had started exploring projects with both Dr. Henry Adams and Dr. Emily J. King in the mathematics department. Late that summer, Dr. Imme Ebert-Uphoff, a member of the Electrical and Computer Engineering faculty and the machine learning lead at the Cooperative Institute for Research in the Atmosphere (CIRA) approached Dr. Adams with the idea of using topological data analysis and other mathematical tools to study satellite imagery of clouds. Dr. Adams brought this up to me as a potential research avenue, and after a couple meetings I suggested that Dr. King would likely also have complementary areas of expertise to bring to the project that I wanted to learn about.

At this point, the goal was to detect convection in satellite imagery, with the motivation that human analysts use texture as a key characteristic in detecting convection, while previous machine learning approaches had focused more on overall brightness than textural properties. The hope was that TDA could emphasize the texture and thus make the machine learning project easier. We pursued this line of investigation for about a year overall, including several meetings with CIRA researchers in large and small groups to both get feedback on our methods and to raise awareness of the existence of TDA tools. In particular, I was mostly using datasets created and pre-processed by Dr. Yoon-jin Lee at this point, as she was simultaneously working from a more pure machine-learning perspective on the same problem.

Ultimately, the TDA tools I could use did not prove as fruitful as we had hoped on this problem. The key issue was that the most distinctive properties of convection only appeared in animations or time series data, as bubbles appeared and disappeared, and the available data was too discontinuous in time to usefully apply existing TDA time series techniques such as vineyards. I didn't feel like I was able to make useful progress on this first problem, so after some conversations with Dr. Adams, Dr. King, and Dr. Ebert-Uphoff, we broadened the scope of the problem beyond detecting convection. I started testing methods for classifying cloud organizations using TDA, at first using unsupervised methods such as $k$-means clustering on TDA summary statistics. After some initial interesting results, Dr. Ebert-Uphoff brought up the Rasp et al. dataset that we ended up using in this chapter as a possible proving-ground for this application of TDA. The existence of this large, hand-labeled dataset made it much easier to evaluate how successful the algorithm was, and to understand how TDA algorithms responded to various types of features.

I experimented with a number of different methods through the summer and fall of 2021, and by late 2021 had mostly developed the methods used in this paper. Throughout this time, I was meeting weekly with Dr. Adams, Dr. King, and Dr. Ebert-Uphoff. While many Ph.D. research topics in mathematics are hypotheses that need to be proven, this research project felt far more exploratory: the question for me was rarely "How do I do the next step?" but "What is the next step?", and these weekly research meetings helped both by making me explain what I had done and was thinking about (which often prompted new ideas) and as a way to get feedback and ideas for next steps.

By early 2022, I had most of the theory, methods, and results presented in this chapter set, and turned to writing. Over the spring semester, I collaboratively wrote this paper with Dr. Adams, Dr. King, and Dr. Ebert-Uphoff. I was the main author, writing the vast majority of the text and making the final decision on all edits, but I did receive several rounds of edits from my co-authors, and they did write some portions of the text, particularly regarding connections between this work and the wider research community, both in atmospheric science and TDA. As part of the editing process, I edited all these portions that I did not write (along with everything I wrote) to ensure that the paper had a consistent authorial voice. The remainder of this chapter outside this subsection is presented as it appears in *Artificial Intelligence for the Earth Systems*, a journal of the American Meteorological Society.

## 2.2 Guiding Application - Classifying the Mesoscale Organization of Clouds from Satellite Data

To illustrate the use of TDA we consider the task of identifying patterns of mesoscale (20-20,000km) organization of shallow clouds from satellite imagery, which has recently attracted much attention [18, 62, 78]. Climate models, due to their low spatial resolution, cannot model clouds at their natural scale [29, 61]. Since clouds play a major role in the radiation budget of the earth [49], the limited representation of clouds in climate models causes significant uncertainty for climate prediction [29]. There has been progress in addressing this limitation from the climate modeling side, e.g., using ML to better represent sub-grid processes [5, 47, 61, 88].

A different approach is to build a better understanding of cloud organization in satellite imagery [18, 62, 78]. One goal is to track the frequency of occurrence of certain cloud patterns across the globe, reaching back in time as far as satellite imagery allows, to better understand changes to the underlying meteorological conditions. To this end, in 2020 a group of scientists from an International Space Science Institute (ISSI) International Team identified the primary types of mesoscale cloud patterns seen in Moderate Resolution Imaging Spectroradiometer (MODIS; [34]) True Color satellite imagery, focusing on boreal winter (Dec-Feb) over a trade wind region east of Barbados [78]. Using visual inspection they identified four primary mesoscale cloud patterns, namely sugar, gravel, fish and flowers, shown in Figure 2.2. Subsequent study of these four cloud types using radar imagery [78], and median vertical profiles of temperature, relative humidity and vertical velocity [62], indicate that the four cloud types occur in climatologically distinct environments and are thus a good indication of those environments.

While humans are fairly consistent at recognizing these four patterns after some training, it is difficult to describe them objectively so that a machine can be programmed to do the same. Deep learning offers a potential solution; however, most deep learning approaches require a large number of labeled images to learn from.

### 2.2.1 Approaches for dealing with lack of labeled samples

[62] solve the lack of labeled data for this application by a crowd-sourcing campaign using a two-step process. First they developed a crowdsourcing environment and recruited experts to label 10,000 images. Experts used a simple interface to mark rectangular boxes in the imagery and label them with one of the four patterns. The labeled data set enabled the use of *supervised* learning

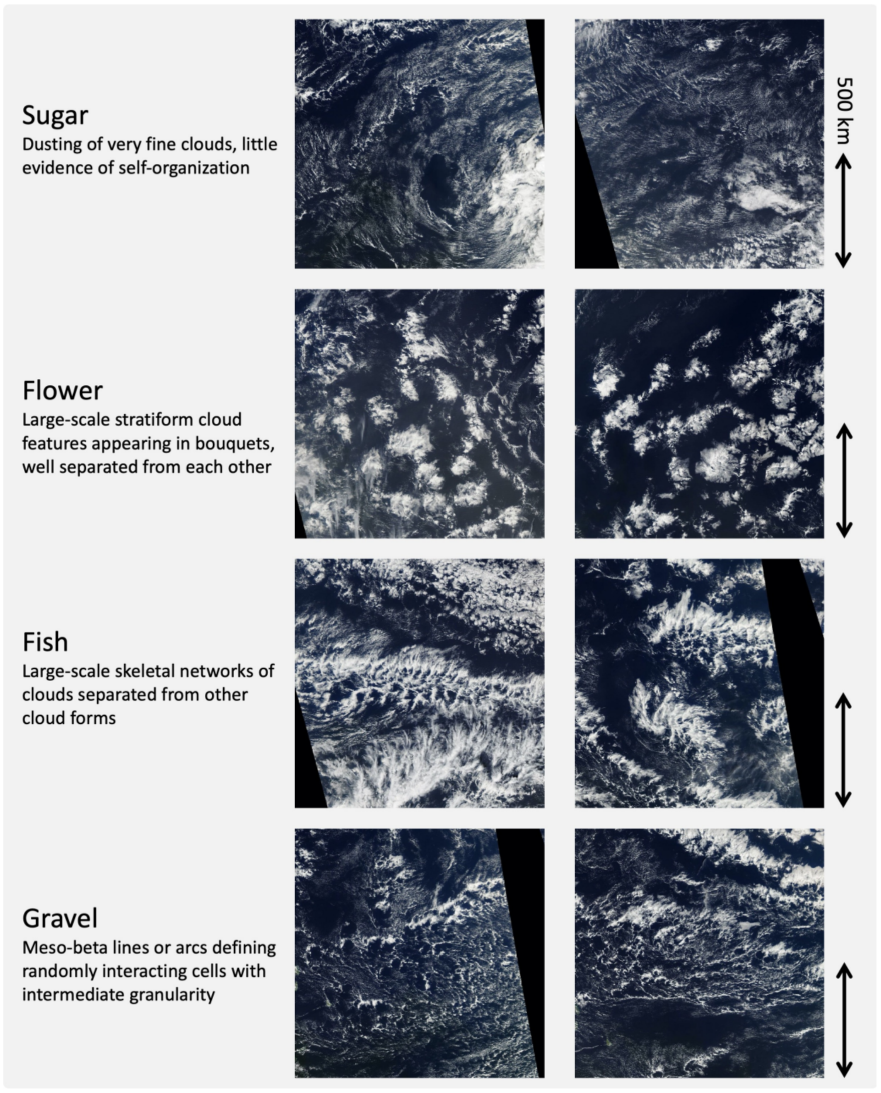

Sugar
Dusting of very fine clouds, little evidence of self-organization
Flower
Large-scale stratiform cloud features appearing in bouquets, well separated from each other
Fish
Large-scale skeletal networks of clouds separated from other cloud forms
Gravel
Meso-beta lines or arcs defining randomly interacting cells with intermediate granularity
500 km

**Figure 2.2:** Examples of the four cloud types from the sugar, flowers, fish, and gravel dataset from [62]. Note that [62] use the term “flower”, while we follow [78] in referring to this type as the plural “flowers”. *Image credit: Figure 1 in [62]. © American Meteorological Society. Used with permission.*

algorithms as a second step, i.e. the algorithms were supplied with pairs of input images and output labels and then trained to estimate output labels from given imagery. Two types of supervised deep learning algorithms were developed, one for object recognition and one for segmentation. Both algorithms performed well [62].

In contrast, *unsupervised* learning approaches seek to develop models from unlabeled data samples. Clustering—which divides unlabeled input samples into groups that are similar in some way—is a classic unsupervised learning algorithm. For example, [18] trained an unsupervised deep learning algorithm, in combination with a hierarchical clustering algorithm, for a closely related application, namely grouping image patches from Geostationary Operational Environmental Satellite (GOES; [69]) imagery into clusters of similar cloud patterns. Their algorithm identified a hierarchy of clustered mesoscale cloud patterns, but since the classes of cloud patterns were generated by a black box algorithm rather than by domain scientists, their meaning is less understood than the four patterns from [62]. Indeed, a necessary step that comes after the unsupervised learning is to test whether the patterns identified by an algorithm correspond to climatologically distinct environments, and if so which ones.

TDA is an alternative approach to address the lack of labels. With TDA we seek to match imagery to the original four classes identified by [78], yet only require a small number of labeled samples. We map patches of the MODIS imagery into topological space, then investigate whether there are significant differences in the topological properties that we can leverage to distinguish the patterns. TDA can thus be viewed as a means of sophisticated feature engineering, giving new, physically meaningful topological features. Our motivation is that this approach would allow us to identify the well established patterns from [78], but with two key differences: (1) this approach does not require a large number of labels (less crowdsourcing required); (2) this approach is more transparent than the supervised (such as [62]) and unsupervised (such as [18]) deep learning approaches, since topological properties can be understood intuitively.

We note that TDA can also be used in an unsupervised fashion similar to the approach of [18], only with more transparency and computational efficiency. On its own, TDA provides an embedding of the image data. However, rather than the embeddings being a learned property of a neural network whose properties can only be inferred after it is trained, and then only with difficulty, the TDA embedding is deterministically based on topological properties of the image. For this primer, however, we focus on the supervised task of identifying the previously established patterns of [78].

### 2.2.2 Dataset details and preprocessing

The dataset from [62] provides approximately 50,000 individual cloud type "annotations" (where each annotation is a rectangle placed on an image surrounding a particular cloud type) on around 10,000 base images. To evaluate the quality of these crowd-sourced annotations, [62] used a comparison of Intersection over Union (IoU) scores (also known as Jaccard index [25, 39]) between annotators analyzing the same image, and their analysis indicated that these annotations were generally of high quality. See Figure 2.2 for examples of these cloud types; in general, sugar type clouds are small, relatively uniformly distributed clouds; gravel type clouds are somewhat larger than sugar clouds, and tend to show more organization; flowers type clouds are yet larger clouds that clump together with areas of clear sky between; and fish type clouds form distinctive mesoscale skeletal patterns. Each image in the dataset is a $14^\circ \times 21^\circ$ (lat-lon) visible-color MODIS image from the Terra or Aqua satellite. On these images, annotators could draw rectangular annotations encompassing a single cloud type, and could apply as many annotations to each image as they desired, so long as each annotation encompassed at least $10\%$ of the image.

As we will discuss later, persistent homology takes as its input a space with an intensity value at each point, which in our case corresponds to a grayscale image. The MODIS images in the dataset from [62] were NASA Worldview True Color images in RGB [34], which we converted to grayscale using the python package `pillow`, which uses the ITU-R BT.601-7 luma transform [38] for computing intensity from RGB input:

$$I = 0.299R + 0.587G + 0.114B.$$

This is a transform originally developed for television broadcasting and approximates the overall perceived brightness for each pixel, which is appropriate here as the NASA Worldview True Color images are a close approximation of what a human observer in orbit would see. We note that this is a difference between our work and that of [62], as they used the RGB images throughout.

## 2.3 Introduction to Topological Data Analysis

In this section we provide a brief introduction to relevant mathematical topics. For more details, we refer readers to [10], [30], and [24].

### 2.3.1 Topology

In the broadest sense, topology is the study of the fundamental shapes of abstract mathematical objects. When we speak of the "topology" of an object, we speak of properties that do not change under a smooth reshaping of the object, as if it is made of a soft rubber. Some example properties include: how many connected components the object contains, how many holes or voids it contains, and in what ways the object loops back on itself. In this paper, we focus on the first two properties: connectivity and holes.

### 2.3.2 Homology

Homology is one of the tools from topology that focuses on connectivity and holes. The $d$-dimensional homology $H_d$ (for $d \in \mathbb{Z}_{\geq 0}$) counts the number of $d$-dimensional holes (or voids) in that object. For $d = 0$, the 0-dimensional homology $H_0$ captures the number of connected components present in an object. For $d \geq 1$, the homology $H_d$ captures holes—a 1-dimensional hole is one that can be traced around with a 1-dimensional loop (like a loop of string), while a 2-dimensional hole is a void. As shown in Figure 2.3, these holes and the surrounding surface need not be circular. Because homology is only interested in counting the presence of these features, it is invariant under any transformation of the space that does not create or destroy any holes or components. In our application of grayscale images, no holes of dimension 2 or larger can exist, as that would require a dataset that is at least 3-dimensional.

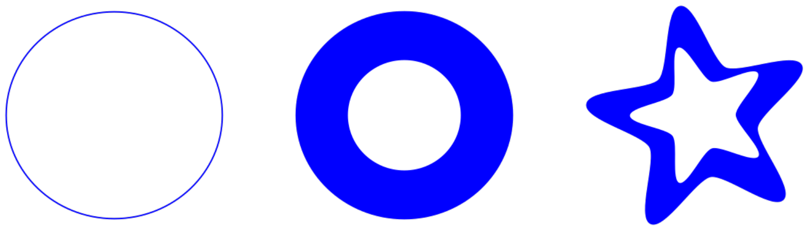


**Figure 2.3:** Three shapes that each have the same homology—a single connected component, a single 1-dimensional hole, and no higher-dimensional holes.

### 2.3.3 Persistent homology

While homology focuses on global features of the space, there is an extension, known as *sublevelset/superlevelset persistent homology*, that captures more small-scale geometry [24]. Superlevelset persistent homology is the primary tool from TDA we use in this paper, as it gives the best descriptor of image texture.

For an example of superlevelset persistent homology being computed on a simple surface, see Figure 2.4. The input to superlevelset persistent homology is a $d$-dimensional space plus an intensity value at every point. In our example, this is a grayscale image with two spatial dimensions ($d = 2$) with the pixel values as intensities. This input is then converted into *superlevelsets*: each superlevelset is a binary mask of the original space, in which only points that have an intensity value *greater* than a particular cutoff value have been included. As this cutoff value sweeps down from the maximum intensity, the homology of each superlevelset is computed and the cutoff values at which homological features (connected components, holes) appear and disappear are tracked.

For our example, this means that as the cutoff value decreases, more and more pixels with gradually decreasing intensities are included in the superlevelsets, and we track the connected components and holes that appear and disappear. Because we are using superlevelsets, in which

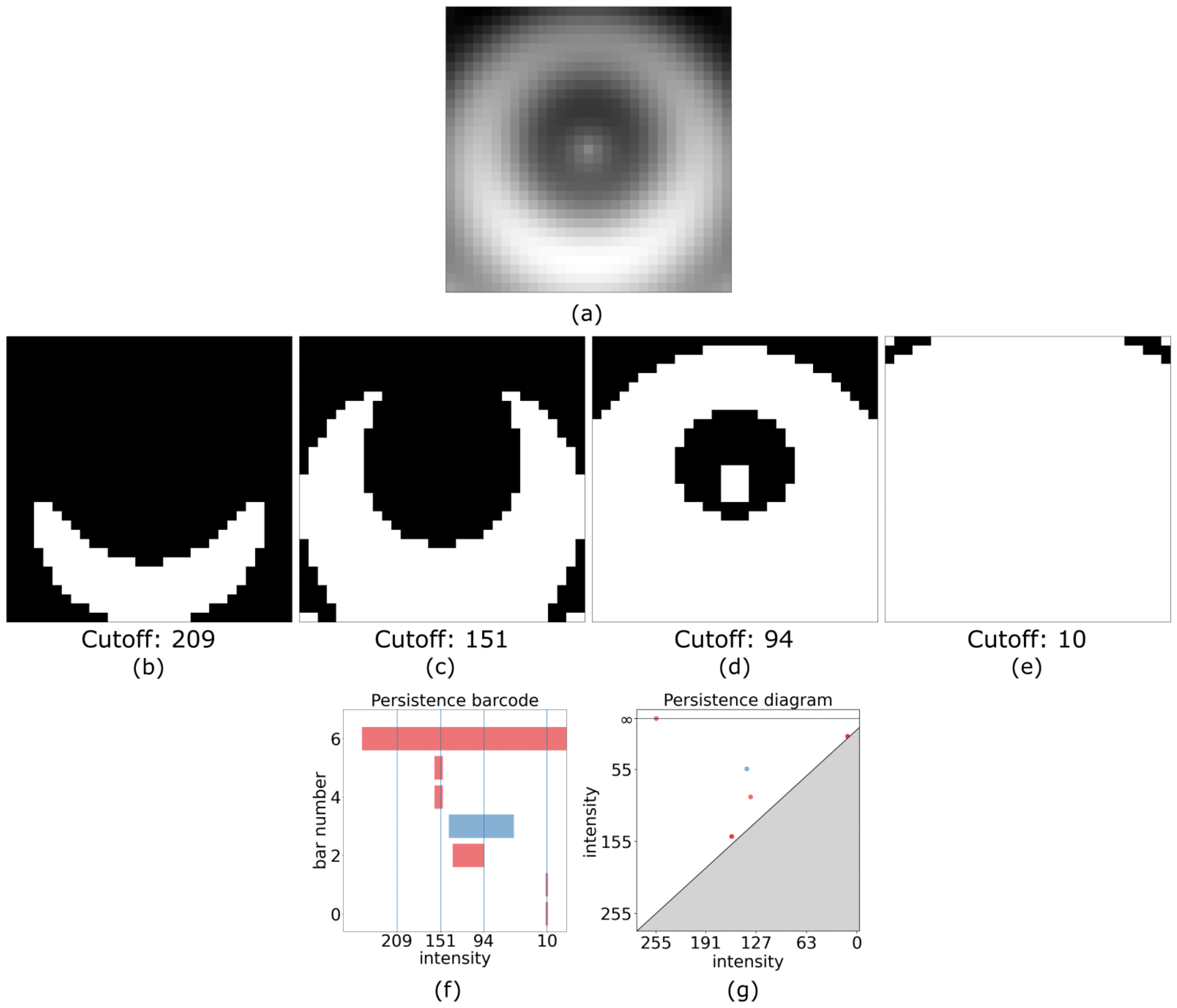


**Figure 2.4:** A grayscale image (a) ranging between intensity 0 for black and 255 for white. Four superlevelsets from (a) at cutoff values 209, 151, 94, and 10 are shown in (b) through (e), respectively. The pixels included in the superlevelsets are colored white. The persistence barcode for (a) is in (f), with vertical lines indicating the intensities corresponding to the four superlevelsets in (b) through (e). The equivalent persistence diagram is in (g). In the barcode, diagram, and landscape, red elements (bars, points, and lines, respectively) indicate connected components ($H_0$ features), while blue elements indicate 1-dimensional holes ($H_1$ features).

we start by including the highest intensities, we can view connected components appearing at high intensity value as being analogous to cloud tops, which are typically brighter, and holes as darker regions within these bright clouds.

This added interpretability motivates our focus here on *superlevelset persistent homology*, which are a simple variation (reflection) of the more commonly used *sublevelset persistent homology*. Sublevelset persistent homology is computed the same way, but instead of each set including all the pixels with intensities above the cutoff value, pixels with intensities below the cutoff value are included, and the cutoff value is viewed as sweeping from low intensities up to high intensities. Throughout this paper, we will frequently omit the prefix "superlevelset" in "superlevelset persistent homology" and simply use "persistent homology" to refer to this technique—this should not be confused with the persistent homology technique which takes as its input a cloud of data points [10].

In practice, it is not necessary to compute the homology for infinitely many superlevelsets—there are algorithms which discretize the data and then use linear algebra to implement this computation efficiently. These implementations are fast for low-dimensional data (e.g., the $2$-dimensional grayscale images used in our guiding example) but become more resource-intensive when the input space consists of higher order tensors. In this work, all homological and other TDA computations were performed using the GUDHI software package in Python [50].

### 2.3.4 Persistence barcodes and diagrams

There are two main ways to display the output of persistent homology: persistence barcodes (Figure 2.4f) and persistence diagrams (Figure 2.4g).

In a barcode, each homological feature that appears is represented by a horizontal bar, which stretches from the cutoff value at which the corresponding feature first appears (is *born*) to the value at which it disappears (*dies*). Because we are using superlevelset persistent homology, our cutoff values are decreasing; thus, the intensity values on the $x$-axis are decreasing from left to right. The *persistence* of each feature is the length of its bar. To distinguish between different homological classes, we color the bars depending on what dimension the homological feature is. We use red bars for 0-dimensional features (connected components), and blue bars for 1-dimensional features (holes). The $y$-axis of a persistence barcode counts the number of bars, typically ordered by birth value.

A persistence diagram contains the same information as a persistence barcode, but represents each feature as a point rather than as a bar. In persistence diagrams, both the $x$-axis and $y$-axis represent intensity. The $x$-coordinate of this point is given by the birth cutoff value, while the $y$-coordinate is the death cutoff value. Because features always die at a higher cutoff value than they are born, all points lie above the diagonal line $y = x$. The persistence of a feature is represented by how far a persistence diagram point lies above the diagonal. We present persistence diagrams here to familiarize the reader with their use in, e.g., [40], [83], and [74], but for our case study we focus on barcodes and landscapes.

In persistent homology, there are features that have infinite persistence—features which are born at a particular intensity, but never die. The most common example of this is that the first connected component to appear will eventually become the only remaining connected component, as all other components eventually merge into it at high enough cutoff values. These infinite-persistence points are represented as infinite bars (rays) in persistence barcodes, stretching out of the frame to the right, and as infinite points appearing on a special "$+\infty$" line in persistence diagrams.

### 2.3.5 How to read and interpret persistence barcodes

To demonstrate how to read a persistence barcode we return to Figure 2.4. The four vertical lines in the barcode (f) correspond to the four superlevelsets in the middle row, where white pixels show regions included in the superlevelset. In (b), we see the first connected component appear, corresponding to the top, infinite-length red bar in the barcode. In (c), two small components in the lower corners appear, corresponding to the two short red bars in the barcode crossed by the second vertical line. The short length of these bars indicates that these components are short-lived, and soon merge into the larger component. In (d), we see both the central 1-dimensional hole, which corresponds to the blue bar, and the connected component within that hole which corresponds to the red bar that is about to end near the third vertical line in the barcode. Finally, in (e), we see almost the entire image is in the superlevelset, as the cutoff value is very small. However, the upper two corners have just been included as two new components, which are even more short-lived than the lower corner components, as indicated by their extremely short red bars.

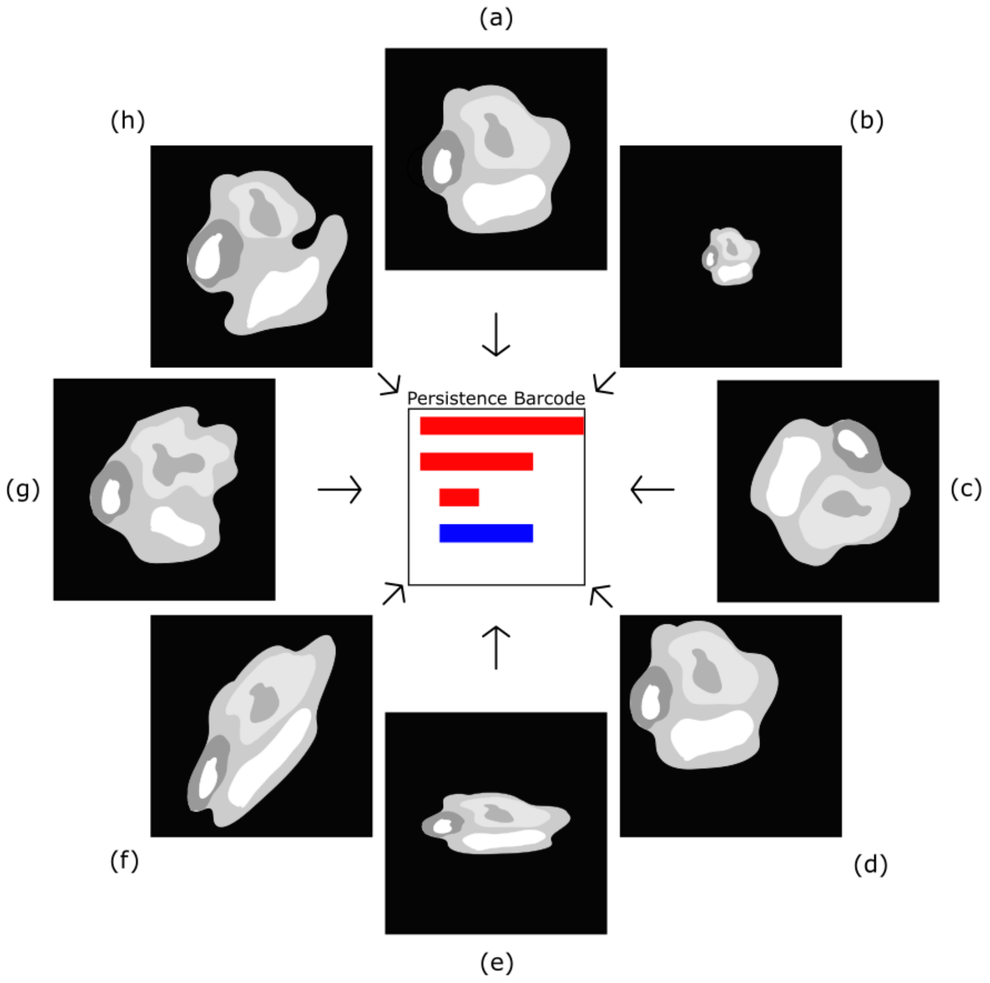


**Figure 2.5:** Examples of deformations that all result in the same persistence barcode. The original image is shown in (a), and the transformations are as follows: scaling in (b), rotation in (c), translation in (d), uneven scaling in (e), shearing in (f), and general homeomorphisms in (g) and (h).

The first thing to notice about this barcode is that there are relatively few red bars, apart from the infinite-length bar, and those bars are quite short. This indicates that few connected components appear and disappear as we scale through intensity values, and thus the base image is quite smooth. There is one red bar of reasonable length, so we would expect there to be one somewhat significant "bump", a bright region surrounded by darker regions, which is precisely what we see in the middle of Figure 2.4a. We also notice that there is one relatively long blue bar, which tells us that there is a hole (dark region surrounded by brighter regions) which persists for a relatively wide range of intensities.

Persistent homology is invariant under *homeomorphisms* of the input space, which are continuous deformations with continuous inverses; see Figure 2.5. Examples include all the rigid motions of the plane (rotation and translation), affine transformations (scaling, skewing, etc), as well as more radical reshapings, so long as no "ripping" occurs. Superlevelset persistent homology is invariant over all such transformations. So, a cloud that has been reshaped, expanded, and moved but which retains the same overall texture as in its original incarnation would have the same superlevelset persistence barcodes. See Section 2.5 for some brief comments on versions of persistent homology that can distinguish between such different deformations of an image.

### 2.3.6 Persistence landscapes

Persistence barcodes and diagrams have a drawback: they are not always convenient inputs for use in machine learning tasks, as described by [7], [2], and [54], since they do not naturally live in a vector space. To deal with this, we use *persistence landscapes* to summarize and vectorize the persistence diagram [7]. A persistence landscape is a collection of piecewise-linear functions that capture the essence of the persistence diagram.

An example of a persistence landscape computed from a small persistence barcode is shown in Figure 2.6. We separate out a particular homological dimension (e.g., $H_0$ or $H_1$) and remove any infinite bars, then create a new figure containing a collection of isosceles right triangles with hypotenuses along the $x$-axis, one for each bar in the barcode. These triangles are scaled so that the triangle corresponding to a bar is the same width as that bar. We view this collection of triangles as a "mountain range", and begin to decompose it into landscape functions. The first landscape function is the piecewise-linear function that follows the uppermost edge of the union of these triangles, i.e., it is the top silhouette of the mountain range. To compute the next landscape function, we delete the first landscape function from the mountain range, then find the piecewise-linear function that follows the uppermost edge of this new figure at every point, and so forth for the further landscape functions. This collection of piecewise-linear functions is the persistence landscape. The $x$-axis still represents intensity, while the height of each peak is proportional to the length of the bar it came from, and is thus a measure of persistence.

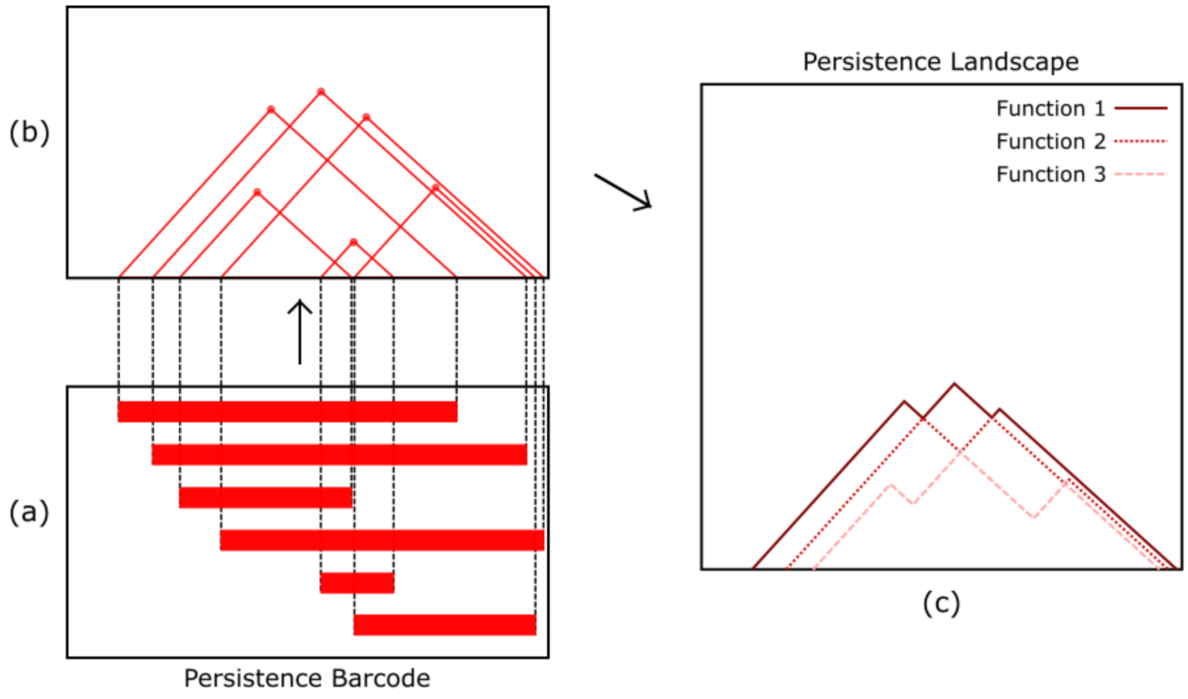


**Figure 2.6:** The process of computing a persistence landscape from a barcode. Beginning with the barcode in (a) (which already has the infinite bar removed), we raise a "mountain" above each bar to obtain the "mountain range" in (b). Our first persistence landscape function is the piecewise-linear function that follows the highest edges of the mountain range in (b), shown as the solid line in (c). The next function in (c) is obtained by deleting in (b) the lines corresponding to this first function, then finding the piecewise-linear function that follows the highest edges of this modified mountain range in (b)—this is the dotted line in (c). Further landscape functions are computed similarly, by deleting previous functions in (b) and tracing along the highest remaining edges. In (c), only the first three landscape functions are shown.

This representation is stable—small changes to the input will only result in small changes in the persistence landscape [7]. While the entire persistence landscape determines a persistence barcode exactly, in our work we retain only the top several persistence landscape functions, which means that we obtain a descriptive summary capturing the information of high-persistence points, but ignore some information about low-persistence points.

To compute landscapes, we once again use the GUDHI software package running in Python [50].

### 2.3.7 How to read and interpret persistence landscapes

We now look at a realistic example in Figure 2.7. Reviewing the barcode in 2.7b, we first notice two red bars with high persistence—the infinite bar, as well as another that stretches nearly all the way across the barcode. This indicates that there are two high intensity regions that are separated by a dark region. Over middling intensities, there are few bars, indicating that outside the two bright regions we already identified, there is little going on. Finally, when we get to lower intensities (darker regions), there are many short bars, representing subtle variations in the dark regions. This information, however, is somewhat hard to read in a barcode with as many bars as this. Thus, we turn to landscapes as a way to summarize this information in a more readable format.

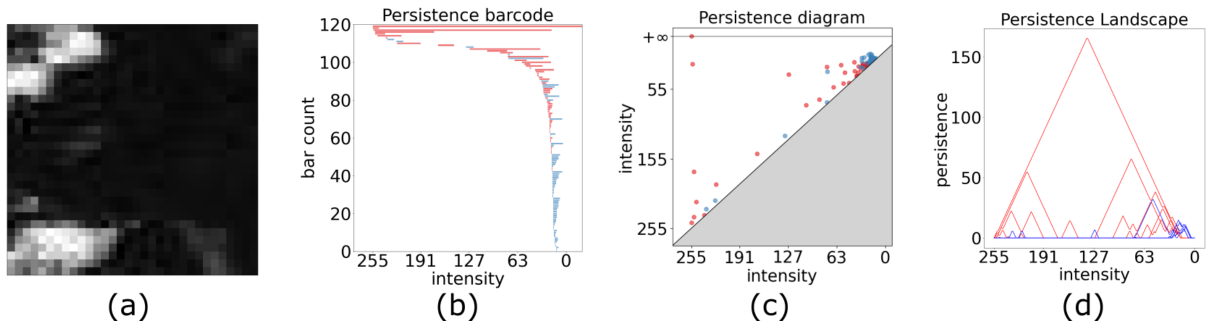


**Figure 2.7:** (a) A 32 × 32 sample image from the grayscale MODIS imagery used in the sugar, flowers, fish, and gravel dataset, along with (b) its persistence barcode, (c) persistence diagram, and (d) persistence landscape with the first 5 piecewise-linear functions for each of the $H_0$ and $H_1$ classes.

In the landscape in Figure 2.7d, the single tall red mountain indicates that the original image (Figure 2.7a) contains two connected components that persist over a large range of intensity levels (as the infinite-persistence component is implicit in the landscape). The only blue mountains in the landscape are much smaller than the red mountains and are mainly to the right of them. This indicates that while the original image contains numerous holes within the connected components, they only appear near the bottom end of the intensity range, i.e. the holes do not appear until we have started including relatively dark regions. As an example, consider the single extremely dark pixel in the upper left-hand corner adjacent to the bright clouds, and surrounded by a moderately dark region. This hole contributes a moderately tall blue peak far to the right in the landscape, as it does not appear until the relatively dark region surrounding it is included into the bright adjacent component, but will not fill in until the nearly black pixel in the middle of the hole is included.

In general, high-persistence features (long bars, tall mountains) give information about large-scale features—the presence of two bright clouds in Figure 2.7a, for example. On the other hand, low-persistence features (short bars, small mountains) give information about texture. In 2.7, the short bars and small mountains appearing at lower intensity values indicate that the background darkness in the image is relatively noisy, rather than being uniformly black or smoothly graded. The few small blue mountains in Figure 2.7d indicate that the bright clouds also contain some textural elements - regions of slightly darker cloud within brighter regions.

## 2.4 Environmental Science Satellite Case Study

Now that we have established the basic theory of TDA, we return to its application to classifying mesoscale clouds.

### 2.4.1 Adapting persistent homology to this dataset

We want to use persistent homology and landscapes to compare the clouds present in the rectangular annotations on images in the dataset of [62], so we need to find a consistent vector representation for these annotations. While the overall images in the dataset are of consistent size

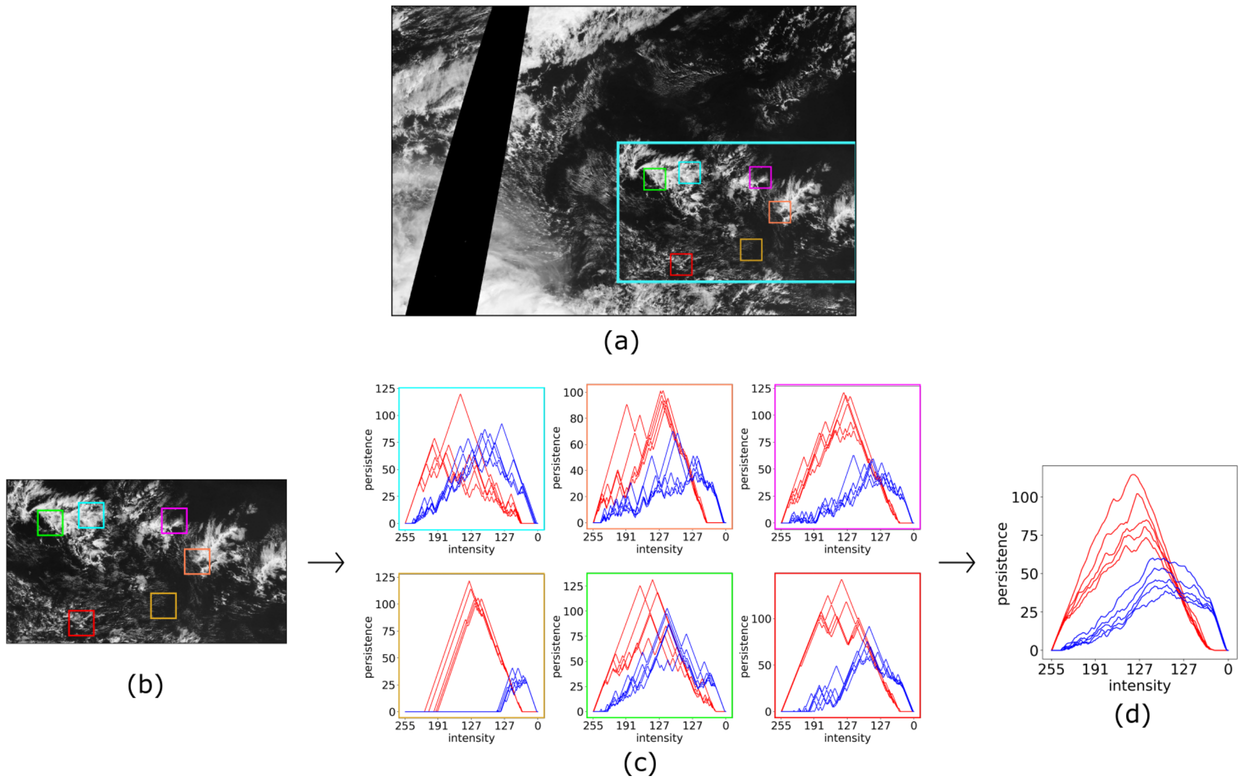


**Figure 2.8:** An illustration of the annotation and subsampling process. In (a), we see a full $14^\circ \times 21^\circ$ image, with an example Fish annotation outlined in blue, with the subsample boxes in various colors inside. In (b), we zoom in on this annotated region, including the same color-coded subsamples, and in (c) we see the corresponding landscapes for each subsample. Finally, in (d), the resulting averaged landscape is displayed.

($1400 \times 2100$ pixels), the annotations are not. An annotated region covering more area has inherently more complexity, which would yield a barcode with more bars (and thus a different vector representation) than a smaller annotation. To account for this, we implemented a subsampling routine, which is illustrated in Figure 2.8. For each annotation, we randomly chose six $96 \times 96$ pixel regions (the *subsamples*) and computed their persistence landscapes individually—this is shown in Figure 2.8 (a) – (c). Each landscape consisted of 10 piecewise linear functions: five giving information on connected components (plotted in red) and five giving information on 1-dimensional holes (plotted in blue). The height of each function was recorded at 200 evenly spaced locations along the intensity axis, giving a vector of length 200 representing that function—these 10 vectors of length 200 were concatenated to yield a vector of length $2000$ (i.e., a point in $\mathbb{R}^{2000}$), representing the persistence landscape of that particular subsample. This vectorization is shown in Figure 2.9. At that point, the persistent homology of each annotation was represented by a small point cloud of six points in $\mathbb{R}^{2000}$, one for each of the subsamples. To obtain a single point to represent the annotation, we took the geometric vector average of the six points in the point cloud, giving us the single vector that we can use to compare and analyze our annotations. Additionally, we can display and interpret this average vector as a landscape, in that it can be displayed as 10 functions, five representing the persistence and intensity ranges of connected components and five representing the same for 1-dimensional holes. This averaged landscape is visualized in Figure 2.8(d). In particular, we can view (for instance) the first 200 values of the average landscape vector as the heights of the first average landscape function at the 200 evenly-spaced intensity values where we sampled the piecewise landscape functions. We can view these values as coming from two equivalent formulations: first, as described above, as coming from a pointwise vector average, or second, by taking the average height of the first piecewise landscape function at each of the 200 evenly-spaced intensity values across the six subsamples.

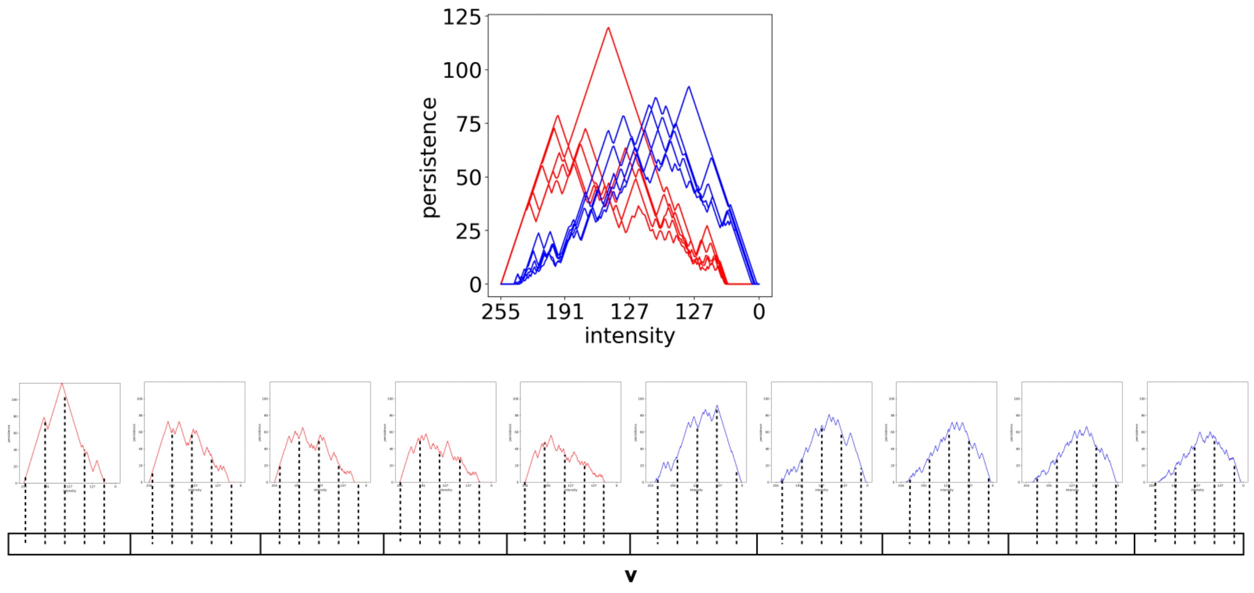


**Figure 2.9:** A cartoon showing the sampling and concatenation of a persistence landscape into a vector. In this figure, we see the landscape, each landscape function individually, and the process of sampling each landscape function evenly and then concatenating the results into a single vector, $v$.

### 2.4.2 Dimensionality reduction and adding a simple machine learning model to build a classifier

Once we obtained vectorized representations of each annotation, we sought to visualize the dataset. As $\mathbb{R}^{2000}$ is not visualizable, we applied a dimensionality reduction algorithm to yield a representation that we can plot. We used principal component analysis (PCA), as it is a widely-used and relatively simple technique, which in our case produced quite good results. We found that patterns in the data were visible upon projecting down onto the first three principal components, which captured over 90% of the variation in the high-dimensional data. We also note that the principal component vectors from repeated random samplings were extremely consistent, indicating that our projections were quite stable.

Once the data were projected down to three dimensions, we could visualize the data as a point cloud, with points colored according to which cloud pattern they represent. As a note, the PCA algorithm was entirely unsupervised with regards to these cloud pattern labels—it used only the vectorized representation of the persistence diagram.

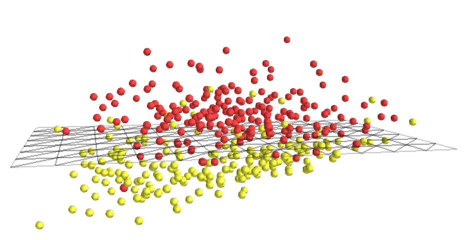
We analyzed each of the six pairs of classes separately by training a support vector machine (SVM, [4]) to find the plane that best separates the two classes of projected data in three dimensions. We considered running the SVM on the high-dimensional data, but initial testing indicated that this tended to overfit the data, and actually resulted in reduced classification performance. The SVM was trained on a random sample of 350 annotations of each class, then performance metrics were computed for a test set of 200 random annotations of each class. For visualizations of the test data and SVM separating plane, as well as performance metrics for each of the six pairwise comparisons, see Figure 2.10.

### 2.4.3 Results - what initial patterns emerged from applying persistent homology?

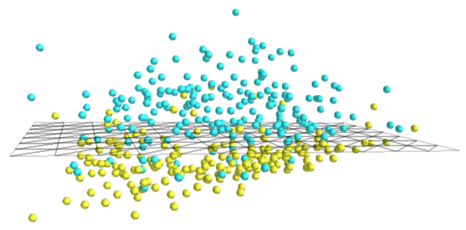
Overall, the projected points separated well, with one notable exception. We began our investigation by comparing the sugar vs. flowers patterns, as these are visually the most dissimilar (Figure 2.2). As expected, these classes had the most distinct separation out of the six possible pairs, as seen in Figure 2.10a. While the separation was not perfect, most of the error comes in the form of some intermingling near the separating hyperplane. The performance of the algorithm in separating these classes was striking, given that only a small, random subset of each annotation was included, and that the PCA projection algorithm was entirely blind to the data labels. This example is a clear indicator of the potential that persistent homology has to usefully extract textural and shape differences in satellite imagery.

While not quite as exceptional, the flowers and gravel patterns also separate well, as seen in Figure 2.10c. There is again a degree of intermingling near the separating plane, and in this case that intermingling extends a bit farther to either side. This is what we would expect, based on the visual presentation of the cloud regimes—the gravel class falls somewhere between sugar and flowers in terms of cloud size and organization.

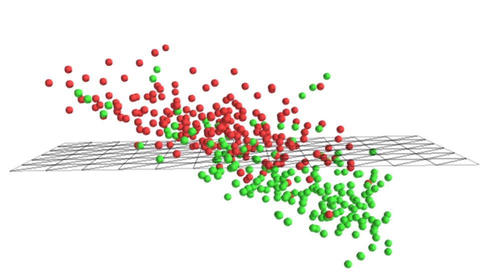
We begin to see the algorithm struggle a bit more when we attempt to separate the sugar regime from the gravel regime. As we can see in Figure 2.10e, the intermingling of data points stretches



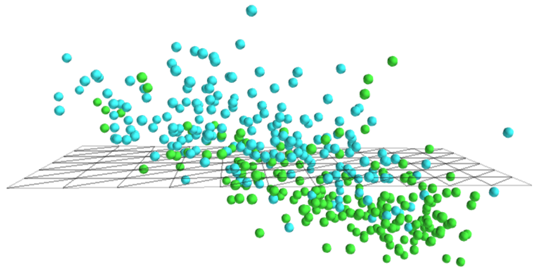
(a) Sugar vs. flowers. Test data performance: 89.25%.



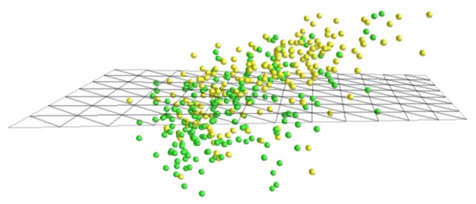
(b) Fish vs. sugar. Test data performance: 86%.



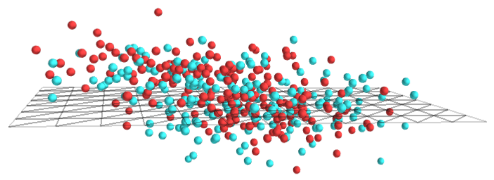
(c) Flowers vs. gravel. Test data performance: 81%.



(d) Fish vs. gravel. Test data performance: 79.5%.



(e) Sugar vs. gravel. Test data performance: 71.25%.



(f) Fish vs. flowers. Test data performance: 57%.

**Figure 2.10:** Plots of the embedded landscape test points for each pairwise combination. Flower points are red, sugar points are yellow, gravel points are green, and fish points are blue, with color representing the class assigned in the crowdsourced data set in [62]. The SVM separating plane for each pair is shown in wireframe, and the percentage of correctly classified points is reported.

throughout the point cloud, although there is still a difference in densities between the classes on either side of the separating hyperplane.

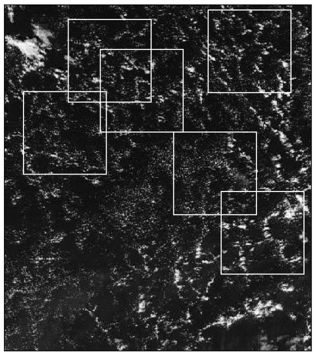
However, there are two classes that are effectively indistinguishable by this algorithm: the flowers and fish patterns. The plot of these points can be seen in Figure 2.10f, and it is apparent that there is no effective linear separator between these classes. While there is a "separating" hyperplane plotted, it is much less relevant in this case than in the others; the data points are remarkably evenly-mixed. This proved to be the case even when more principal components were included. A potential explanation for why the algorithm struggles so much with this task is that the distinguishing features of fish vs. flowers are simply too large-scale for the subsampling technique to pick up on. The fish pattern is characterized by its mesoscale skeletal structure, particularly in its difference from the flowers regime, which is more randomly distributed. This mesoscale organization is simply not visible to the subsamples, as the $96 \times 96$ patches are too small to detect that skeletal structure. Future analyses could include using larger patches to better capture these features and perhaps distinguish between these classes more effectively. We also note that in [62], the fish pattern was the most controversial amongst the expert labelers, so it is perhaps not surprising that our algorithm also struggles.

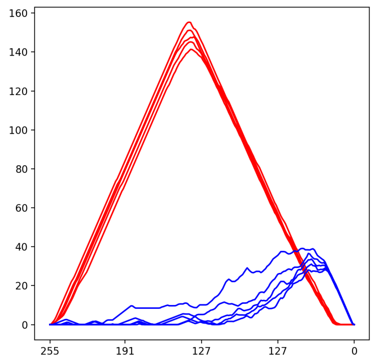
When we look at fish vs. sugar and fish vs. gravel in Figures 2.10b and 2.10d respectively, we can see how similar these plots appear to those in Figures 2.10a and 2.10c, in which flowers was compared with sugar and gravel. This similarity is made even more remarkable by the fact that the sugar samples in these plots were drawn separately rather than being reused for the pairwise comparisons (and similarly for the gravel samples). While the algorithm is not doing well at distinguishing between fish and flowers, we can at least see that its behavior is consistent: fish and flowers are projected similarly into the 3D embedding space, so they compare similarly with the other classes.

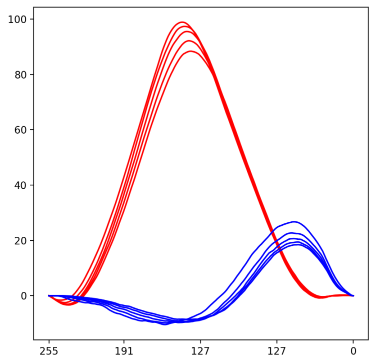
Overall, this case study suggests that it is possible to use persistent homology to quantify and understand the shape and texture of satellite cloud data. While there are cases where the algorithm struggles, these are understandable in terms of the visual task being requested, and are internally consistent from sample to sample. Moreover, in the cases where the algorithm does well, it does so consistently across repeated samples, and suggests that when these tools are appropriately applied, excellent results can be obtained from very limited sample sizes.

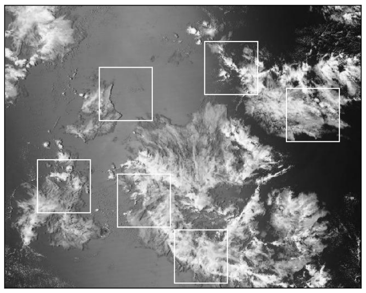
### 2.4.4 A novel interpretation method - deriving interpretations in terms of weather and homology

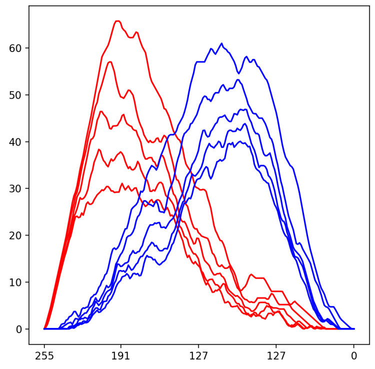
As an example of how this separation can be interpreted, we examine the case of sugar vs. flowers. Recall that in Figure 2.11a, we saw that this pair of classes had the strongest separation in the dataset.


(a) Most extreme sugar example.


(b) Landscape for the sample in (a).


(c) "Virtual" sugar landscape.


(d) Most extreme flowers example.


(e) Landscape for the sample in (d).

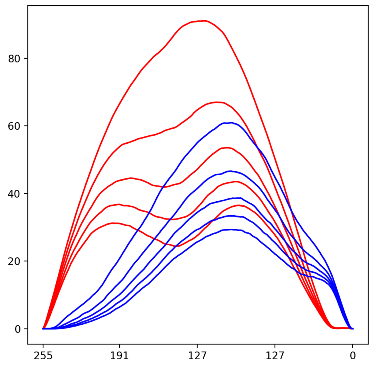

(f) "Virtual" flowers landscape.

**Figure 2.11:** Samples showing the sugar and flowers samples farthest from the separating hyperplane (in (a) and (d)), and their landscapes (in (b) and (e)). The "virtual" landscapes obtained by traveling along the line normal to the separating hyperplane are shown in (c) and (f).

To begin, we explore what can be learned just from the summarized data, without looking at examples. To discover what the separating plane between sugar points and flowers points represents, we create "virtual" landscapes. We first lift the separating plane in $\mathbb{R}^3$ to the hyperplane in $\mathbb{R}^{2000}$ consisting of all the points that project (under PCA) into the separating plane in $\mathbb{R}^3$. Next, we find the line normal to this hyperplane that passes through geometric center of the data. Finally, we choose points on this line that fall at the outer extent of the data point cloud. These points live in the landscape embedding space ($\mathbb{R}^{2000}$), but are not sampled data points. However, by applying the inverse landscape embedding, we can visualize the landscape-like set of curves that would give this embedded point. Virtual landscapes for sugar and flowers can be seen in Figures 2.11c and 2.11f, respectively.

An advantage of this approach is that it synthesizes trends from the real data into a readable, controllable format that demonstrates how SVM is separating these classes. When we compare these virtual landscapes with the actual landscapes farthest from the separating hyperplane (seen in Figures 2.11b and 2.11e), we can see that the virtual landscapes are smoother, but that the overall shapes are remarkably similar.

We can also interpret the shapes of these landscapes in terms of the features present in the images. Let us examine the images and corresponding landscapes in Figure 2.11. The most prominent feature in the two landscapes is the tall red peak in the sugar landscape, shown in Figure 2.11b. Recall that the red lines denote 0-dimensional homology (connected components) while the blue lines denote 1-dimensional homology (holes). This red peak represents the presence of strongly persistent connected components, i.e., separated regions of bright cloud strongly contrasting against a much darker background. The sharpness of this peak also indicates that these features are similar in both the intensity of the cloud top and the intensity of the surrounding background. The comparatively low blue curves with only a small peak at the end indicate a lack of 1-dimensional homological features (holes), and thus the texture within connected components is relatively uniform. Looking at the image in Figure 2.11a, we see these observations borne out: there are numerous small clouds of similar brightnesses which stand in stark contrast to the overall uniformly dark background, matching the tall $H_0$ peak. Because these clouds are relatively small, there is little discernible texture within each cloud, which corresponds to the relative absence of $H_1$ features.

On the other hand, the flowers landscape in Figure 2.11e displays a lower red peak, with more separated curves. This lack of concentration indicates that there is more variation in the intensity values at which connected components appear (the brightest part of the component) and at which they merge together (the intensity of the bridge connecting that component to another), while the lower height indicates that these features are overall less persistent—they merge into one another more rapidly. Additionally, there is a much stronger $H_1$ signal in this case than in the sugar landscape, meaning that the connected components have more internal texture, with numerous holes appearing and disappearing over a wide range of intensity values. These observations match with what we see in Figure 2.11d. The clouds in this image are much larger and cover more of the frame, with varying intensities within and between the clouds, leading to the more varied $H_0$ landscape. This image also shows much more internal texture to the clouds, with far more of a dimpling effect than in the sugar example.

In summary, this example shows how the patterns learned by the TDA-SVM algorithm can be translated back to homological features which in turn correspond to weather-relevant features in the original image. This is made possible by the fact that the SVM model can be represented by a single separating plane, which can be translated back into the space of persistence landscapes and then interpreted, yielding a highly interpretable approach to the pairwise classification problem.

### 2.4.5 Comparison of this classifier to those in Rasp et al. [62]

**Accuracy:** The accuracy of our approach cannot be directly compared to the deep learning algorithms in [62] because they address different tasks. The task considered here is to (1) choose a single class (out of two) for an annotation assumed to consist of a single cloud type, (2) based on several small patches ($96 \times 96$ pixels). In contrast, the task considered in [62] is much more complex, namely to (1) assign one or more labels for an annotation; (2) based on a very large image. We choose the simpler task for our TDA approach in order to expose the properties of a

TDA algorithm, and trying to implement a multi-label assignment (for example by using a sliding window approach) likely would have made this exploration more complicated without providing new insights. However, even without a direct comparison it is obvious from the results that this first TDA-SVM approach cannot nearly achieve the accuracy of the deep learning approaches.

**Required data samples:** Our approach only requires a few hundred labeled data samples to develop a classifier. This reduces the required labeling effort by two orders of magnitude relative to the tens of thousands of labeled samples in [62].

**Computational effort:** Computations were performed on a Surface Pro 6 with an Intel Core i5-8250U CPU. The computational bottleneck in this case was computing the persistent homology—for 800 samples (and therefore 4800 subsamples to compute persistent homology for), approximately 45 minutes of wall-clock time was required. This is already much less computational time than is generally required to train a deep network, and it is likely that this could be significantly improved by parallelizing, as each sample can be processed entirely separately.

**Interpretability and failure modes:** Our approach yields a highly interpretable model that provides an intuitive explanation of how the algorithm distinguishes different classes, while the deep learning methods do not. Furthermore, the interpretation of the separation plane in our model makes it easy to provide insights into failure modes, i.e. which types of mesoscale patterns can be easily or not so easily be distinguished by their topological features, and thus by this approach.

## 2.5 Advanced TDA concepts

In this section we briefly discuss and provide references for some advanced TDA concepts that are beyond the scope of this article, along with motivations for when and why readers might find them useful.

Figure 2.5 shows that while persistent homology measures some spatial aspects of the intensity function, it is also invariant under nice deformations ("homeomorphsims") of the domain. However, there is another (very popular) type of persistent homology, constructed using growing offsets of a shape, or unions of growing balls, that distinguishes between different deformations of the domain [10, 30]. We expect that this variant of persistent homology will also find applications in atmospheric science, and we refer the reader to [83] for such an example.

We are particularly interested to explore the use of TDA to analyze cloud properties from satellite imagery, e.g., to detect convection. While the example here looked at large scale organization of clouds, to analyze properties like convection we would zoom far into a single cloud and analyze its texture, e.g., seek to identify whether there is a "bubbling" texture apparent in a considered area of the cloud. Preliminary analysis leads us to believe that it might be necessary to use more sophisticated TDA tools for this purpose than discussed here, such as *vineyards* [17] or *crocker plots* [82], which incorporate temporal context by analyzing the topological properties of *sequences* of images, rather than individual images.

We have considered persistent homology that varies over a single parameter—the intensity of the satellite image. However, one frequently encounters situations in which two or more parameters naturally arise. For example, one can perform superlevelset persistent homology on a 2-channel image, containing the intensities with respect to two frequencies, with respect to the parameter from either the first channel or the second. In these contexts, multiparameter persistence [11–13, 81] allows one to consider both parameters at once, even though the underlying mathematics is more subtle and computations are more difficult. A version of multiparameter persistence was applied recently to the atmospheric domain in [79].

Persistence barcodes and diagrams are not ideal as inputs into machine learning algorithms, because they are not vectors residing in a linear space. This is evidenced by the fact that averages of persistence diagrams need not be unique [54]. There are a wide array of options for transforming persistence diagrams for use in machine learning, including not only persistence landscapes [7] but also persistence images [2] and stable kernels [63], for example. TDA has been gaining traction in machine learning tasks as more tools become available to integrate it into existing workflows, in both neural network layers [58] and loss functions [15]. As an example application, TDA has recently been used to compare models with differing grids and resolutions [60].

There is a variant of superlevelset persistent homology, called *extended* persistent homology [24, Section VII.3], which performs two sweeps (instead of just one) over the range of intensity values. Extended superlevelset persistent homology detects all of the features measured by superlevelset persistent homology, plus more. It may be the case that one can extract more discriminative information from a satellite image by instead computing the extended persistence diagram.

## 2.6 Conclusions and Future Work

The primary contributions of this manuscript are as follows. (1) This paper presents, to the best of our knowledge, the first attempt to provide a comprehensive, easy-to-understand introduction to popular TDA concepts customized for the environmental science community. In particular, we seek to provide readers with an intuitive understanding of the topological properties that can be extracted using TDA by translating cloud imagery into persistence landscapes, interpreting the landscapes, then highlighting the topological properties in the original images. (2) In a case study, we demonstrate step-by-step the process of applying TDA, combined with a simple machine learning model (SVM), to extract information from real-world meteorological imagery. The case study focuses on how to use TDA to classify mesoscale organization of clouds from satellite imagery, which has never been addressed by TDA before. (3) The most novel contribution is the interpretation procedure we developed that projects the class separation planes identified by the SVM algorithm back into topological space. This in turns allows us to fully understand the strategy used by the classifier in meteorological image space, thus providing a fully interpretable classifier.

In future work we seek to explore several of the advanced methods outlined in Section 2.5. We believe that there are many applications to be explored with TDA, including the applications suggested by [62] for their methods, namely *"detecting atmospheric rivers and tropical cyclones in satellite and model output, classifying ice and snow particles images obtained from cloud probe imagery, or even large-scale weather regimes"* [62]. Furthermore, as discussed in Section 2.1, TDA has already been shown to be useful to identify certain properties of atmospheric rivers, wildfires, and hurricanes, and we expect TDA to find additional use in those areas as well. Our group is particularly interested in using TDA to detect convection in clouds, and to distinguish blowing dust from, say, blowing snow, in satellite imagery.

We have only scratched the surface here of exploring how TDA can support image analysis tasks in environmental science, but we hope that this primer will accelerate the use of TDA for this purpose.

## Data Availability Statement

The underlying crowd-sourced data from [62] are available at https://github.com/raspstephan/sugar-flower-fish-or-gravel. The code used in our analysis will be available as a GitHub repository at https://github.com/zyjux/sffg_tda.

# Chapter 3

# Comparing Rotationally Invariant and Conventional CNNs

## 3.1 Introduction

When one begins trying to create neural networks on spatial input data (i.e., data that is not just a list of numbers, but has the additional structure that certain values come from adjacent locations), a very natural question that arises is "how do we deal with content-preserving transformations?" For instance, if we are attempting to perform the classic example task of distinguishing between images of cats and dogs, a cat that has been rotated by 90 degrees or which appears in a different place in the image is still a cat and should be treated as such. However, with an entirely naive network architecture, a cat rotated by 90 degrees or even translated by a few pixels could be treated as entirely different. There are two approaches to enable the network to handle such transformations: first, increase the sample size and diversity sufficiently (either through data augmentation, or through increasing the number of unique samples) and make the network large enough that it can learn these equivalences; or second, design a network architecture that *a priori* enforces these behaviors.

One approach to enforce these symmetry invariances is called *group convolution* [16]. Broadly speaking, group convolution generalizes the convolution used in convolutional neural networks (CNNs) to represent a broader class of transformations. Rather than simply translating the convolutional filter around our input space as in CNNs, in group convolution, we apply the action of a particular *symmetry group* of our input space to the filters in order to build a network that respects that symmetry group. We will provide a more thorough introduction to group convolution in Section 3.2.

Basing a network on group convolution has a number of potential advantages: (1) by enforcing the symmetry group, we restrict our search space when optimizing the network, which can enable faster convergence in training; (2) the network is able to carry more information through its hidden layers without increasing the number of trainable parameters; and (3) the network is inherently robust to any transformation included in its symmetry group. As we will see later, traditional CNNs are a special case of group convolution, where the group in question is one of translations. Some of the desirable properties of CNNs, such as their comparative speed and low number of parameters compared to equivalent fully-connected networks, as well as their robustness to translations, can thus be linked to this group convolution structure.

While group convolution is theoretically elegant and has some potential advantages, it is worth investigating how those properties play out when applied to real-world data. For this chapter, we applied the ideas of group convolution to a dataset from the National Center for Atmospheric Research (NCAR). The underlying data came from a collection of runs of the Weather Research Forecasting (WRF) model described in [71, 77], on which the Hagelslag algorithm [27] was run to extract information on severe weather events. See Section 3.3 for more information on this dataset. Recent work on this problem has used conventional CNNs to build a semi-supervised algorithm to classify each storm event into one of three categories: disorganized, supercell, or quasi-linear system [76]. This is relevant for forecasting, as each category has different severe impact weather event probabilities—supercells are associated with hail, severe thunderstorms, and tornadoes, while quasi-linear systems are more associated with severe wind events. In that work, some of the variables that proved useful were the lengths of the major and minor axes of the minimal ellipse surrounding the storm. To explore the performance of group convolution on this form of data, we focused on the task of predicting the major and minor axes lengths from forecasted radar reflectivity. Our goal was to explore whether implementing rotation-invariant group convolution would yield performance gains or superior generalizability on this task as a test case to see whether it would be worth incorporating into more involved models.

### 3.1.1 Organization of this Chapter

We will begin in Section 3.2 with an introduction to the theory group convolution. Next, in Section 3.3, we will describe in more detail the dataset we use. In Section 3.4, we will discuss our results in testing this approach on the storm data, and finally in Section 3.5, we give some concluding thoughts and discuss potential future directions.

### 3.1.2 Narrative and Contributions

The majority of this work took place while I was a visitor at NCAR in the summer of 2022. I was hosted by the Machine Integration and Learning for Earth Systems (MILES) team and supervised by Dr. David John Gagne, who first brought up the topic of geometric deep learning. Through a series of meetings, Dr. Gagne, my co-advisors Dr. Henry Adams and Dr. Emily King, my committee member Dr. Imme Ebert-Uphoff, and myself collaboratively came up with the specific topic of utilizing geometric deep learning to enforce rotation invariance in a CNN on storm data. The storm dataset was provided by the MILES group, and several team members including Dr. Gagne, Dr. John Schreck, and Charlie Becker helped me understand and utilize the data.

I did the complete implementation of the rotationally invariant CNN, including translating the concepts from [6] into practical algorithms and implementing those algorithms in Python code. The resulting analysis is also principally my work. I met weekly with Dr. Gagne throughout this period to discuss results and issues, and to explore new directions of inquiry.

After the visit officially ended at the end of the summer, I continued corresponding with Dr. Gagne and continuing to work at a reduced pace on the project, and presented some preliminary results at the Data Science Seminar at CSU, as well as part of my preliminary exam. Discussions after those presentations led to several additional developments in the project, particularly the section on tuning how deep in the network rotation invariance should be applied.

## 3.2 Group Convolution

Our implementation of group convolution was inspired by the proto-book on geometric deep learning by Bronstein et al. [6] as well as Cohen and Welling's earlier work on group convolution [16]. Code implementations of these ideas can be found in the associated GitHub repository at https://github.com/zyjux/gdl-storm-mode; the specific implementation of group convolution and the RICNN class can be found in the `notebooks/imports` subfolder.

### 3.2.1 The Group Convolution Operation

We begin by separating our input into two parts. The first part is our input space, which we denote as $U$. This input space is the structure underlying all the data—when analyzing image-style data (such as was done in this chapter), this input space is the pixel grid system. The second is a signal on that input space, which we will denote $x(U)$, which assigns one or more values to each point in $U$—i.e., it is a function $U \mapsto \mathbb{R}^n$ for some $n$. In image applications, this signal is the data on the grid—in a grayscale image, this would be a single value per pixel, while for RGB images each pixel is assigned a length 3 vector. When dealing with finite images, it is frequently useful to think of our input space $U$ as being the infinite discrete plane $\mathbb{Z}^2$, and our signal $x(U)$ as being equal to 0 anywhere outside the finite region where our image is defined.

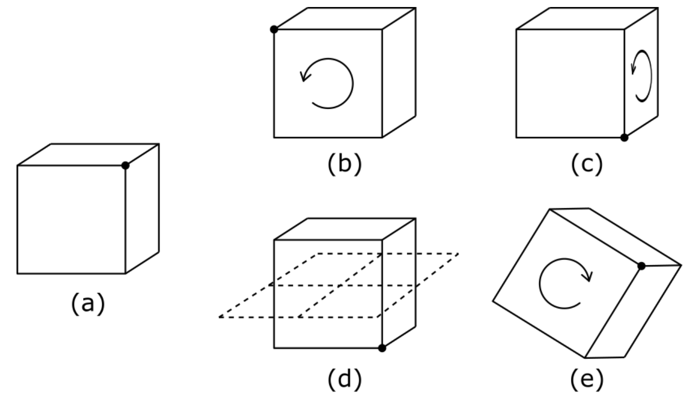


**Figure 3.1:** Some symmetries of a cube, with a marked point to illustrate what has changed. In (e), we also see a transformation that is not a symmetry.

We also choose a group $G$ of symmetries for $U$. By "symmetry", we mean an action on $U$ such that the image of $U$ under this action is equal to $U$—for some examples, see Figure 3.1. For example, if we rotate a cube by $90°$ about one of its faces, we get the same cube back, but if we rotate by an angle that is not a multiple of $90°$, the result does not lie nicely on top of our original cube, and this is therefore not a symmetry of the cube. When dealing with images, we think of them as being part of the infinite discrete plane, where the image is the only non-zero region on the plane, so it is worth considering symmetries of the plane. As one example, if we rotate the plane by a multiple of $90°$ about any point, we get another copy of the plane, and if we translate by an integer we also get another copy of the plane; both of these are symmetries because the plane is infinite, so even if the origin moves, the result is still the same set. Importantly, we also require that $G$ is a group in the algebraic sense—it is closed under a binary operation (composition, in our case), it has an identity element, and every element has an inverse. For more on symmetry groups, see [22].

To understand how we can connect these symmetries to convolution, we begin by reviewing the equation conventionally used for discrete convolution, where the input space $U$ is the discrete plane $\mathbb{Z}^2$:

$$(x \star \theta)(t) = \sum_{u \in U} \langle x(u), \theta(u - t) \rangle \tag{3.1}$$

for $t \in U = \mathbb{Z}^2$, and where $\theta$ is the *convolutional filter*. The notation $\langle a, b \rangle$ represents the *inner product*:

$$\langle a, b \rangle = a_1 b_1 + a_2 b_2 + \cdots a_n b_n$$

for $a, b \in \mathbb{R}^n$. Note that the inner product within the sum is occurring at each pixel—if the data has only a single channel, then this inner product is simple multiplication. Here, $x$ and $\theta$ are both functions $U \mapsto \mathbb{R}^n$ for some $n \geq 0$, while $x \star \theta$ is a function $U \mapsto \mathbb{R}$. Geometrically, this gives us another function on the plane, where the value at each point is obtained by translating the filter $\theta$ to that point, multiplying it against the signal $x$ and summing all the resulting values.

Our formulation for group convolution, where here we still have $U = \mathbb{Z}^2$, but we are using a different group of symmetries, is

$$(x \star \theta)(h) = \sum_{u \in U} \langle x(u), \theta(h^{-1}u) \rangle \tag{3.2}$$

where now $h \in G$. We once again have that $x$ and $\theta$ are functions $U \mapsto \mathbb{R}^n$, but in this case $x \star \theta$ is now a function $G \mapsto \mathbb{R}$. The term $\theta(h^{-1}u)$ represents the action of $G$ on the signal $\theta$—we can find out how $h$ acts on $\theta$ by looking up how $h^{-1}$ acts on $u$, and computing $\theta$ of that value. It is useful to observe that by the properties of inner products and the way this group is acting, this is equivalent to

$$(x \star \theta)(h) = \sum_{u \in U} \langle x(hu), \theta(u) \rangle;$$

that is, we can apply the action of $G$ to either the signal $x$ or the filter $\theta$. If we let $G = \mathbb{Z}^2$ acting by translation, then we recover the conventional equation for convolution from Equation 3.1, confirming that standard convolution is in fact a special case of group convolution.

The key feature of group convolution is that it is *equivariant* to the action of $G$: if $g \in G$, then $\theta \star (g \cdot x) = g \cdot (\theta \star x)$, where $g \cdot x$ represents $g$ acting on $x$. That is, applying a transformation before performing group convolution yields the same result as performing group convolution then applying the equivalent transformation. While we may ultimately want a network that is *invariant*—one that is completely unaffected by the action of $G$ on the input space, so that for $g \in G$, $\theta \star (g \cdot x) = \theta \star x$—applying invariance too early in the network can reduce its expressivity. Thus, we build equivariant layers, then later in the architecture apply a step that converts this equivariance to invariance.

For many groups, even non-commutative ones, we can write every element as some group element followed by a translation. For instance, if our group $G$ is the set of all translations and rotations of the plane, we can write every element as first a rotation then a translation. This case enables easier computation, as we can leverage the existing tools for computing convolutions. In the case described where $G$ is the set of all translations and $90^\circ$ rotations of the plane, instead

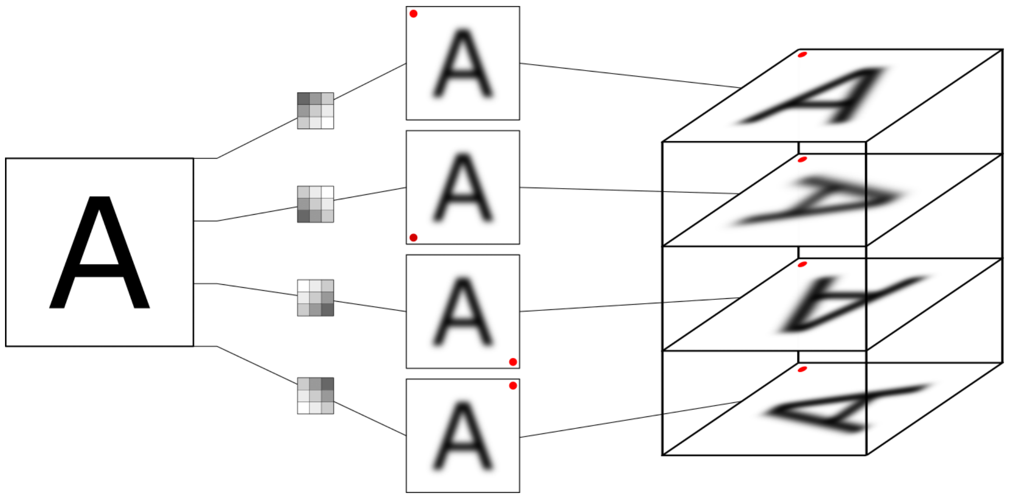


**Figure 3.2:** The initial rotationally invariant (RI) convolution layer. The input image (left) is convolved with four rotated copies of the same filter resulting in the feature activations shown in the center, then those copies are stacked so that the starting positions for each rotated filter (shown as red dots) are aligned.

of directly computing the group convolution in Equation 3.2, we can take our single filter $\theta$ and compute the four rotated versions of it, then do four convolution operations with the rotated filters. After computing these convolutions separately, we can stack them as slices in a new signal $(x \star \theta)$ on the underlying space $G$. A graphical representation of this can be seen in Figure 3.2.

When stacking the convolution up this way, we need to consider how to align the stacked images. This alignment comes from considering the “starting position” of each rotated filter $\theta$. Suppose the starting position for the un-rotated filter is in the upper left corner of the input image, and we apply rotation about the center of the input image. We choose the center of the image rather than the center of the filter because we can view $G$ as acting on either the filter or input and those perspectives should have a common center of rotation. By choosing the center of the image, we ensure that after applying a rotation to the image, we do not need to deal with recentering. Thus, if we apply a counterclockwise rotation by $90°$ to the filter, the resulting starting position is in the lower left corner; two rotations have a starting position in the lower right corner, and three rotations have a starting position in the upper right corner; these locations are marked with red dots in Figure 3.2. In separating the group $G$ into four slices organized by which rotation has been applied, we want to align the starting positions, and as such, we rotate the outputs of each convolution so that their starting positions are in the upper left corner. This also ensures the desired property that rotation of the input image corresponds to permutation of the slices in the output stack.

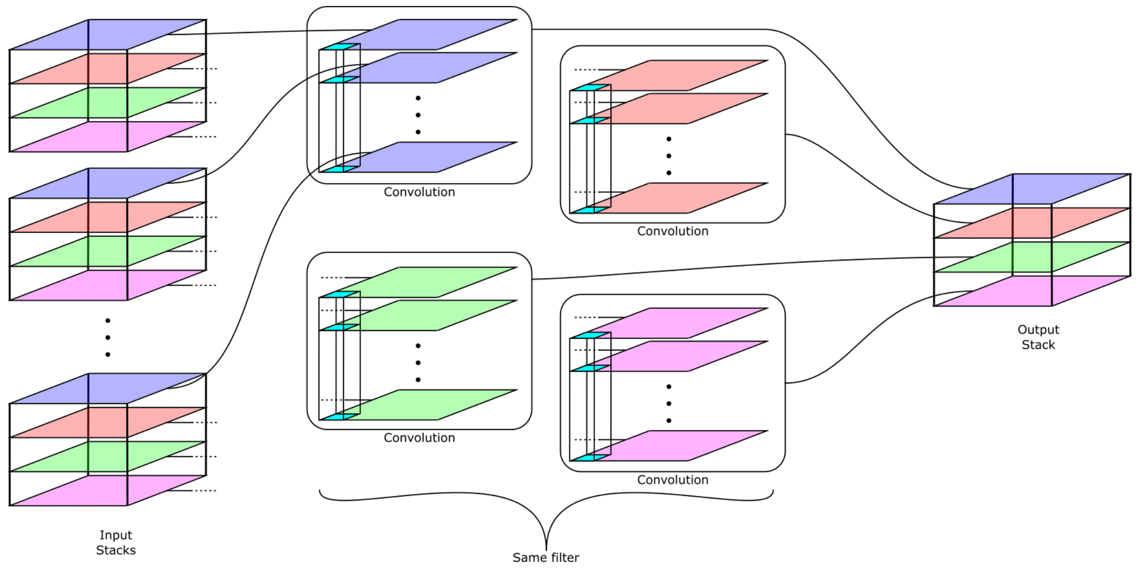


**Figure 3.3:** The internal rotationally invariant (RI) convolution operation. Note that the four convolution operations in the center all use a single, common filter. Note that all the slices of the same color in the input stacks are collected for the convolution operation in the center, as shown for the blue slices; the links for the other colors are not shown for the sake of clarity.

As we have mentioned, the output of group convolution $x \star \theta$ is a signal on $G$ rather than the input space $U$, typically the plane $\mathbb{Z}^2$. Thus, the group convolution operation we have just described only applies to the initial convolution step, when the input is a 2-dimensional image, so we will refer to this as *initial rotationally invariant (RI) convolution*. For layers after the first, we need a third, yet more general convolution operation, which we will call *internal rotationally invariant (RI) convolution*:

$$(x \star \theta)(h) = \sum_{g \in G} \langle x(g), \theta(h^{-1}g) \rangle,$$

where now $x$ and $\theta$ are functions $G \mapsto \mathbb{R}^n$, and $x \star \theta$ is still a function $G \mapsto \mathbb{R}$. It is important to note here that the action of $G$ on $x$ and $\theta$ is different than in Equation 3.2: in that case, we viewed $G$ as (for instance) translating and rotating the plane $\mathbb{Z}^2$, while in this case, $G$ is acting on itself.

In terms of implementation, this means that while we thought of the initial group convolution layer as being four distinct traditional convolutions with the same filter rotated by multiples of $90°$, we can now think of these future convolutions as being more similar to traditional convolutions, but with a higher-dimensional space for the filter to be moved through. We can achieve this as seen in Figure 3.3 by first separating out all the slices from the output stacks of the previous layer and grouping them by which slices they were in, then performing four conventional convolutions using a single, common filter on the resulting arrays. The outputs of these convolutions are then arranged as slices in the next stack corresponding to the slice they originally came from.

### 3.2.2 Resulting Architecture

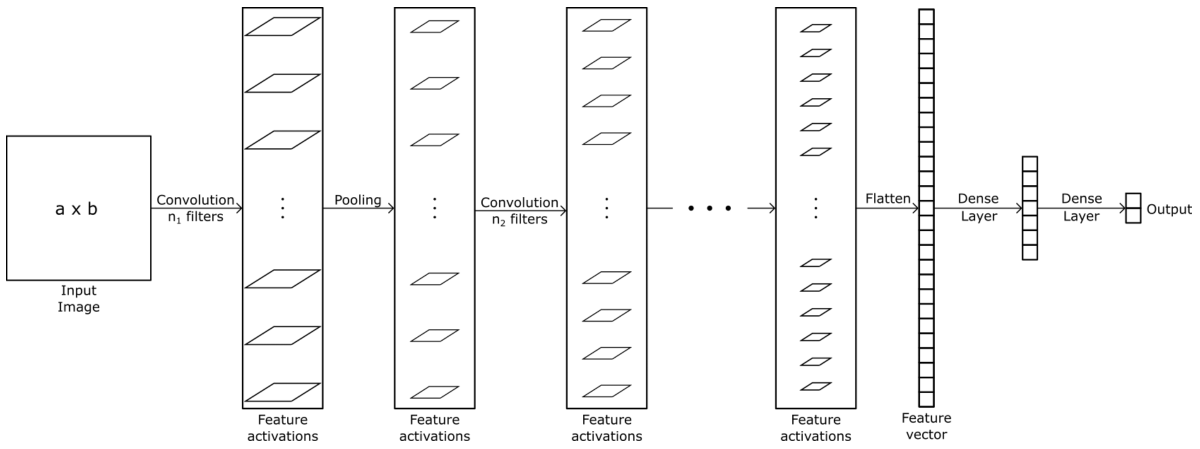


**Figure 3.4:** A conventional CNN architecture, showing the flow of data through the network.

To see how this can be used to construct a neural network architecture, let us consider Figures 3.4 and 3.5. In Figure 3.4, we see a diagram of a conventional CNN architecture. We convolve our input space with $n_1$ different filters of size $x_1 \times x_1$ (where $x_1$ is an integer, most commonly 3 or 5), which result in $n_1$ outputs of size $a \times b$, called *feature activations*. To these feature activations, we apply a pointwise nonlinear *activation function*, as well as *pooling* to allow connections between more and more distant parts of the input space. We then convolve again with $n_2$ filters, where typically $n_2 > n_1$. Note that these filters are $x_2 \times x_2 \times n_1$ as each filter collects information from all the feature activations from the previous layer. Eventually, we take the results of these convolutions and flatten all these feature activations into a single feature vector. This vector is then the input into one or more dense neural network layers, which do the final analysis and reduce the dimensionality down to our final output variables.

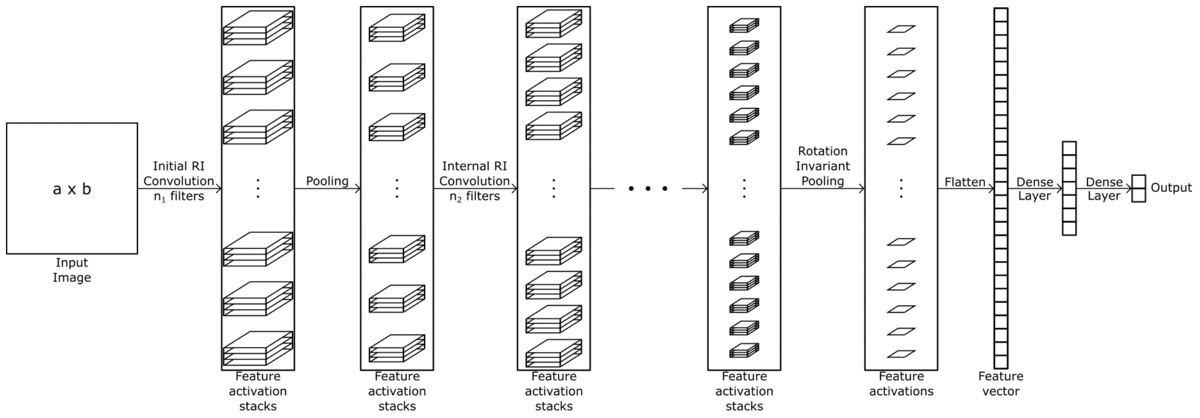

a x b
Input Image
Initial RI Convolution $n_1$ filters
Feature activation stacks
Pooling
Feature activation stacks
Internal RI Convolution $n_2$ filters
Feature activation stacks
Feature activation stacks
Rotation Invariant Pooling
Feature activations
Flatten
Feature vector
Dense Layer
Dense Layer
Output

**Figure 3.5:** Rotationally invariant CNN architecture, showing the flow of data through the network. The initial RI convolution step for a single filter is shown in Figure 3.2, while the internal RI convolution step for a single filter is shown in Figure 3.3.

Now consider Figure 3.5, which shows an example architecture of a network where $G$ consists of translations and $90°$ rotations of the plane. As in Figure 3.4, we begin with $n_1$ different filters of size $x_1 \times x_1$. We compute the rotations of these filters and use those as shown in Figure 3.2 to compute the initial RI convolution layer, the output of which is $n_1$ feature activations, each of which is a $a \times b \times 4$ stack. In this architecture, we view points on different slices as being distant from one another, so when we perform pooling, it is within each slice, not between slices. For deeper, internal RI convolutional layers, recall that we no longer need to take our filters and generate rotations of them; instead, we are simply convolving our filters through each stack, maintaining the underlying structure of the space as we do so, as shown in Figure 3.3.

After performing a number of these rotationally equivariant convolutions, we reach a point where we are ready to flatten and move to the dense part of the network. However, before we do so, we would like to move from equivariance to invariance, which we can do by pooling across the slices in each feature activation stack, as shown in Figure 3.6. For this step, any pooling method that does not depend on the ordering of the elements, such as pixelwise average or max pooling, will yield an invariant network. In our work, we used max pooling for this rotation invariant pooling layer, based on testing performed by Hong et al. in [36]. From this point on, the network is identical to a conventional CNN, except that the output will be entirely invariant to rotations of the input space.

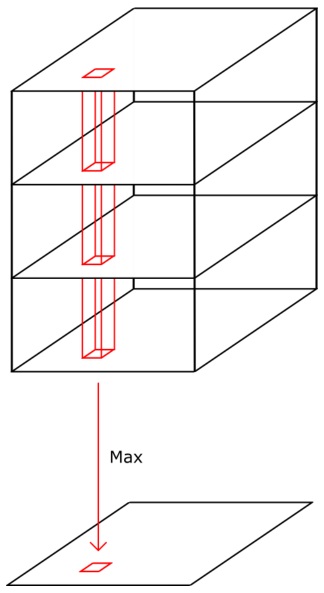


**Figure 3.6:** Rotational invariant pooling layer, using pixelwise maximum pooling.

We implemented these rotationally equivariant convolution and pooling layers and the rotation invariance pooling layer as custom layers in TensorFlow [1], the code for which can be found at https://github.com/zyjux/gdl-storm-mode. We refer to networks built using these layers as "rotationally invariant CNNs" or "RICNNS".

This approach is theoretically distinct from the third architecture type we will compare in this paper: *augmented CNNs*. In an augmented CNN, the same architecture as a CNN is used, but modifications are made to the training dataset. To *augment* the data, we create copies of each sample in the training dataset, each of which has had a different transformation applied to it, then include those comments back into the training dataset, effectively increasing its size. In particular, for the augmented CNN used here, we generated additional samples for the training dataset by taking each existing image and adding each $90°$ rotation, so that each image appeared four times, once each rotated by $0°, 90°, 180°$, and $270°$. The training dataset for the augmented CNN is thus four times as large, and the intention is that the CNN learns some degree of rotation invariance on its own, without having it hard-coded as in the RICNN.

## 3.3 Datasets

### 3.3.1 Storm Data

The dataset we used came from the underlying dataset used in Schwartz and Sobash [71] and Sobash et al. [77]. It consisted of around $50,000$ patches of forecast data from a numerical weather prediction model, each of which was $144 \times 144$ pixels in size and contained a number of different meteorological channels. These patches were centered around severe storm events, and the channel we principally used in this work was forecasted radar reflectivity, which is commonly taken as a measure of storm intensity—see Figure 3.7 for an example of one of these patches. The regions of severe storm on which these patches were centered were identified by the Hagelslag algorithm [27], which also flagged a region of pixels as being part of that storm or not; the ellipse best fitting these pixels was computed by Hagelslag, and its major and minor axes lengths reported. Throughout this chapter, all the models used these major and minor axes lengths as their target variables. In the paragraphs below, we give a more detailed and technical summary of how these data were generated.

The dataset contained 682 total runs of convection-allowing reforecast data from the Weather Research Forecasting (WRF) algorithm version 3.6.1 [75] covering the Continental United States (CONUS) region over the 2010-2019 date range. Of this dataset, we used the first 150 runs, covering the date range 2010-2012. All runs used the same physics parameters (see [77] for details), the same 3 KM grid, and the operational 0.5 degree Global Forecast System (GFS) for initial conditions and boundary conditions. These runs did not represent every day within the given date range. Instead, they represented days on which the NOAA Storm Prediction Center archives had records of severe weather events. Moreover, Sobash et al. excluded 15-July to 15-October from their date selection each year in an attempt to focus the dataset on “high-impact warm- and cool-season se-

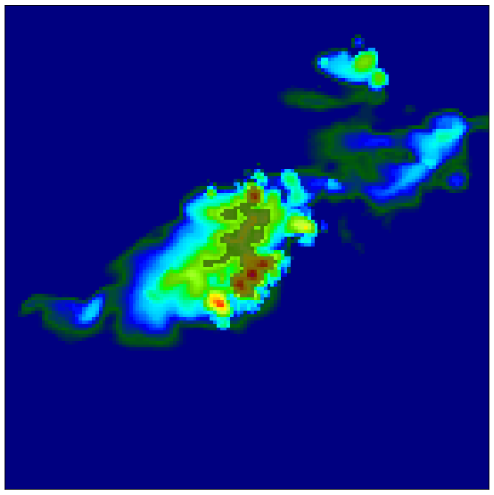


**Figure 3.7:** An example patch of forecasted radar reflectivity centered on a storm event. The dark patch in the center represents the region identified as “severe storm” by the Hagelslag algorithm [27].

vere weather events east of the Rockies, while neglecting events in the western United States and in association with landfalling tropical cyclones” [77].

The output of each run was a grid covering the entire CONUS region, which is not an appropriate input for a classification or regression CNN, as those are typically more effective when applied to smaller patches. To identify relevant portions of the dataset, the MILES group had processed these forecast runs through the Hagelslag algorithm [27]. Hagelslag is a package developed to identify and locate severe weather events in large-scale data, as well as compute various properties of the storms. Of particular relevance to this work, Hagelslag output a mask containing all the pixels it identifies as being part of the storm as well as the ellipse that best fits that storm region. The dataset that we used for this project consisted of 51,346 distinct patches from the 150 runs discussed earlier, each of which was $144 \times 144$ pixels and was centered on a single storm event identified by Hagelslag. Of note is that as the WRF forecasts covered multiple hours, the same storm could appear multiple times in the dataset as it evolved. While these temporal links were in the dataset, this was not incorporated in any of our models.

We separated this dataset into training, validation, and test sets. The training data contained 34,061 patches from 105 forecast runs between 24-October, 2010 and 02-March, 2012. The validation data consisted of 4,300 patches from 15 forecast runs between 15-March, 2012 and 07-April, 2012, while the test data contained 12,985 patches from 30 forecast runs between 09-April, 2012 and 14-June, 2012.

The primary channel that we used as an input was forecasted radar reflectivity. As a standard preprocessing step, we centered and normalized this channel across the training set, then used the mean and standard deviation computed from the training set to normalize the validation and test sets. Our output target variables were, for all the models below, the length of the major and minor axes of the storm-surrounding ellipse. As the minor axis length is (by definition) always shorter than the major axis length, sometimes by a significant margin, we also centered and normalized the target variables so that our computation of error would not be more heavily weighted towards the major axis values.

These training, validation, and test sets were used in all the following sections that reference "storm data" with the exception of one portion of Section 3.4.2, which used a different methodology for splitting these same data into training, validation, and test sets; in particular, the dataset described here was used for Figure 3.10, Table 3.2, and Table 3.4. We re-used these same datasets for all models, as we were interested in comparing models, so utilizing identical dataset minimized potential confounding sources of error in the comparisons.

### 3.3.2 Toy Dataset

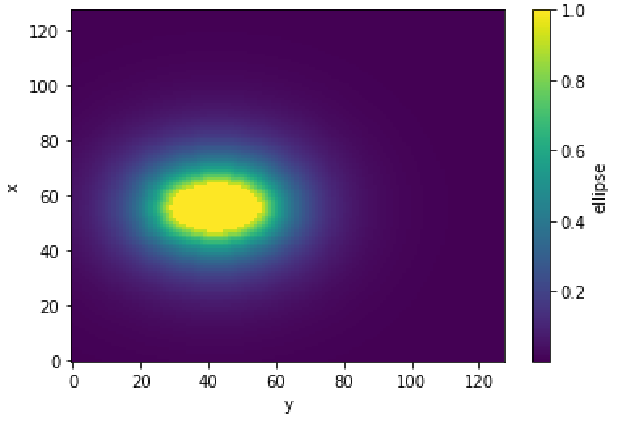


**Figure 3.8:** An example image from the toy dataset.

To test the potential of our technique in an extremely controllable setting, we first created a synthetic toy dataset with similar target variables to our eventual storm dataset, but without any of the noise or complexity of real data. This toy dataset consisted of 12,000 generated images, 10,000 of which were used for training, 1,000 of which were validation, and the final 1,000 of which were the testing dataset. Each image was $128 \times 128$ pixels, and had an ellipse of a random size and orientation placed at a random location within the image. The major axes of these ellipses ranged from 10 to 40 pixels in length, and the minor axes were bounded between 1 pixel and $2/3$rds of the major axis length. The ellipse centers were chosen so that no part of the ellipse could fall outside the borders of the image. To smooth these ellipses, we computed the distance transform of the image, in which pixels within the ellipse had a value of 0, and every other pixel had a value that was the Euclidean distance from that pixel to the nearest ellipse pixel. Finally, we took the negative exponential of this distance transform, so in the final images, pixels within the ellipse had a value of 1, and pixels around the ellipse fall off exponentially to 0. An example image is shown in Figure 3.8.

Our training target for this toy dataset was the lengths of the major and minor axes. The point of creating this toy dataset was first to debug our RICNN architecture, and once that was complete, to test that the generalization properties that we hoped RICNNs should have would actually show up in practice. To test this behavior, we set up the toy dataset so that the orientation of the major axes of all ellipses in the training and validation sets fell within $\pi/6$ of horizontal. However, the test set had unconstrained orientation, so the major axis could be in any orientation. Our motivation for setting up the test set in this fashion was to see how well the three architectures were able to generalize.

As with the storm data, these exact training, validation, and test sets were used in all following sections that reference the "toy dataset", specifically in Figure 3.9, Table 3.1, and Table 3.4.

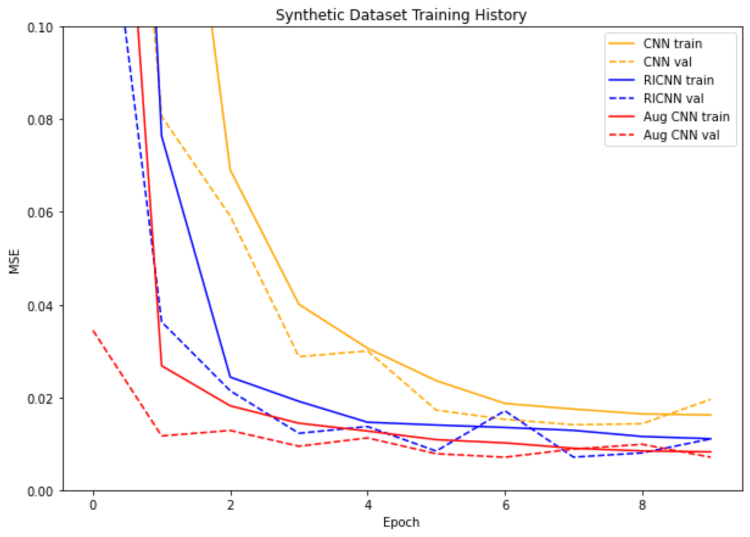


**Figure 3.9:** Mean Squared Error on the training and validation sets across the training epochs on the toy dataset. The CNN errors are the orange line, the augmented CNN errors are the red line, and the RICNN errors are the blue line.

## 3.4 Results

### 3.4.1 Initial comparison on toy dataset

We trained three models, a CNN, an augmented CNN, and a RICNN, with equivalent architectures and the same number of trainable parameters for 10 epochs using the `adam` optimizer [42] and mean squared error (MSE) for the loss. The model architecture we used consisted of five convolutional layers, all with $3 \times 3$ filters, and (starting at the beginning of the network) 32, 32, 64, 64, and 128 output channels, all using the ReLU activation function. These convolutional layers were alternated with four max pooling layers, each using $2 \times 2$ pooling fields. After the final convolutional layer (and rotation invariant pooling layer, if applicable), the data was flattened to a single long vector, to which two dense neural network layers were applied, the first of which contracted down to 32 output nodes using the ReLU activation function, and the second of which outputted the final two prediction values with no activation function. Overall, this architecture, which was used for all three model types, had a total of 204,482 trainable parameters on the toy dataset images. Throughout this paper, we present results training for 10 epochs—we tested training for up to 20 epochs, but found that performance did not typically increase, indicating that the models had converged by the 10th training epoch.

In training, all three models performed similarly—the RICNN and augmented CNN converged somewhat more quickly in terms of epochs, but each epoch took 4-5 times longer to process in wall-clock time when compared to the CNN. This behavior during training can be seen in Figure 3.9. However, on the testing data (shown in Table 3.1, along with final training and validation errors), the CNN lost all performance, going back to an error that was worse than after the first epoch of training, while the augmented CNN and RICNN suffered effectively no loss of performance.

**Table 3.1:** Training, validation, and testing mean squared errors (MSE) for CNN, augmented CNN, and RICNN on toy dataset. The training and validation sets contained only ellipses with major axis orientation within $\pi/6$ of horizontal, while the test set contained all orientations.

| Model | Training MSE | Validation MSE | Test MSE |
|---|---|---|---|
| CNN | 0.0162 | 0.0196 | 0.7100 |
| Aug CNN | 0.0082 | 0.0071 | 0.0075 |
| RICNN | 0.0111 | 0.0110 | 0.0141 |

This showed that the at least in simple cases, the RICNN's hoped-for ability to generalize did in fact hold. However, the augmented CNN still achieved overall better performance than the RICNN.

### 3.4.2 Initial Comparison on Storm Data

When moving to the real dataset, we used the same architecture as the one used on the toy dataset. With the slightly larger input patch size ($144 \times 144$ vs $128 \times 128$), the total number of trainable parameters for each model rose to $241,346$. Performance during training (shown in Figure 3.10) for the three models showed a similar behavior to that on the toy dataset: the RICNN trained more quickly at first, and achieved slightly better performance than the CNN, but in terms of ultimate accuracy, the augmented CNN was still superior. The final performance of each model on the training, validation, and test set is shown in Table 3.2.

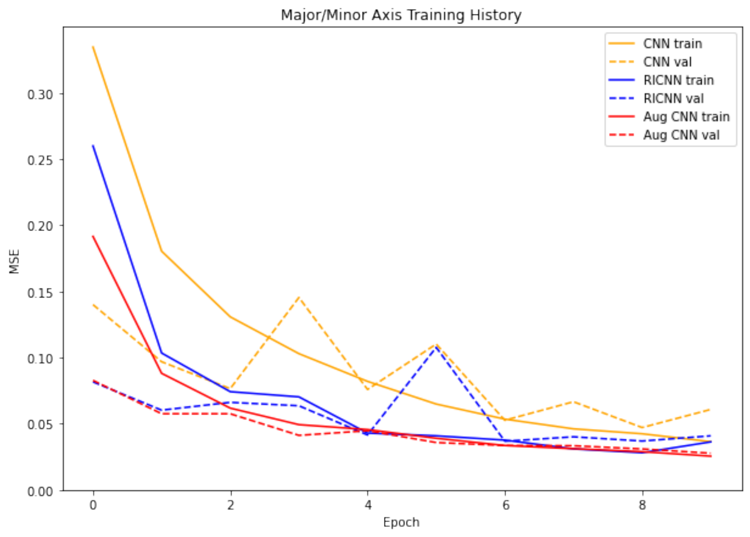


**Figure 3.10:** Training and validation errors for the CNN, augmented CNN, and RICNN during training on storm data.

**Table 3.2:** Comparison of final-epoch MSE values for the three models compared on storm data.

| Model | Training MSE | Validation MSE | Test MSE |
|---|---|---|---|
| CNN | 0.0365 | 0.0608 | 0.0705 |
| Aug CNN | 0.0256 | 0.0278 | 0.0364 |
| RICNN | 0.0363 | 0.0409 | 0.0483 |

We also wished to test whether the RICNN would improve upon the generalizability of a CNN on real data, similar to how it did in the toy dataset. However, in this case, separating out storms based on orientation for training and test data (as we did for the toy dataset) was too artificial a distinction for any real-world task. Instead, we separated our training and test data based on storm motion direction, as computed by the Bunkers Storm Motion algorithm [8]. For our training and validation sets, we used only storms moving in an eastward direction, while our test dataset used only storms moving in a westward direction. This method of separating did mean that for this test, our test set was not separated by date, but we hoped that separating by motion would make the generalization task even harder.

The results of this test are presented in Table 3.3. It is immediately apparent that while the augmented CNN and RICNN retained their ability to generalize from the training set to the test set, in this case the traditional CNN retained this ability as well. This suggests that for a given

**Table 3.3:** Comparison of final-epoch MSE values for the three models compared on storm data in which the test set is composed only of west-moving storms, while the training and validation sets contain east-moving storms.

| Model | Training MSE | Validation MSE | Test MSE |
|---|---|---|---|
| CNN | 0.0290 | 0.0684 | 0.0281 |
| Aug CNN | 0.0304 | 0.0743 | 0.0333 |
| RICNN | 0.0317 | 0.0642 | 0.0255 |

overall direction of storm motion, there was sufficient variety in storm orientation to allow the CNN to not require the additional machinery of augmentation or enforced rotation invariance to be able to generalize to other motions.

It is also notable that across all three models, the performance on the validation dataset was significantly worse than on either the test set or training set; this is surprising, as the validation set was intermingled with the training set, while the goal was for the test set to be disjoint in storm motion, and thus making predictions on the test set should have been harder than for the validation set. We do not have a satisfactory explanation for this behavior, unfortunately. One hypothesis is that when the data were split up, the validation set happened to contain a larger number of "harder" examples, or the test set overall contained "easier" than expected examples, but this is not something we have been able to investigate thoroughly.

### 3.4.3 At what layer should the network be invariant?

One aspect to model selection when building a RICNN that we mentioned briefly earlier was the choice of how deep in the network one should move from equivariance to invariance; that is, after which layer of the neural network should we apply the rotationally invariant pooling layer shown in Figure 3.5. Recall that *equivariance* is the property that for a symmetry $g \in G, \theta\star(g\cdot x) = g\cdot(\theta \star x)$, while *invariance* is the property that $\theta \star (g \cdot x) = \theta \star x$. To reduce this question to the extreme, if we were to do rotation invariant pooling using max pooling after the first layer, then the network could not distinguish between an image with a circle in one quadrant and an image with that same circle in all four quadrants. In order to explore the consequences of applying invariance at varying depths in the network, we trained a number of models on each dataset, in which we varied how many rotationally equivariant layers we applied before moving to invariance. In these tests, we also included a model in which invariance was never applied, and the model remained equivariant throughout. To keep the comparison fair, we kept the total number of layers constant, using rotationally equivariant layers before moving to invariance, and conventional convolutional layers after the invariant pooling step. The results of these tests are shown in Table 3.4.

**Table 3.4:** Error results for RICNNs run on both the toy dataset and storm dataset with invariance applied after various layers. For table entries marked "Equivariant", no rotationally invariant pooling layer was applied, and the architecture remained equivariant throughout.

Toy Dataset

| Invariance After: | Training MSE | Validation MSE | Test MSE | Running Time |
|---|---|---|---|---|
| Layer 1 | 0.0297 | 0.0414 | 0.0395 | 140.0 sec |
| Layer 2 | 0.0214 | 0.0337 | 0.0350 | 169.3 sec |
| Layer 3 | 0.0159 | 0.0181 | 0.0232 | 184.6 sec |
| Layer 4 | 0.0120 | 0.0167 | 0.0197 | 191.1 sec |
| Layer 5 | 0.0102 | 0.0060 | 0.0068 | 193.6 sec |
| Equivariant | 0.0116 | 0.0071 | 1.278 | 196.3 sec |

Storm Dataset

| Invariance After: | Training MSE | Validation MSE | Test MSE | Running Time |
|---|---|---|---|---|
| Layer 1 | 0.0714 | 0.1052 | 0.0897 | 571.6 sec |
| Layer 2 | 0.0548 | 0.0885 | 0.0711 | 707.7 sec |
| Layer 3 | 0.0468 | 0.0786 | 0.0748 | 765.6 sec |
| Layer 4 | 0.0334 | 0.0684 | 0.0667 | 791.6 sec |
| Layer 5 | 0.0293 | 0.0665 | 0.0603 | 808.9 sec |
| Equivariant | 0.0309 | 0.0684 | 0.0555 | 807.6 sec |

On the toy dataset, having invariance appears to be very important, and as we moved the rotation invariant pooling layer deeper into the network, the accuracy increased significantly, with some of the largest gains happening at later layers—this indicates that for this application, maintaining equivariance as long as possible was useful. However, having invariance present in the model was also important: the model which remained equivariant and never moved to invariance did not perform as well on the training and validation sets as the Layer 5 model, despite having more trainable parameters in the final dense layers, and failed to generalize from the limited angle training set to the test set in which ellipses could appear at any orientation. On the other hand, on the storm data, while moving the invariant pooling layer later in the architecture did in general increase performance, after the first few layers we saw diminishing returns, and on this dataset, the equivariant model performed as well if not better than the invariant models.

As equivariant convolutional layers are approximately four times as costly to compute as standard convolutional layers, each equivariant convolutional layer replaced by a standard convolutional layer can reduce computational cost and running time for the network. In Table 3.4, we also present the running time for each model, and as we can see, moving invariance deeper into the model generally resulted in an increased run-time. The correct choice for what level to apply invariance after will depend on the application and priorities of the user, as these models exhibited different behaviours even on these relatively similar datasets.

### 3.4.4 Exact versus Approximate invariance

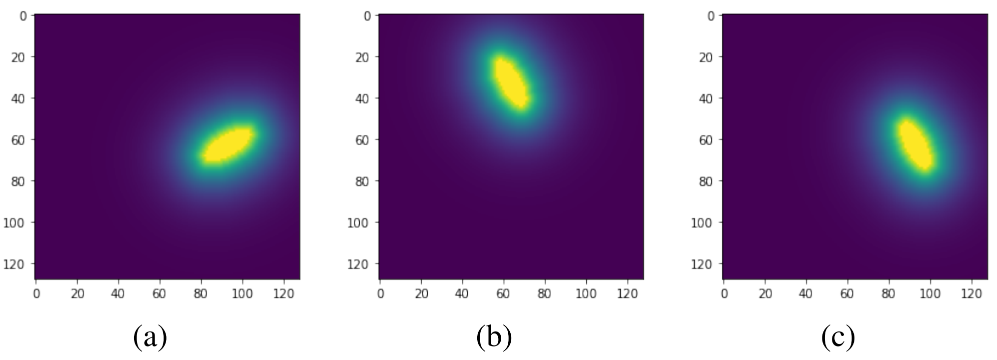


**Figure 3.11:** Examples of rotations used in this section. In (a), we have an example ellipse. In (b), we see the resulting image after rotating the image in (a) by $90^\circ$ about the center of the image. In (c), we see the resulting image after rotating the ellipse by $90^\circ$ about the center of the ellipse.

One aspect of these algorithms that we sought to compare was exact vs. approximate invariance. The RICNN algorithm enforced exact invariance to rotations by $90^\circ$ about the center of the image (as shown in Figure 3.11 (a) and (b)), which was confirmed by numerical experiments—the network gave functionally identical predictions for the same input image rotated by $90^\circ$. On the other hand, the augmented CNN did not have this constraint built in, and was simply encouraged to learn some degree of approximate invariance by augmenting the dataset. In numerical experiments, the augmented CNN treated the original image and that same input image rotated by $90°$ about its center as two distinct entries, and while its predictions in both cases were of high quality and close to the true number, they were not strongly correlated with each other. For an example of these behaviors, see Table 3.5.

**Table 3.5:** Model predictions for an example ellipse (shown in Figure 3.11(a)) as well as that same image rotated by $90°$ about the image center (shown in Figure 3.11(b)). Note that the augmented CNN's predictions differ between the original and rotated image, while the RICNN's predictions are invariant.

| | Maj. Axis Orig. | Min. Axis Orig. | Maj. Axis Rot. | Min. Axis Rot. |
|---|---|---|---|---|
| Ground Truth | 25 | 10 | 25 | 10 |
| Aug. CNN | 24.576 | 10.420 | 25.444 | 10.005 |
| RICNN | 23.982 | 9.855 | 23.982 | 9.855 |

While the RICNN was invariant to $90°$ rotations, we also investigated how these networks responded to rotations of smaller degree as well. To examine this, we used the data generation algorithm for our toy dataset to conduct two experiments by rotating and moving an ellipse of constant size within the image frame. In the first experiment, we took an ellipse and rotated it in a full circle about the center of the image in $1°$ increments. In the second, we took the same ellipse and rotated in a full circle about the center of the ellipse again in $1°$ increments. See Figure 3.11 for examples of both of these rotation types. For both experiments, we used both the augmented CNN and RICNN to predict the major and minor axes at each angle of rotation, then computed the variances of each axis across the rotations. The results are presented in Table 3.6.

**Table 3.6:** Variance in model predictions when rotating a generated member of the toy dataset around either the center of the image or center of ellipse in $1°$ increments.

Rotating about center of image

| Model | Maj. Ax. Var. | Min. Ax. Var. |
|---|---|---|
| Aug. CNN | 0.00379 | 0.00231 |
| RICNN | 0.00497 | 0.00228 |

Rotating about center of ellipse

| Model | Maj. Ax. Var. | Min. Ax. Var. |
|---|---|---|
| Aug. CNN | 0.00277 | 0.00277 |
| RICNN | 0.00407 | 0.00217 |

As we can see from this example, the strict invariance enforced by the RICNN architecture for rotations of $90°$ about the center of the image did not imply invariance to all rotations. While the network was approximately invariant to small-degree rotations, it had in most cases even higher variance than the augmented CNN on the same task.

## 3.5 Conclusions and Future Work

In this chapter, we have discussed our test of using rotationally invariant CNNs (RICNNs) on storm data, particularly in comparison to data augmentation and conventional CNNs. Overall, while the RICNN has several theoretically desirable properties (such as strict invariance to $90°$ rotations of the input space), these did not necessarily translate into useful performance gains on real-world data. The conventional CNN run on augmented data out-performed the RICNN in nearly all cases, and had faster running time, as it is using highly optimized built-in TensorFlow layers. This performance even resulted in lower variance in predictions for incrementally-rotated inputs, despite the theoretical advantage that the strictly invariant RICNN should have had on that test.

For this task, the RICNN's theoretical elegance does not justify its use over the simpler and better-performing augmented CNN, but the group convolution framework, and geometric deep learning in general, may still prove its worth on other datasets and tasks. There remain a number of open questions that would be worth pursuing in future work. For instance, how would group convolution as presented here interact with other network architectures, such as a U-Net [67] or even transformer [84]? In this work, all the networks used the same underlying architecture with the same number of filters; however, in theory, the augmented CNN should have to dedicate more filters to learning the rotationally symmetric patterns in its augmented dataset when compared to the RICNN, so would we be able to achieve similar performance by reducing the sizes of the filter banks in the RICNN? We would also like to explore uncertainty quantification methods such as deep evidential regression to see if any of these model architectures have differing aleatoric and epistemic uncertainty properties [3].

While the RICNN did not prove to be qualitatively better than augmented CNNs on this task, it did prove to be competitive in most areas, and was a flexible and customizable technique. We hope that future work will find the tasks and datasets that can more fully utilize RICNNs and other geometric deep learning toolsets.

# Chapter 4

# Using Harmonic Analysis Techniques to Enhance Gravity Waves in the Day-Night Band

## 4.1 Introduction

In this project, we present work done to identify an appropriate pre-processing algorithm that will enhance the signature of atmospheric gravity waves in night-time satellite imagery. Due to the extremely limited number of available hand-labeled samples, utilizing machine learning methods that require training (such as convolutional neural networks) has not yet yielded satisfactory results [31]. While more sophisticated machine learning methods may yet prove effective, we take a different approach and focus here on deterministic methods drawn from harmonic analysis.

### 4.1.1 Motivation

As described in [55], gravity waves are an important element of atmospheric energy transfer. These atmospheric pressure waves can be generated by a number of different energetic events, such as powerful thunderstorms, hurricanes, and strong updrafts near geographic features like mountain ranges, known as *orographic* gravity waves, as well as dynamics within the upper atmosphere such as jet stream instabilities and other less-obvious causes. These waves transmit energy between different parts of the atmosphere, and can have strong effects on upper atmosphere currents, which in turn can have impacts on weather and climate. In weather and climate models, the atmospheric processes governing the generation of gravity waves occur at smaller scales than are resolved by these models, and as such the effects of gravity waves need to be included as a summary statistic (or *parameterized*). To do this effectively, however, there is a need for reliable observations of gravity waves to calibrate and constrain these parameterizations. However, until recently there was no systematic way of observing gravity waves in the upper mesosphere (around $90$ km above the surface). Thus, most methodologies have explored different parameterizations by testing the effects of adjustments on final model results, but this is not as efficient or explainable as being able to calibrate the gravity wave portion directly.

The key contribution of [55] is the discovery of a gravity wave signature in nightglow imagery captured by the Day/Night Band (DNB) sensor of the Visible Infrared Imaging Radiometer Suite (VIIRS) flying on the Suomi National Polar-orbiting Partnership (NPP) and NOAA-20 satellites. *Nightglow* is an atmospheric phenomenon in which excited particles near the mesopause emit faint amounts of visible light, which sufficiently sensitive sensors can detect [37]. The pressure changes within a gravity wave can affect this light emission, causing wave-like patterns to appear in the nightglow. While the DNB sensor is able to detect nightglow, it is only able to do so on moonless nights, so the window of opportunity to detect a gravity wave event is quite small: there must be a suitable event occurring on a moonless night within a specific time and location window during which either the Suomi NPP or NOAA-20 satellites (both of which are polar-orbiting) pass over the event. These stringent requirements mean that detectable gravity waves are extremely rare events within the collection of VIIRS data. While there is a vast quantity of VIIRS data collected over the last decade, and therefore a large number of examples of even rare events like detectable gravity waves, the task of finding a large number of those examples by hand is intractable.

### 4.1.2 Chapter Overview

In this chapter, we examined a number of pre-processing algorithms which we hope to eventually develop into an automated system that can identify gravity waves in DNB imagery, making the process of collecting a large dataset of gravity wave observations more practical. We began by examining what is distinctive and characteristic about gravity waves in DNB imagery from an image processing perspective, and articulating what we are looking for in a detection algorithm. Starting with this allowed us to conduct a more organized and directed search for algorithms, and gave us a framework by which to assess the results beyond simple numerical results. The key features that we identified are the periodicity and linearity of gravity waves, which we attempted to leverage.

We discuss three algorithms that we tested to focus in on these properties: local autocorrelation, wavelet-based ridge detection, and the finite Radon transform (FRT).

Autocorrelation is a simple harmonic analysis tool which we applied here in a local fashion to try to detect periodicity. While this did detect some periodic gravity waves, it was not extremely successful and produced many false positives coming from city lights and cloud structures. The key issue here is that autocorrelation is sensitive to the amplitude of periodic signals, and even in normalized patches, gravity waves are frequently low amplitude signals. Next, we used an existing toolbox for detecting ridges in images using a wavelet-based edge and ridge detection method. This yielded good results in some instances when tuned properly, identifying the ridge-like structures making up the gravity waves. However, this method also had many false positives, and does not really leverage either the periodicity or linearity properties that we identified as being important. Finally, we tested the FRT, which breaks down a finite patch of prime size into all possible finite-geometry lines through that space, allowing us to focus on the linear structure of gravity waves. The finite geometry structure of these lines also meant that they are themselves periodic, and thus also respond to the periodicity of gravity waves at the same time. However, these aspects are not detected independently, and are entangled in complex ways that make the algorithm less than ideal.

All of these techniques yielded some promising results, but are not quite what is needed, so we finish by discussing two approaches that we will pursue in the near future. The first is utilizing local autocorrelation and the wavelet-based ridge detection algorithm in combination, using the ridge detection algorithm to first detect and highlight the ridge-like features of the gravity waves (perhaps with some modifications to restrict to more linear ridges), followed by using autocorrelation to detect periodicity. The second is a different generalization of the continuous Radon transform to finite spaces called the Mojette transform; in contrast to the FRT, this utilizes non-periodic lines and does not rely on prime geometry, so would hopefully give us similar results to the FRT while allowing us to disentangle the linearity and periodicity aspects.

This work is structured as follows: in Section 4.2, we discuss the properties of gravity waves in DNB imagery and the corresponding desirable properties for an ideal pre-processing algorithm; in

Sections 4.4–4.6, we present algorithms that we tested and discuss in what ways they meet or fail to meet the criteria established in Section 4.2; finally, in Section 4.7, we present our conclusions and future work.

All the code and data used for this project is available on GitHub at https://github.com/zyjux/harmonic_gw.

### 4.1.3 Narrative and Contributions

The idea to initiate this project originated in late 2021 when my committee member Dr. Imme Ebert-Uphoff brought up the problem of identifying gravity waves in DNB imagery, and particularly the extremely small size of the datasets available. My co-advisor, Dr. Emily King, made the suggestion of using a wavelet-based approach to identify the wave-like structures in DNB imagery, which we later collaboratively expanded to include more classical harmonic analysis techniques. I began to work in earnest on this in September of 2022, following the conclusion of my summer visit at the National Center for Atmospheric Research (NCAR). Throughout this project, I continued meeting weekly with my co-advisors, Dr. King and Dr. Henry Adams, as well as Dr. Ebert-Uphoff, to discuss progress, challenges, and results, as well as to brainstorm new ideas and next steps. The dataset of gravity wave examples was provided by Cooperative Institute for Research in the Atmosphere (CIRA) researchers, principally Dr. Katherine Haynes, who created and coded the core of the pre-processing routine for the DNB images including the log-scaling routine used.

The first techniques I tested were local autocorrelation and wavelet-based ridge detection. For local autocorrelation, the idea to use autocorrelation on small patches of the image came from Dr. King; based on that idea, I came up with the specific implementation presented in Section 4.4, coded up the resulting algorithms, and analyzed the resulting data. The algorithms used in the ridge detection section (Section 4.5) were developed by Dr. King's former doctoral student, Dr. Rafael Reisenhofer [66]. I utilized the MATLAB toolbox he developed, but did parameter tuning to focus the ridge detection on gravity waves, and analyzed the results.

After these techniques were giving preliminary results, I met with John Forsythe, a CIRA scientist who had studied gravity waves in the past to get input on our preliminary results. I also presented these results at a CIRA meeting, with the aim of getting more feedback on how the algorithms were doing from a meteorological perspective and what to pursue next. This feedback led to more resources for destriping algorithms, and after reviewing the feedback and results, I decided to keep looking at alternative algorithms that could focus more on the linearity of gravity waves. That discussion led to Dr. King suggesting I look at the ridgelet transform, which is described in [21]. The first step in computing the ridgelet transform is computing the finite radon transform (FRT), and I found that the FRT itself was a potentially useful tool for identifying gravity waves. I did the implementation of the FRT into Python code, as I could not find any modern code implementations of it, and analyzed the results.

After reviewing all the techniques so far, I developed the ideas for next steps presented as future work in Section 4.7, and hope to pursue those in the near future.

## 4.2 Properties of an Ideal Transform

As we discussed in the introduction, gravity waves in DNB imagery appear as variations in nightglow. These wavelike patterns are faint, but have distinctive structures, and multi-channel approaches utilizing infrared imagery can be used to more fully disambiguate gravity waves from wave-like cloud structures. In this section, we analyze the image-level features that characterize gravity waves in DNB imagery, which allowed us to perform a more focused and systematic search for a transform that responds well to those features. An example DNB image containing gravity waves is shown in Figure 4.1.

These waves have two key properties that, taken together, set them apart from other features in DNB scenes: periodicity and linearity. The algorithms we used focused on one or both of these properties, so it is worth discussing them in more depth here. First, "periodicity" here means that the gravity wave signature is periodic in space, in that it has repeated, alternating regions of light and dark. However, while this periodicity is in general similar to a sine curve wave structure, it also

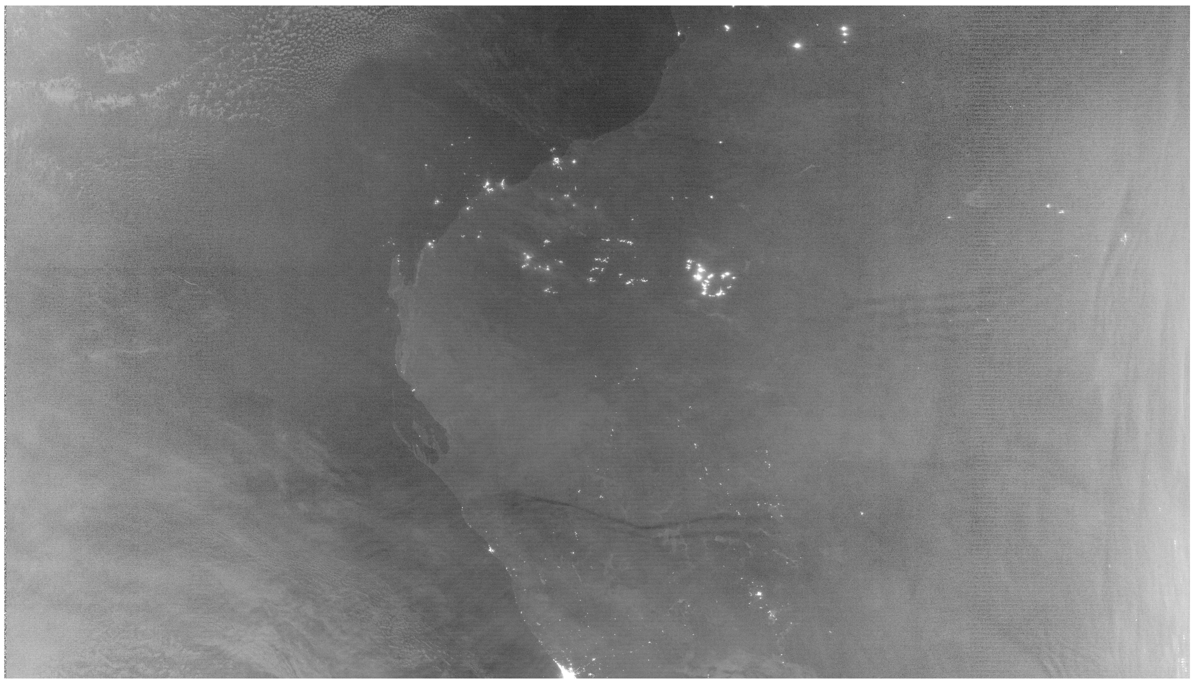


**Figure 4.1:** An example of DNB imagery containing gravity waves (bottom center, and more faintly at the center right). This image was collected by the NOAA-20 satellite over Western Australia at 17:16 UTC on May 13th, 2018.

has some dissimilarities. In particular, the brightest peaks of the wave are in general no brighter than the surrounding background nightglow, so the periodicity is principally in the valleys rather than the peaks. The other key feature is "linearity", which here refers to how the light and dark regions defining the waves are composed of relatively straight lines, at least at the scales and patch sizes at which distinguishable gravity waves appear. The ideal algorithm for preprocessing, then, is one that responds strongly to these properties of linearity and periodicity, while de-emphasizing features that do not have those properties.

## 4.3 Dataset

The dataset we are using is a small set of hand-selected VIIRS DNB [9] images that were found to have identifiable gravity waves in them. The data were originally downloaded from the NOAA Comprehensive Large Array-data Stewardship System (CLAss). Due to the number of constraints on time of day, moon phase, and atmospheric conditions that are required to make gravity waves identifiable in DNB imagery, finding examples is extremely time-consuming. Because of this, we used only 61 total sample images, which includes some duplication and negative examples—because the VIIRS instrument flies on two satellites in similar orbits, when gravity waves were identified in imagery from either the Suomi NPP or NOAA-20 satellites, imagery from the other satellite over the corresponding location was pulled as well (if available), but in many instances the gravity waves are not visible in these other images. Each sample was a $4064 \times 2304$ grayscale image, which was pre-processed using a log-scaling method and converted into $0 - 255$ radiances then stored as a netCDF4 database.

## 4.4 Local Autocorrelation

When choosing algorithms for a task, we would like to use the simplest algorithm that gives good performance, as that generally increases interpretability. To that end, we began our exploration of algorithms with *local autocorrelation*, a modification of one of the simplest analysis tools in harmonic analysis.

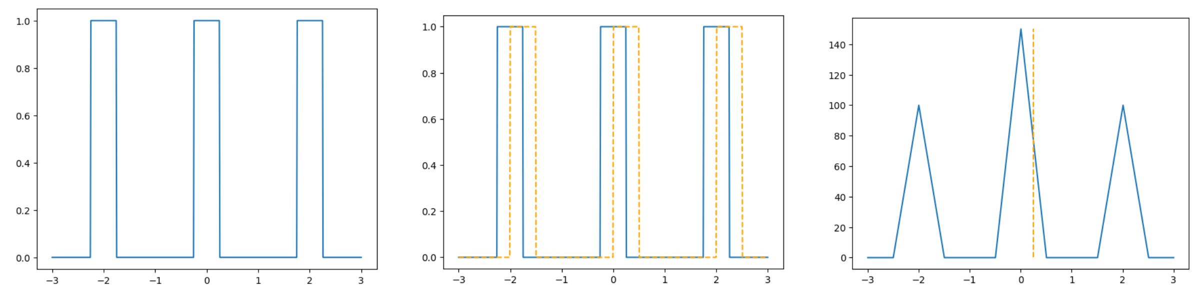
### 4.4.1 Theoretical setup

For more theoretical background on autocorrelation and its use in signal processing, see [86]. We will begin our discussion of local autocorrelation by first discussing *correlation*: the correlation between two signals (e.g., images) $f$ and $g$ at a particular offset is given by

$$(f \star g)(t) = \sum_x f(x)g(x-t).$$

That is, we offset one signal by some amount, multiply the resulting signals against one another, and sum across the domain. In the machine learning world, this is frequently called *convolution* and is the operation that convolutional neural networks are based upon—recall that much of Chapter 3 was based upon exploring and expanding convolution. However, in the domain of image analysis, correlation is most frequently used with two somewhat similar images, and correlation is used to align them—the shift value $t$ that maximizes the correlation is taken as the best alignment.

Next, *autocorrelation* is simply the correlation of an image with itself—that is, where $f = g$. This is used to detect periodicity—while autocorrelation always has a maximum at no shift (i.e., $t = 0$), if the signal has periodic elements, the autocorrelation will have local maxima at shifts corresponding to multiples of the period. See Figure 4.2 for an example of autocorrelation applied to a 1D function.



(a) (b) (c)

**Figure 4.2:** An example of autocorrelation on a 1D function. Subfigure (a) shows an example 1D function with periodic elements. In (b), we see the original function overlaid with a copy (the dashed orange line) that has been translated by $0.25$. Finally, (c) shows the resulting autocorrelation with a vertical dashed orange line showing the location in the autocorrelation corresponding to the translation shown in (b). As a note, the $x$-axis in (b) represents how much of a shift has been applied to the second function. The functions have been zero-padded outside their domain for the purposes of these multiplications, which results in the lower peaks in the autocorrelation function at $x = \pm 2$.

Finally, while autocorrelation is typically used on an entire image or scene, for our task we wished to detect periodic behavior in local regions within a larger scene; we therefore implemented *local autocorrelation*. To perform local autocorrelation, we subdivide a larger scene into many smaller sub-images, then compute the correlation between each small sub-image and the full-size original image with a limited range of shifts, zero-padding the smaller sub-image to ensure its compatibility with the large image. That is, we are focusing on a number of small patches and seeing if there is any periodicity within their local neighborhoods. This is closely related to the idea of *self-convolution* seen in [35].

**Table 4.1:** A glossary of notation used in the pseudocode algorithms in this chapter.

| Symbol | Meaning |
|---|---|
| $x \leftarrow y$ | Assign the variable $x$ the value $y$. |
| $\overline{x}$ | The mean of $x$. |
| $\sigma_x$ | The standard deviation of entries in $x$. |
| $\lfloor x \rfloor$ | The floor of $x$; return the greatest integer less than or equal to $x$ |
| $A_{x,y}$ | The value in the matrix $A$ in row $x$ and column $y$. |
| $A_{x_1:x_2,y_1:y_2}$ | The submatrix of $A$ containing rows $x_1$ through $x_2$ and columns $y_1$ through $y_2$ (including the left endpoints but not the right). |
| $\mathbb{M}_{x,y}$ | The space of $x \times y$ matrices. |
| $A^T$ | The transpose of the matrix $A$. |

### 4.4.2 Code implementation

To implement this in code, we used a relatively naive version of the algorithm that does all the multiplication directly, rather than utilizing faster techniques using the Fast Fourier Transform (FFT) [86]. Our aim was to test the efficacy of this method, so the simplicity of the direct method outweighed the computational efficiency of the Fourier methods. The local autocorrelation function which computed the autocorrelation of a small patch of a larger scene against the background is given as Algorithm 1. This function output a patch of approximately the same size as the input (it may be smaller if either the width or height of the input patch is odd) containing the autocorrelation values for shifts of up to half the width or height in either direction. Because the autocorrelation value was dramatically higher at no shift (i.e., at the translation vector $(0, 0)$), to ensure that the display scaling is accurate, we set the center of each resulting autocorrelation patch to a null value.

In Algorithm 2, we present the computational scheme for sequentially using Algorithm 1 on non-overlapping $50 \times 50$ patches across an entire DNB scene, with a $25$ pixel border around the outer edges of the scene so that the local autocorrelation for each patch can be computed without padding. As the output patches were the same size as the input patches, we could stitch them together to yield the local autocorrelation throughout the entire scene, as shown in Figure 4.3.

These algorithms were implemented in Python 3 principally using the `numpy` package, and can be found in the `local_autocorrelation.py` file in https://github.com/rgcda/SymFD.

**Algorithm 1** Local autocorrelation function for a patch of a larger image. For notation used here, see 4.1.

```
AUTOCORRELATE($A, x, y, w, h$)
$A$ is the entire scene as a matrix, $(x, y)$ are the coordinates of the upper left-hand corner of the local patch, and $(w, h)$ are its width and height
$P \leftarrow A_{x:x+w,y:y+h}$
$P \leftarrow \left(P - \overline{P}\right)/\sigma_P$
$n \leftarrow \sum_{i,j} P_{i,j}^2$
for $x_t = -\lfloor w/2 \rfloor \ldots \lfloor w/2 \rfloor$ do
    for $y_t = -\lfloor h/2 \rfloor \ldots \lfloor h/2 \rfloor$ do
        $x_r \leftarrow x + x_t$ and $y_r \leftarrow y + y_t$
        $C \leftarrow A_{x_r:x_r+w,y_r:y_r+h}$
        $C \leftarrow \left(C - \overline{C}\right)/\sigma_C$
        $R_{x_t,y_t} \leftarrow \left(\sum_{i,j} P_{i,j} \cdot C_{i,j}\right)/n$
    end for
end for
$R_{0,0} \leftarrow NaN$
return $R$
```

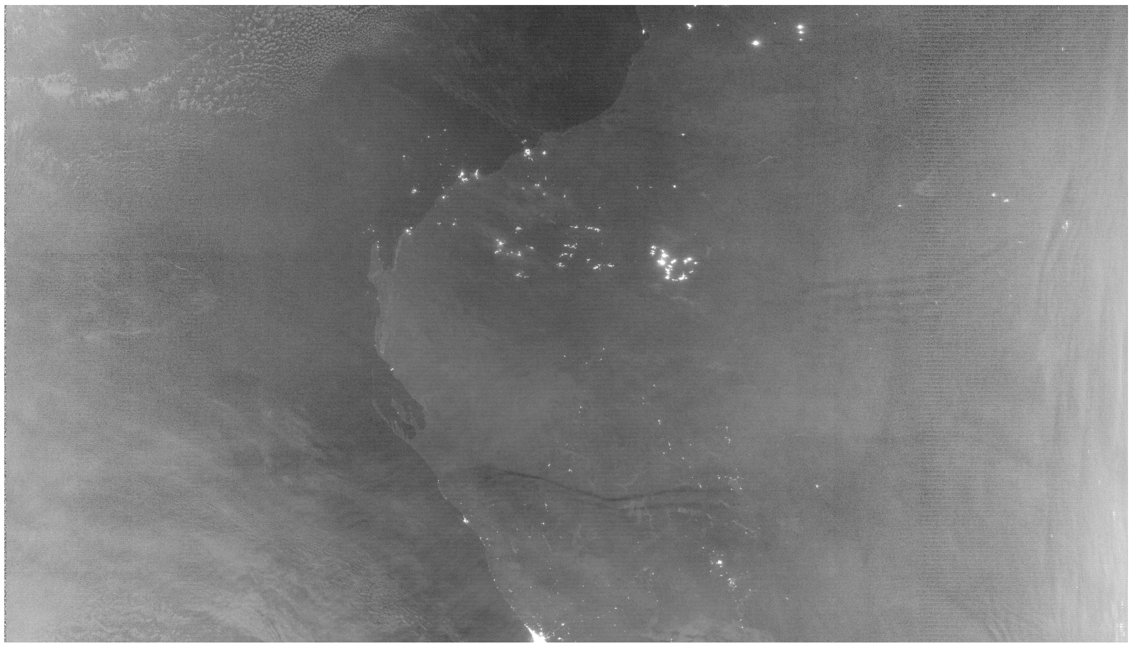
**Algorithm 2** Routine for computing local autocorrelation across an entire scene. For notation used here, see 4.1.

```
$x_n = \lfloor 4064/50 \rfloor - 1 = 80$ and $y_n = \lfloor 2304/50 \rfloor - 1 = 45$
for $x = 25, 25 + 50, \ldots 25 + x_n \cdot 50$ do
    for $y = 25, 25 + 50, \ldots 25 + y_n \cdot 50$ do
        $R_{x,y} \leftarrow$ AUTOCORRELATE($A, x, y, 50, 50$)
    end for
end for
return $R$
```

### 4.4.3 Results

As we can see in Figure 4.3, the local autocorrelation had the ability to detect the periodicity of gravity waves in some DNB scenes. This was a promising first result, as it indicated that at least one of the aspects we identified as being distinctive of gravity waves can be enhanced by an appropriate mathematical transformation.

However, local autocorrelation also responded very strongly to other features in the scene, particularly bright city light regions. This is due to a flat, bright region giving a strong response when multiplied against a periodic signal or, particularly, itself. Correlation detects how well two



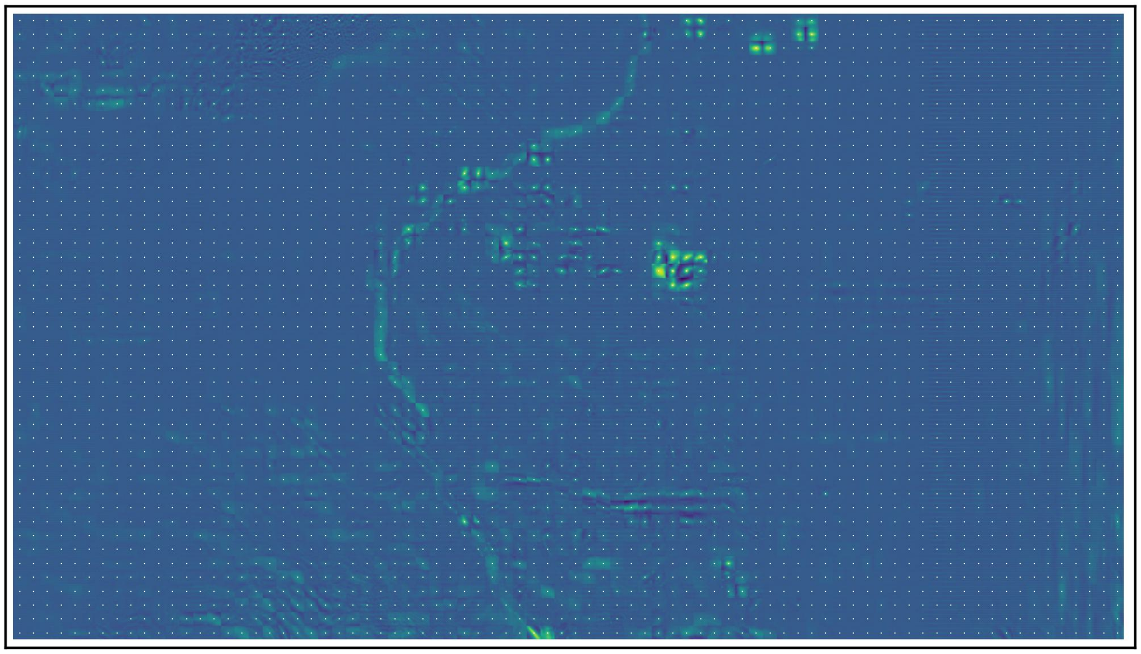


**Figure 4.3:** The same DNB scene shown in Figure 4.1, along with its local autocorrelation. Notice that local autocorrelation detects the gravity waves at the bottom center, but does not respond to the fainter example in the center right. Moreover, the strongest response is to the city lights in the center of the image.

images are aligned, and these flat, bright regions aligned very well with themselves, even when offset slightly. While thresholding the brightest parts of these regions yielded some improvement in this regard, the issue persisted even in less bright regions.

The key issue in this case is that gravity waves have a faint signal when compared to other features in the scene, particularly clouds. Local autocorrelation is sensitive to the amplitude of periodicity—because we were multiplying the two signals together, given two features with the same amount of periodicity, the one with the larger amplitude was showing up more clearly in the autocorrelation. This means that even features that are not as periodic as gravity waves but which have greater dynamic range had equivalent or larger signatures in the output autocorrelation. While we did normalize the patches used in the computation of local autocorrelation, the gravity waves were of low amplitude even relative to the dynamic range within a typical patch, so normalization wasn't able to resolve this issue.

Autocorrelation also did not particularly utilize the linearity of the gravity waves, so we are not leveraging this second attribute fully. Thus, we chose to explore more complex methods as well.

## 4.5 Wavelet-based Ridge Detection

Wavelets are most frequently used in multiresolution analysis and synthesis, in which an image is broken down at multiple orientations, scales, and positions, analyzed or filtered in some fashion, and then a modified image can be reconstructed from the resulting weights. For a theoretical introduction to this method of analysis, see [32]. However, wavelets can also be used on just the analysis side to construct edge and ridge detection algorithms. Inspired by the cosine- and sine-based edge detection algorithm of Kovesi [44, 45], Reisenhofer and King developed and refined a wavelet-based analysis technique that can detect edges, ridges, and blobs [41, 64–66]. We utilized the fine-tuned ridge detection algorithm as presented in [66], and did computations principally using the accompanying SymFD MATLAB toolbox.

### 4.5.1 Theoretical setup

While full details can be found in [66], we give a high-level introduction to the methodology here.

Like autocorrelation, wavelet analysis is based on multiplying two signals against one another. However, in wavelet analysis, the second signal is not another copy of the image being analyzed, but is a particular *wavelet function*, which has been translated, scaled, and/or rotated. By computing these *wavelet coefficients* across a number of different positions, orientations, and scales, we can obtain a rich representation of the original image called a *multiresolution analysis*.

There are a large number of possible wavelet functions with which to perform this analysis, from the simple Haar wavelet to more complex Morlet and Daubechies wavelets. In [66], the primary wavelet functions used are derivatives of the Gaussian function and their Hilbert transforms. These wavelet functions have the desirable property that they are either even or odd symmetric, and can be arranged in pairs such that each pair contains one even-symmetric and one odd-symmetric function which are largely equivalent in other ways. When computing the wavelet transform using each element in a given pair of wavelet functions, then, we can deduce that differences in the wavelet coefficients arise principally from how a feature responds differently to odd vs. even symmetry.

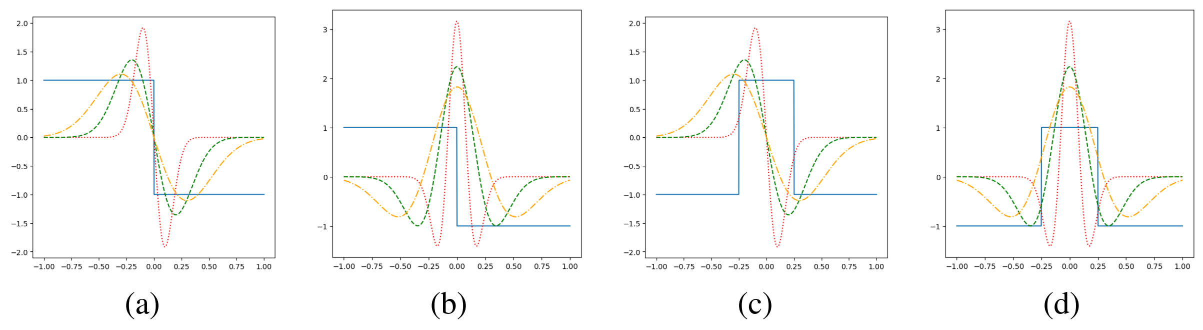


**Figure 4.4:** Edge and ridge detection using Gaussian wavelets for ideal 1D edges and ridges. In (a) and (b), we see an ideal 1D edge along with odd and even-symmetric wavelet functions (respectively) at three different scales. In (c) and (d), we see an ideal 1D ridge with the same wavelet functions.

Let us first consider the theoretically simpler discussion of detecting an idealized 1D edge using odd and even symmetric wavelet functions, as shown in Figure 4.4 (a) and (b). We note that the odd-symmetric wavelet will have a strong response when aligned precisely on the edge across all scales, while the even-symmetric wavelet will integrate to zero, also at every scale. This observation is the key to the edge-detection measure produced in [66].

On the other hand, we are interested more in ridges than edges, as we can think of each wave in a gravity wave as a distinct ridge. An idealized 1D ridge is shown in Figure 4.4 (c) and (d). We can observe that in this case, the responses of the odd and even symmetric wavelets have mostly switched, with the odd-symmetric wavelet giving a response of zero at all scales when centered on the ridge. However, while the even-symmetric wavelet gives a non-zero response across all scales, the magnitude of that response is maximized when the width of the wavelet is scaled to be near that of the ridge. Thus, we can detect both the presence and width of a ridge, or (alternatively) search for ridges within specific width ranges.

To generalize this idea to two dimensions, the wavelet functions need to be modified, as the above wavelets are themselves one-dimensional. To obtain appropriate two-dimensional wavelets, we take our odd- and even-symmetric 1D wavelets and tensor each of them with a Gaussian function to obtain a two-dimensional function that has odd or even symmetry along one direction, and falls off as a Gaussian in the other. These two-dimensional wavelets are then translated, scaled, and rotated throughout the input image—the notable addition here is rotation, which is necessary to detect edges and ridges at all orientations. The scaling done here is also not necessarily the same in all directions (or *isotropic*). *Anisotropic* scaling, or scaling which applies unevenly to each direction, can help detect features like edges and ridges that are more prominent in one direction, particularly in the presence of noise, but can introduce issues in situations where the ridge or edge is not smooth. The degree of isotropy is one of the hyperparameters that we tuned when applying these algorithms. For further details on how this method generalizes to two dimensions, we refer the interested reader to [66].

### 4.5.2 Code implementation

For the purposes of testing whether this algorithm would be successful at identifying gravity wave ridges in DNB imagery, we utilized the SymFD toolbox for MATLAB produced by Reisenhofer to accompany [66] and found at https://github.com/rgcda/SymFD. This toolbox provides both MATLAB functions usable in scripts and a GUI. We began by using the GUI on several examples to find a parameter set which seemed to do a reasonable job of identifying gravity waves, then used a script to automate ridge detection across all our samples.

The parameters we used are given in Table 4.2.

**Table 4.2:** Parameters used in SymFD for computing ridge detection.

| Parameter | Value |
|---|---|
| maxFeatureWidth | 20 |
| maxFeatureLength | 20 |
| minFeatureWidth | 10 |
| minContrast | 4 |
| alpha | 0.2 |
| nOrientations | 2 |
| scalesPerOctave | 2 |
| evenOddScaleOffset | 1 |
| generator | SFDMexicanHatVsGauss |
| orientationOperator | rot |
| thinningThreshold | 0.1 |
| minComponentLength | 5 |
| maxFeatureHeight | 0 |

### 4.5.3 Results

In general, this wavelet-based ridge detection algorithm did detect gravity wave ridges. In Figure 4.5, we see that the ridge detection algorithm principally highlighted the two regions where gravity waves are present, while doing a significantly better job than autocorrelation at ignoring non-gravity wave features.

However, on scenes that are not as simple it also detected many other ridges throughout the image, to the point where it was challenging to identify the gravity waves in the scene, as in Figure

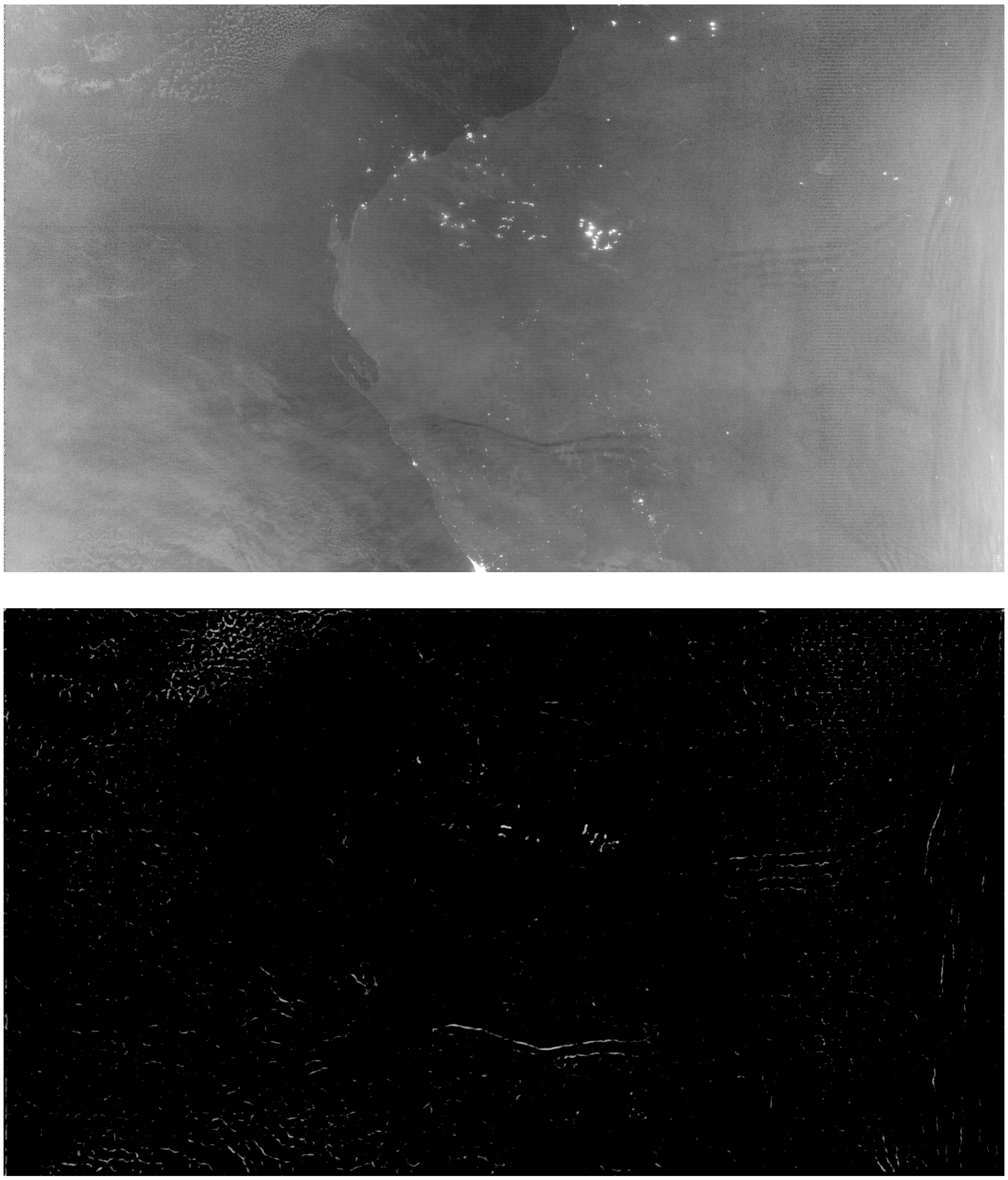


**Figure 4.5:** The same DNB scene shown in Figure 1 along with the output of the ridge detection algorithm. In the bottom image, the brighter regions correspond to more "ridge-like" regions.

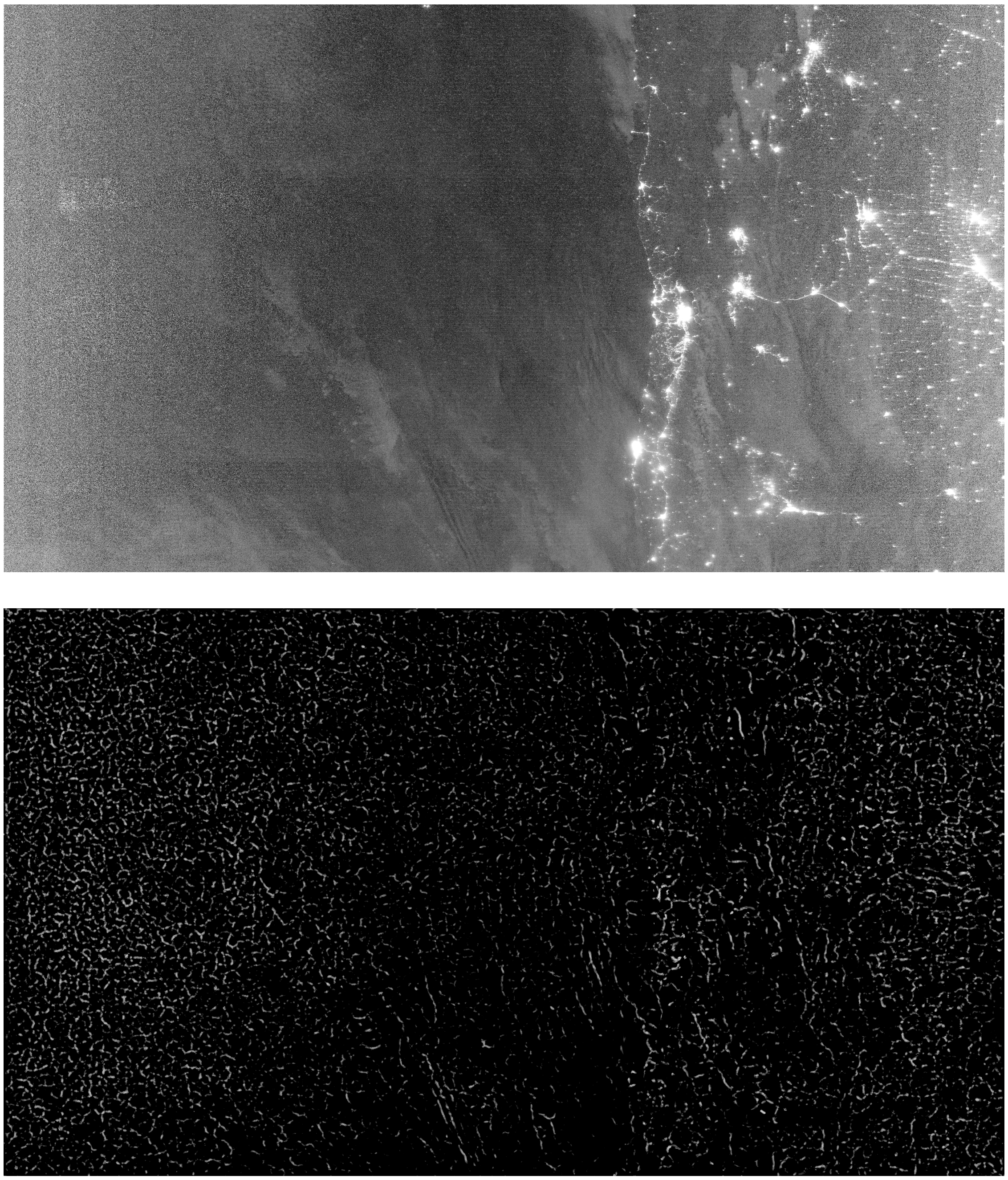


**Figure 4.6:** Another DNB scene containing gravity waves (seen in the bottom center) along with the corresponding ridge detection output, run using the same parameters as in Figure 4.5

.

4.6. Part of this is down to the choice of parameters—we have not necessarily found the optimal set of parameters defining the width and length of ridge to be searched for, as well as which wavelet functions to use. Moreover, it appears that this optimal choice may not be the same for all scenes, making the best overall choice even trickier.

More fundamentally, this ridge detection algorithm did not fully leverage either of the defining characteristics of gravity waves outlined in Section 4.2. It did not utilize periodicity at all, as each ridge is detected entirely separately. While it did make some use of linearity, in that the waves are more-or-less continuous lines, the ridge detection algorithm did not place any restrictions on how much these ridges can bend. Thus, many of the false positives in the ridge detection output are very "bendy" ridges that could be removed if we were able to make more full use of the linearity property of gravity waves.

## 4.6 Finite Radon Transform

The Radon transform in continuous space is a theoretical construct that underlies a number of practical tools using tomography, such as CT (computed tomography) scans. It is based on the idea that a 2D structure can be perfectly reconstructed from all possible straight line integrals passing through said structure [43]. In practice, this reconstruction is approximated by finitely many line integrals. This idea has been extended to finite images in a number of ways, but we will here explore the Finite Radon Transform (FRT), which utilize prime arithmetic and the geometry of finite fields to achieve perfect reconstruction [21]. Our goal was that by analyzing line integrals through the image, we hoped to focus on the linearity of gravity waves, then use additional techniques such as the Fourier transform to detect periodicity amongst line integrals along parallel lines.

### 4.6.1 Theoretical setup

We utilized the FRT as introduced by Do and Vetterli in [21]. The core idea of the Radon transform is that the collection of integrals along every straight line passing through a structure contains enough information to perfectly reconstruct that structure. When analyzing a finite image, the meaning of a line changes somewhat, as we no longer are dealing with continuous slopes. However, there is one finite context in which lines have been studied extensively: finite geometries. In particular, when our image is of size $p \times p$, where $p$ is a prime number, we can view the underlying space as an instance of $\mathbb{F}_p \times \mathbb{F}_p$, where $\mathbb{F}_p$ is the finite field on $p$ elements. In this space, there are precisely $p + 1$ distinct slopes, and each slope yields $p$ distinct parallel lines, so there are a total of $p(p+1)$ lines, each of which contains precisely $p$ points. These lines are the solution sets to equations of the form

$$ax + by - t = 0$$

where all addition and multiplication is done modulo $p$ and the line is defined by the normal vector $(a, b)$ and intercept $t$. As we can see in Figure 4.7, these lines include a degree of periodicity induced by the mod $p$ arithmetic, the implications of which we will discuss later.

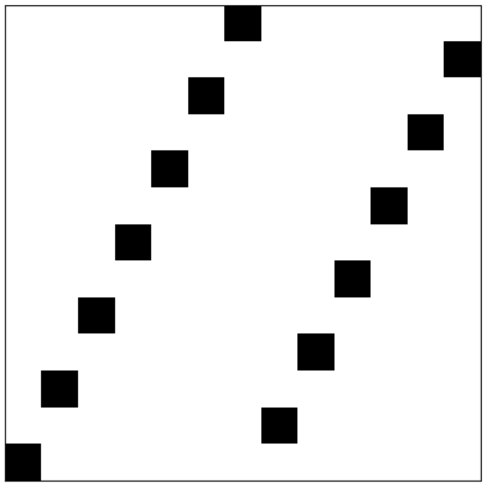


**Figure 4.7:** An example finite line in a $13 \times 13$ patch. This is the line with normal vector $(-2, 1)$ and translation vector 0. Note that this single line wraps around the patch twice.

We can then define the FRT of a zero-centered (i.e., with a mean of 0) function $f$ on a $p \times p$ discrete image as a function of $(a, b)$ and $t$:

$$r_{(a,b)}(t) = \frac{1}{\sqrt{p}} \sum_{(i,j) \in L_{(a,b),t}} f(i, j),$$

where $L_{(a,b),t}$ is the set of points on the line defined by normal vector $(a, b)$ and intercept $t$. The scaling factor $\frac{1}{\sqrt{p}}$ ensures that the $\ell_2$ norm is preserved by the FRT, and the zero-centering of $f$ yields better numerical accuracy and reconstructability properties. The collection of all these $r_{(a,b)}(t)$ can be structured as a $(p + 1) \times p$ image where the $x$ axis indexes all possible lines $(a, b)$ and the $y$ axis is the intercept $t$; such an image is called a *sinogram*, following the convention of the Radon transform in the continuous setting.

As we discussed above, there are a total of $p(p + 1)$ distinct lines, but there are a total of $p^3$ choices of $a, b$, and $t$, so there must be some duplication in parameters. This duplication comes from the fact that (as in Euclidean geometry) every non-zero multiple of the normal vector $(a, b)$ yields the same line. However, unlike in Euclidean geometry, the choice of which multiple of the normal vector does have an impact on the FRT.

In particular, we order the elements of the FRT by the intercept value $t$, and this ordering is affected by which multiple of the normal vector $(a, b)$ we choose. To see this, we transform the line equation into the more intuitive slope-intercept form:

$$ax + by - t = 0 \implies y = tb^{-1} - ab^{-1}x,$$

in which it is useful to recall that in finite fields, the multiplicative inverse of an element $b^{-1}$ is the unique other integer $c$ in $\mathbb{F}_p$ such that $bc = 1$. If we scale $(a, b)$ by $k$, we obtain the modified equation

$$y = t(kb)^{-1} - ab^{-1}x,$$

as the $k$ cancels out of $ab^{-1}$. Thus, when we scale $(a, b)$ by $k$, we are changing the interval in which we step through the intercepts by $k^{-1}$, and thus affecting the ordering of elements in the FRT.

As explored in [21], there is an optimal choice of multiple for each normal vector, where by "optimal", we mean minimizing the "wrap-around" effect due to the mod $p$ arithmetic. To have a common place to start from, we begin with the naive set of normal vectors,

$$u_k = (-k, 1) \text{ for } k = 0, \dots, p-1 \text{ and } u_p = (1, 0).$$

The optimal choice for each $k$ can then be computed as

$$(a_k^*, b_k^*) = \arg \min_{\substack{(a_k, b_k) = n u_k : 1 \le n \le p-1 \\ C_p(b_k) \ge 0}} \left\| (C_p(a_k), C_p(b_k)) \right\|,$$

where $C_p(x)$ is the centralized function of period $p$,

$$C_p(x) = \begin{cases} x & \text{if } x/p < 0.5 \\ x - p & \text{if } x/p > 0.5 \end{cases}$$

for $x \in 0 \dots p-1$. In the minimization procedure, we restrict to non-negative $C_p(b_k)$ to avoid any ambiguity in the minimum, as $\|C_p(a_k), C_p(b_k)\| = \| - C_p(a_k), -C_p(b_k)\|$. It is worth noting that these optimal normal vectors depend only on the prime size of the patch $p$, not on the data $f$. Thus, if the FRT is to be computed repeatedly for patches of the same size, this minimization need only be done once.

Putting this all together then, our procedure for computing the FRT using optimal normal vectors consists of the following steps: compute the collection of optimal normal vectors $(a_k^*, b_k^*)$, then compute the FRT for each patch $f$ as

$$r_k(t) = \frac{1}{\sqrt{p}} \sum_{(i,j) \in L_{k,t}} f(i, j),$$

where $L_{k,t}$ is the set of points on the line $a_k x + b_k y - t = 0$.

One of the desirable properties of the FRT is that it yields perfect reconstructability: given the complete collection of $p(p+1)$ lines, the original $p \times p$ image can be perfectly recovered. This is done via *finite back-projection*:

$$f(i,j) = \frac{1}{\sqrt{p}} \sum_{(k,t)\in P_{i,j}} r_k(t),$$

where $P_{i,j}$ is the set of parameters $(k,t)$ such that the lines $L_{k,t}$ that go through the point $(i,j)$. That is,

$$P_{i,j} = \{(k,t) : a_k i + b_k j - t = 0\}.$$

In practice, we can recover $P_{i,j}$ more efficiently from the way we store the output $r_k(t)$.

### 4.6.2 Code implementation

As this technique has not been extensively utilized in the past decade, little modern code existed, so we had to build our own implementations of these principles in code. The following algorithms were all implemented in Python 3 using `numpy`.

First, using Algorithm 3, we computed the optimal normal vectors given a patch of prime size $p \times p$.

**Algorithm 3** A function to compute optimal normal vectors for the FRT. For notation used here, see 4.1.

```
COMPUTE_NVECS($p$)
$N_0 \leftarrow [0,1]$
for $k = 1,\ldots,p-1$ do
    $u_k \leftarrow [-k,1]$
    for $n = 1,\ldots,p-1$ do
        $T_n \leftarrow n \cdot u_k \mod p$
        if $(T_n)_1 < 0$ then
            continue to next $n$
        end if
        $d_n \leftarrow \|T_n\|$
    end for
    $n_k^* \leftarrow \arg\min_n d_n$
    $N_k \leftarrow n_k^* \cdot u_k \mod p$
end for
$N_p \leftarrow [1,0]$
return $N$
```

We stored these normal vectors and could re-use them for as many patches of the same size as we desire. This is useful, as in searching for gravity waves, we would be applying the FRT to many smaller patches of the same size within a larger image, or even multiple images.

Next, we utilized this list of optimal normal vectors to compute the FRT for an example patch using Algorithm 4. This algorithm made use of the `Roll(v, i)` function, which takes a vector $v$ and cyclically permutes it by $i$. That is,

```
Roll([1, 2, ...,  10], 2) = [9, 10, 1, ..., 7, 8].
```

**Algorithm 4** An algorithm for computing the FRT given a list of optimal normal vectors $N$ and a $p \times p$ input patch $P$. For notation used here, see 4.1.

```
$P \leftarrow P - \overline{P}$
$R \in \mathbb{M}_{p,p+1}$
for $i = 0, \ldots, p$ do
    if $(N_i)_1 = 0$ then
        $R_{:,i} = \sum_{j=0,\ldots,p-1} P^T_{j,:}$
    else
        $\Delta x = (N_i)_1$
        $\Delta y = -(N_i)_0$
        for $k = 0, \ldots, p-1$ do
            $R_{:,i} \leftarrow R_{:,i} + \textsc{Roll}(P_{:,k\cdot\Delta x \mod p}, k \cdot \Delta y \mod p)$
        end for
    end if
end for
return $\frac{1}{\sqrt{p}} R$
```

Finally, although we did not use it in the analysis step, we also coded the finite back-projection (FBP) reconstruction operation, making use of the same normal vectors $N$, and taking as input the sinogram $R$.

The full Python code for all of these algorithms can be found in the `FRT` class in the `frat.py` file in the GitHub repository at https://github.com/zyjux/harmonic_gw.

**Algorithm 5** The Finite Back Projection (FBP) reconstruction algorithm, given a list of optimal normal vectors $N$ and a corresponding sinogram $R$. For notation used here, see 4.1.

```
$\hat{P} \in \mathbb{M}_{p \times p}$
for $i = 0, \ldots, p$ do
    $\Delta x = (N_i)_1$
    $\Delta y = -(N_i)_0$
    for $k = 0, \ldots, p-1$ do
        if $(N_k)_1 = 0$ then
            $\hat{P}_{k,:} \leftarrow \hat{P}_{k,:} + R_{:,i}$
        else
            $\hat{P}_{:,k \cdot \Delta x \mod p} \leftarrow \hat{P}_{:,k \cdot \Delta x \mod p} + \textsc{Roll}(R_{:,i}, -k \cdot \Delta y \mod p)$
        end if
    end for
end for
return $\frac{1}{\sqrt{p}} \hat{P}$
```

### 4.6.3 Results

We first tested this methodology on a synthetic dataset we generated. We created a $149 \times 149$ patch (note that $149$ is a prime number) containing a periodic function shown in Figure 4.8. This function was a cosine curve crossed with the unit interval, then rotated by $\pi/3$; we varied the period of the cosine curve to test the response of the FRT to periodic functions with the same orientation but different frequencies.

Applying the FRT to the example from Figure 4.8 (a), we obtained a sinogram shown in Figure 4.9 (a). We can see that most of the slopes did not yield any significant response or periodicity, with the exception of columns $73$ and $147$, which correspond to normal vectors $(3, 2)$ and $(2, 1)$ respectively. Examining a representative line with normal vector $(3, 2)$, shown in Figure 4.9 (b), we see that the slope of the line was generally aligned with the peaks of the periodic function and that the periodic behavior of the line induced by the mod $p$ arithmetic also matched up with the period of the underlying function.

When we applied the FRT to the example from Figure 4.8 (b), we obtained the sinogram shown in Figure 4.9 (c). In this case, we only had one slope that yielded a strong response, column $48$,

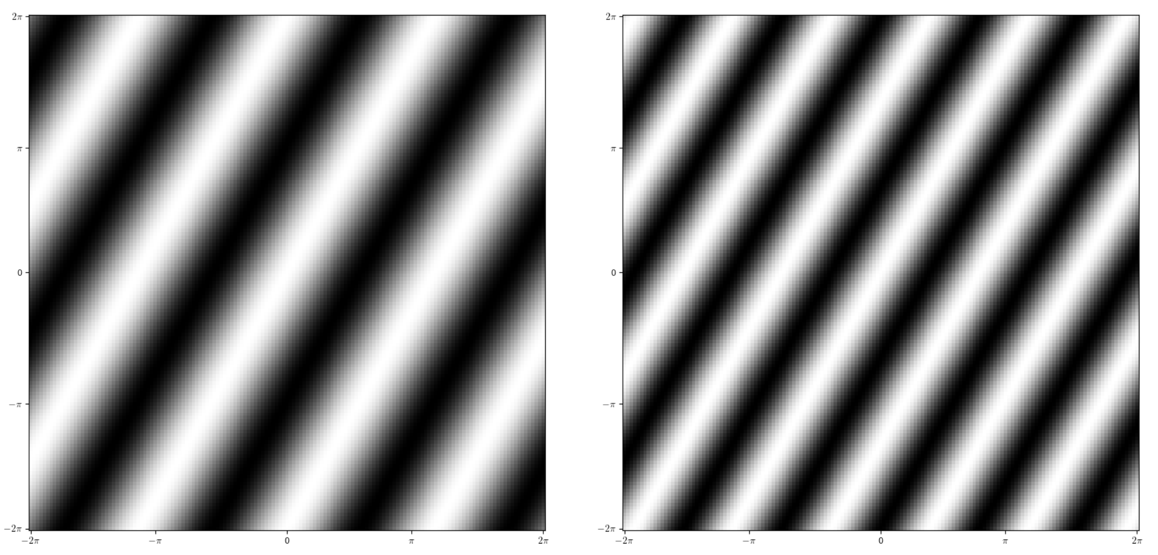
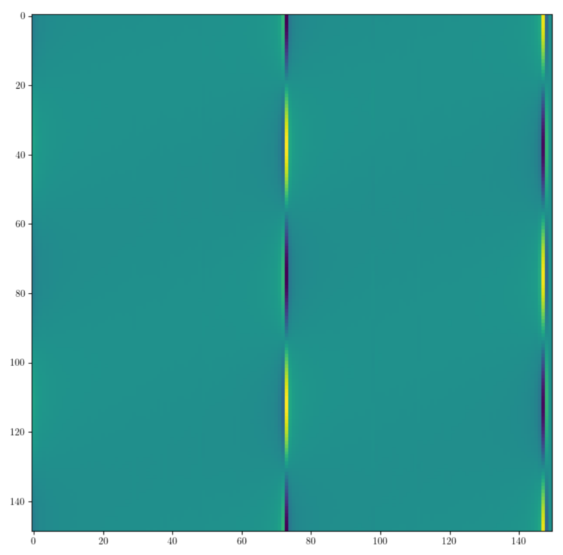

(a) (b)

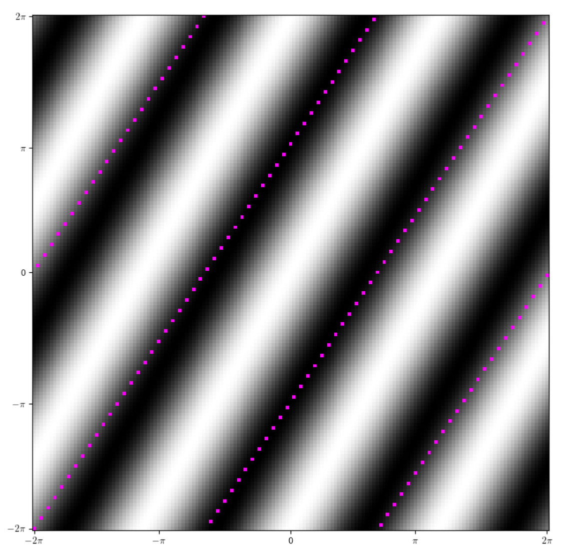
**Figure 4.8:** The two principal synthetic examples we will explore in this section. In (a), we see the example with period $\pi$, and in (b) we see the example with period $2\pi/3$.

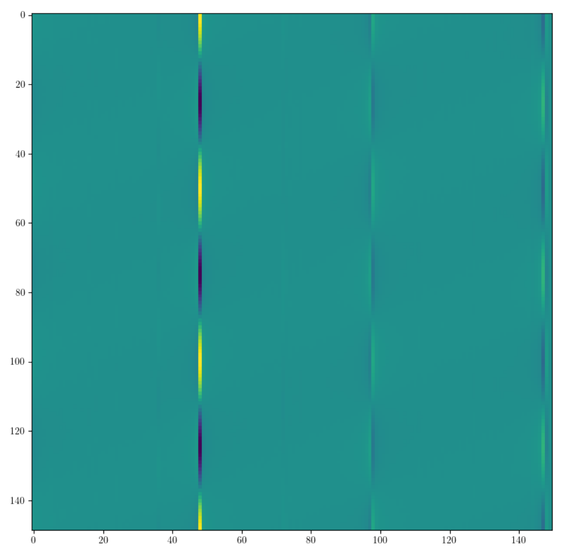
which had normal vector $(5, 3)$. Looking at a representative line with this normal vector, shown in Figure 4.9 (d), we see that again the slope matched well, and the period of the line also matched.

In both instances, the FRT was able to very clearly detect the orientation and period of the input image, and for synthetic data such as this, a technique as simple as total variation within each column would be able to automatically detect the presence of these strong responses.

However, taken together, these examples also show that in order for the FRT to give a strong response for a particular normal vector, both the orientation and period of the resulting lines had to match the image being analyzed. This conflation of period and orientation in the same step is not necessarily desirable, as we had to vary these two parameters in a strongly correlated way, not independently, as we can see in Figure 4.10. Thus, in applications it is likely that we will find neither the orientation nor period that matches the input signal perfectly, but instead some relatively close approximation of both that happen to occur together. In Figure 4.9, the orientation of the underlying periodic function in the input image was constant, only the period was changed, but the normal vectors that responded to each example were completely different, indicating that in fact matching the period was at least as important as matching the orientation.


(a)


(b)


(c)

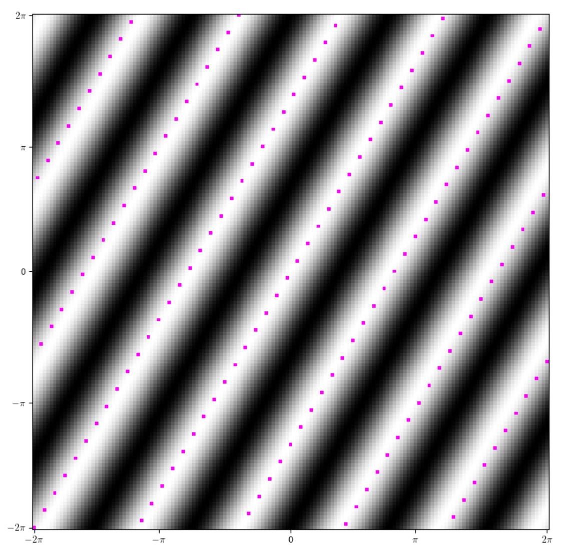

(d)

**Figure 4.9:** The sinograms and example lines for the synthetic examples. In (a), the sinogram of the synthetic image with period $\pi$ is shown, and in (b), an example line with normal vector $(3, 2)$ (corresponding to the 73rd column in the sinogram, which has a strong response) is shown overlaid on the input image. In (c), the sinogram of the synthetic image with period $2\pi/3$ is shown, and in (d), an example line with normal vector $(5, 3)$ (corresponding to the 48th column in the sinogram, which has a strong response) is shown overlaid on the input image.

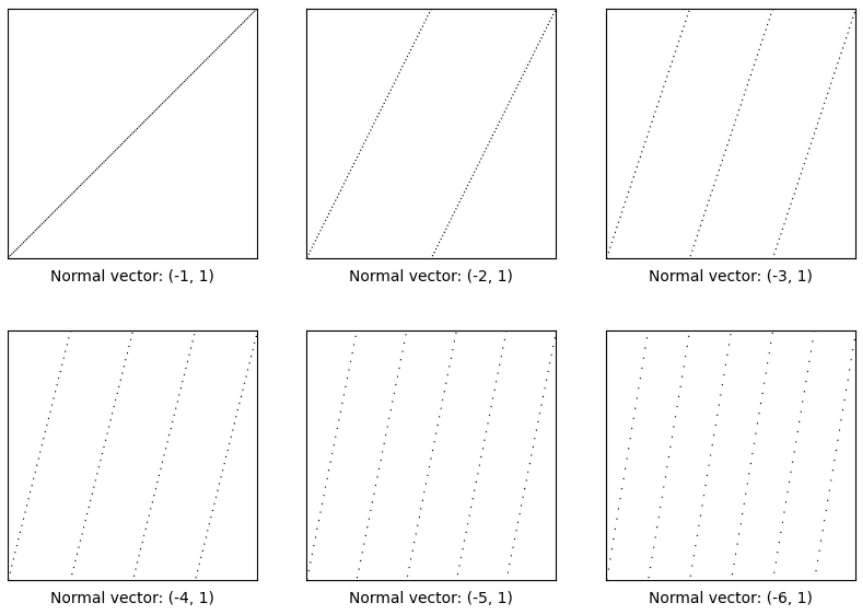


**Figure 4.10:** Example finite lines for consecutive normal vectors, showing the correlation between orientation and period.

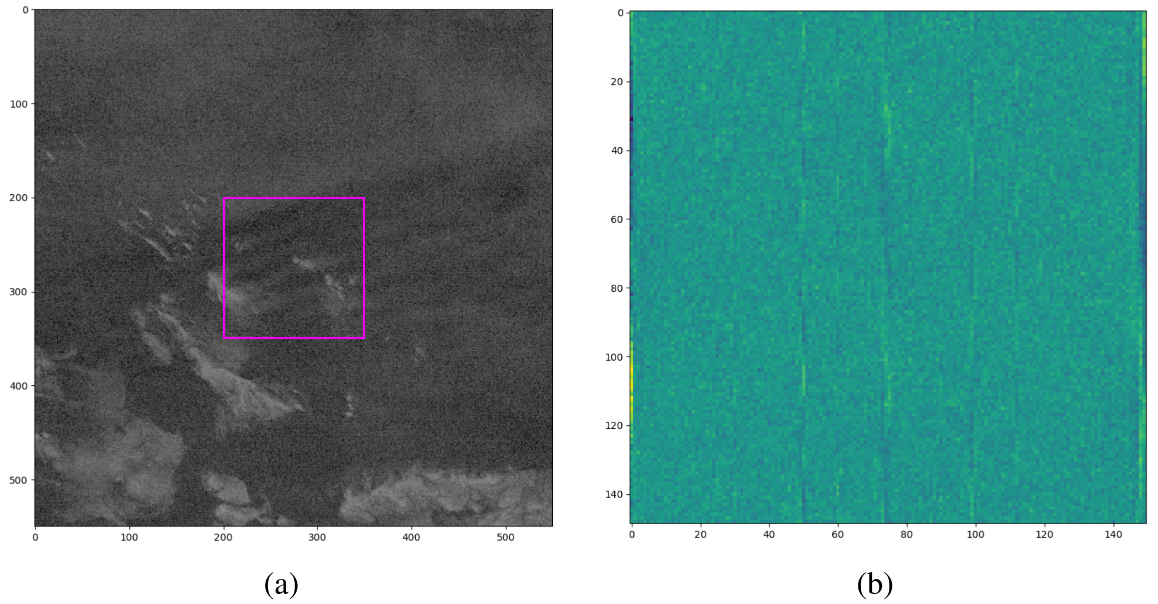
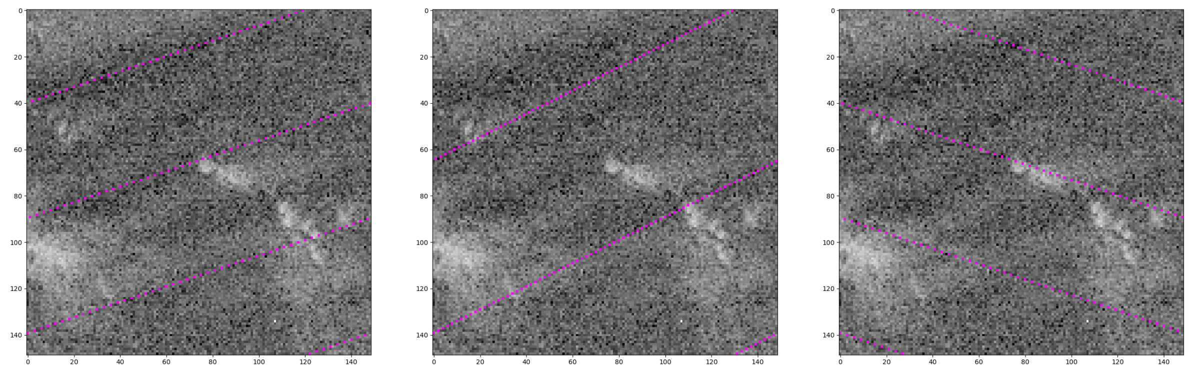


**Figure 4.11:** An example of gravity waves in the DNB, shown in (a), and the FRT sinogram of the $149 \times 149$ region highlighted in magenta in the center of (a).

When we tested this on real data, we found similar results, although as always with real data, the results are not quite as clear-cut. In Figure 4.11, we can see in (a) an example gravity waves region and in (b) the resulting sinogram from applying the FRT. In (b), we note that the largest variation within a column occurred in column 0, which corresponds to simple row sums; however, this variation was likely due to the presence of brighter clouds in the lower part of the example image. Aside from this column, we note that there are 3 potential columns that on visual inspection displayed some form of periodic behavior: column 50 (corresponding to normal vector $(-1, 3)$), column 75 (corresponding to normal vector $(-1, 2)$), and column 99 (corresponding to normal vector $(1, 3)$). Example lines with each of these normal vectors are displayed in Figure 4.12.



(a) (b) (c)

**Figure 4.12:** Examples of lines corresponding to the three distinctive columns in Figure 4.11 (b). In (a), there is a line with normal vector $(-1, 3)$ from column 50 of Figure 4.11 (b), in (b) there is a line with normal vector $(-1, 2)$ from column 75, and in (c), there is a line with normal vector $(1, 3)$, from column 99.

While the lines shown in Figure 4.12 (a) and (b) appear to correspond to the true direction and period of the gravity waves, the line in Figure 4.12 (c) appears to be in a direction unrelated to the gravity waves. The choice of these three columns was subjective and visual, so utilizing a more rigorous technique for detecting periodicity within each column would likely yield better results.

While this algorithm did leverage both the linearity and periodicity properties of gravity waves, it entangled them in complex ways within a single step, and thus made it difficult to analyze them separately and ensure we were obtaining the clearest signal from each. This direction of utilizing finite versions of the Radon transform, however, appears to be a fruitful direction to pursue. There are other ways to translate the continuous Radon transform to finite space that do not rely so heavily on finite geometries and prime arithmetic, and which would thus not impose constraints on the relationship between orientation and period.

## 4.7 Conclusions and Future Work

Overall, all of the three methods we tested—local autocorrelation, wavelet-based ridge detection, and the FRT—showed some degree of promise at enhancing gravity waves in DNB imagery, but none of them were as effective as desired, in part because they are not fully utilizing the characteristic properties of gravity waves identified in Section 4.2. However, the results here do indicate several promising directions to attempt next.

Beyond specific new techniques, we wish to explore making better use of the data available to us. The DNB contains a high degree of noise, at least some of which is a highly regular "striping" pattern due to unavoidable issues with the DNB calibration procedure. We hope to use destriping and denoising procedures, such as the one in [56] to reduce the impact of these artifacts and increase the reliability of our methods. There are also other bands and products available from VIIRS, including infrared channels and a cloud mask product, which could help disambiguate gravity waves from wave-like structures appearing in clouds. While our current methods only made use of the single DNB channel, multi-channel approaches are helpful for humans to reliably identify gravity waves, so it is reasonable to expect that they could be integrated into algorithmic solutions to increase their performance as well.

As far as new methods go, there are two principal techniques we hope to implement in the near future.

First, we intend to combine local autocorrelation and wavelet-based ridge detection. One of the main issues with local autocorrelation was that broader patches like clouds and city lights had strong autocorrelative properties without being the form of periodicity we were hoping to detect, particularly when the signal we are interested in has a relatively shallow dynamic range. On the other hand, the ridge detection algorithm did a good job of isolating relatively narrow, ridge-like features even when their dynamic range is quite shallow, but did not utilize either the linearity or periodicity characteristics of gravity waves. Thus, by first applying the ridge detection algorithm and then performing local autocorrelation on that output, we hope to combine the strengths of both algorithms. As part of this process, we also intend to look at modifying the ridge detection algorithm slightly to take better advantage of the linearity of gravity waves by restricting how much a ridge can "bend".

Second, we will look at an alternate generalization of the Radon transform to finite patches called the *Mojette transform* [33]. Where the FRT relies heavily on the theory of finite geometry and prime arithmetic to enumerate all possible lines in the patch and ensure perfect reconstructability in all cases, the Mojette transform is in many ways more general. It does not require that patches be of prime by prime size, and uses lines that are not periodic and thus do not all contain the same numbers of pixels. However, the number of pixels on each line is predictable based on the image size, line orientation, and intercept, so this variance in number of pixels is relatively easy to normalize for. The orientations and number of lines is also now a hyperparameter that can be chosen based on the needs of the problem. While the Mojette transform can in some situations have "ghosts" (patterns in the image which yield a net 0 contribution to the Mojette transform, and are thus non-reconstructable), this is not a limitation in analysis-only situations such as the detection of gravity waves. However, the lack of periodicity in the lines used for the Mojette transform means that we can separate the detection of linearity and of periodicity, by first using the Mojette transform to focus our attention along lines passing through the space, then applying the Fourier transform or autocorrelation to detect periodicity in each direction.

We hope that at least one of these two variations will yield the hoped-for enhancement of gravity waves in DNB imagery. Our goal is to build a fully automated gravity wave detector that can efficiently scan through the large amount of DNB data available and find more examples of gravity waves. While we (of course) hope that we can build a well-calibrated and accurate detector, we would also be satisfied with a detector that is successful at detecting gravity waves, but which also produced a relatively high rate of false positives; such a detector would make the task of creating a high-quality hand-labeled dataset easier by several orders of magnitude. Once such a dataset exists, more recent techniques such as convolutional neural networks can be brought to bear on the problem. Such methods also make it easy to integrate multi-channel data, which as we mentioned above, is frequently useful for human identification of gravity waves.

Finally, it is worth reflecting on the overall methodology by which we conducted this search. Rather than simply trying a smattering of different techniques, we attempted to organize our search by identifying characteristic properties of gravity waves in DNB imagery then looking for transformations and algorithms that respond well to those types of properties. This structure also gave us a way to analyze the results beyond simple numerical scores. Particularly at early stages of a project like this, the qualitative behavior of algorithms is at least as important as any numerical measure of their success, and focusing on trying to enhance particular properties allowed us to assess these behaviors on a principled way. It also made it easier to give intuitive explanations of instances where the algorithms struggled, and guided us from our initial techniques to further variations that we wish to try in the near future.

# Chapter 5

# Conclusions

In these chapters, we have explored different approaches in how we can use domain knowledge to inform the types of mathematical algorithms we use to analyze data. While mathematical analyses and input are most commonly used on the theoretical development side of science, we hope that these examples show that there is value in bringing this perspective at the application stage as well.

In Chapter 2, we were motivated by observations from domain scientists that in classifying clouds, one of the most important attributes is the *texture* of the cloud. While it is possible that machine learning algorithms will pick up on this texture, it is not guaranteed, as we have little control over what features those methods choose to prioritize. Rather than leaving this to chance, we built an algorithm based on persistent homology that we knew from theoretical results would summarize and emphasize the texture present in an image. Once we had this algorithm running, we used the our knowledge of how persistent homology works to find novel ways of attributing our results back to image-level features. This product would not have been possible without a collaborative environment involving both those with theoretical knowledge of how these algorithms work and those with practical knowledge of what types of features are important to analyze.

In Chapter 3, we were again motivated by a domain expert observation about the data: storm systems are influenced by prevailing winds and geography. The question that arose, then, was "can we construct a model that will generalize better to other situations and locations by enforcing rotational symmetry?" To implement this algorithm required not just coding, but use of abstract algebra, representation theory, and geometry. In this case, the elegant theory underlying this new model did not translate to increased performance, as the accuracy achievable by turning a large model loose on a large collection of data is tough to beat. Still, we now know more about the types of questions these new geometric deep learning algorithms are suitable for, and can continue to search for applications and questions for which they are the best answer.

Finally, in Chapter 4, we were in a situation where modern neural network methods had proven very tricky to use, as there was only a very small dataset of examples. In this work, even more than in the previous chapters, we explicitly used this joint practical/theoretical framework to develop our methods: we started by analyzing the image samples available to identify properties of gravity waves that were characteristic and physically-based and used those properties to orient our search for methods that could enhance gravity waves. This search went back into the archives for methods of image analysis that have largely fallen out of popular use as neural networks have risen to dominance on image analysis tasks. In evaluating these methods, we focused our analyses on qualitative properties and behaviors rather than on numerical scores. For early work on a large project like this, gaining an intuitive understanding of how each algorithm behaved on the data and what types of features it responded to was crucial. By understanding why an algorithm might be struggling in a situation, we were able to direct our attentions towards related algorithms and modifications that might address the issue, driving us towards ever better results.

Overall, we hope that this dissertation can serve as an example of how theoretical mathematics can be turned to very concrete applications, and that the modes of inquiry both in the chapters and in the narrative sections can show how valuable these interdisciplinary projects can be.

# Bibliography

[1] Martín Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S. Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Ian Goodfellow, Andrew Harp, Geoffrey Irving, Michael Isard, Yangqing Jia, Rafal Jozefowicz, Lukasz Kaiser, Manjunath Kudlur, Josh Levenberg, Dandelion Mané, Rajat Monga, Sherry Moore, Derek Murray, Chris Olah, Mike Schuster, Jonathon Shlens, Benoit Steiner, Ilya Sutskever, Kunal Talwar, Paul Tucker, Vincent Vanhoucke, Vijay Vasudevan, Fernanda Viégas, Oriol Vinyals, Pete Warden, Martin Wattenberg, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng. TensorFlow: Large-scale machine learning on heterogeneous systems, 2015. Software available from tensorflow.org.

[2] Henry Adams, Sofya Chepushtanova, Tegan Emerson, Eric Hanson, Michael Kirby, Francis Motta, Rachel Neville, Chris Peterson, Patrick Shipman, and Lori Ziegelmeier. Persistence images: A vector representation of persistent homology. *Journal of Machine Learning Research*, 18(8):1–35, 2017.

[3] Alexander Amini, Wilko Schwarting, Ava Soleimany, and Daniela Rus. Deep evidential regression. In H. Larochelle, M. Ranzato, R. Hadsell, M.F. Balcan, and H. Lin, editors, *Advances in Neural Information Processing Systems*, volume 33, pages 14927–14937. Curran Associates, Inc., 2020.

[4] Bernhard E. Boser, Isabelle M. Guyon, and Vladimir N. Vapnik. A training algorithm for optimal margin classifiers. In *Proceedings of the fifth annual workshop on Computational learning theory - COLT '92*. ACM Press, 1992.

[5] Noah D Brenowitz, Tom Beucler, Michael Pritchard, and Christopher S Bretherton. Interpreting and stabilizing machine-learning parametrizations of convection. *Journal of the Atmospheric Sciences*, 77(12):4357–4375, December 2020.

[6] Michael M. Bronstein, Joan Bruna, Taco Cohen, and Petar Veličković. Geometric deep learning: Grids, groups, graphs, geodesics, and gauges. *arXiv preprint*, April 2021, 2104.13478.

[7] Peter Bubenik. Statistical topological data analysis using persistence landscapes. *Journal of Machine Learning Research*, 16(1):77–102, 2015.

[8] Matthew J. Bunkers, Brian A. Klimowski, Jon W. Zeitler, Richard L. Thompson, and Morris L. Weisman. Predicting supercell motion using a new hodograph technique. *Weather and Forecasting*, 15(1):61–79, February 2000.

[9] Changyong Cao, Frank Deluccia, and NOAA JPSS Program Office. JPSS visible infrared imaging radiometer suite (VIIRS) sensor data record (SDR), 2012.

[10] Gunnar Carlsson. Topology and data. *Bulletin of the American Mathematical Society*, 46(2):255–308, January 2009.

[11] Gunnar Carlsson, Gurjeet Singh, and Afra Zomorodian. Computing multidimensional persistence. In *International Symposium on Algorithms and Computation*, pages 730–739. Springer, Springer Berlin Heidelberg, 2009.

[12] Gunnar Carlsson and Afra Zomorodian. The theory of multidimensional persistence. *Discrete & Computational Geometry*, 42(1):71–93, April 2009.

[13] Andrea Cerri, Barbara Di Fabio, Massimo Ferri, Patrizio Frosini, and Claudia Landi. Betti numbers in multidimensional persistent homology are stable functions. *Mathematical Methods in the Applied Sciences*, 36(12):1543–1557, January 2013.

[14] Moo K Chung, Peter Bubenik, and Peter T Kim. Persistence diagrams of cortical surface data. In *International Conference on Information Processing in Medical Imaging*, pages 386–397. Springer, 2009.

[15] James Clough, Nicholas Byrne, Ilkay Oksuz, Veronika A. Zimmer, Julia A. Schnabel, and Andrew King. A topological loss function for deep-learning based image segmentation using

persistent homology. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, pages 1–1, 2020.

[16] Taco Cohen and Max Welling. Group equivariant convolutional networks. In *International conference on machine learning*, pages 2990–2999. PMLR, 2016.

[17] David Cohen-Steiner, Herbert Edelsbrunner, and Dmitriy Morozov. Vines and vineyards by updating persistence in linear time. In *Proceedings of the twenty-second annual symposium on Computational geometry*, pages 119–126. ACM, ACM Press, 2006.

[18] L Denby. Discovering the importance of mesoscale cloud organization through unsupervised classification. *Geophysical Research Letters*, 47(1):e2019GL085190, January 2020.

[19] Varad Deshmukh, Srinivas Baskar, Elizabeth Bradley, Thomas Berger, and James D. Meiss. Machine learning approaches to solar-flare forecasting: Is complex better? *arXiv preprint*, February 2022, 2202.08776.

[20] R.A. DeVore, B. Jawerth, and B.J. Lucier. Image compression through wavelet transform coding. *IEEE Transactions on Information Theory*, 38(2):719–746, March 1992.

[21] M.N. Do and M. Vetterli. The finite ridgelet transform for image representation. *IEEE Transactions on Image Processing*, 12(1):16–28, January 2003.

[22] David S Dummit and Richard M Foote. *Abstract algebra*. Wiley, 2003.

[23] Imme Ebert-Uphoff and Kyle Hilburn. Evaluation, tuning, and interpretation of neural networks for working with images in meteorological applications. *Bulletin of the American Meteorological Society*, 101(12):E2149–E2170, December 2020.

[24] Herbert Edelsbrunner and John L Harer. *Computational Topology: An Introduction*. American Mathematical Society, Providence, December 2010.

[25] Sam Fletcher and Md Zahidul Islam. Comparing sets of patterns with the Jaccard index. *Australasian Journal of Information Systems*, 22, March 2018.

[26] David John Gagne, II, Sue Ellen Haupt, Douglas W Nychka, and Gregory Thompson. Interpretable deep learning for spatial analysis of severe hailstorms. *Monthly Weather Review*, 147(8):2827–2845, 2019.

[27] DJ Gagne, II, A McGovern, N Snook, R Sobash, J Labriola, JK Williams, SE Haupt, and M Xue. Hagelslag: Scalable object-based severe weather analysis and forecasting. In *Proc. Sixth Symp. on Advances in Modeling and Analysis Using Python*, 2016.

[28] Richard J. Gardner, Erik Hermansen, Marius Pachitariu, Yoram Burak, Nils A. Baas, Benjamin A. Dunn, May-Britt Moser, and Edvard I. Moser. Toroidal topology of population activity in grid cells. *Nature*, 602(7895):123–128, January 2022.

[29] Pierre Gentine, Mike Pritchard, Stephan Rasp, Gael Reinaudi, and Galen Yacalis. Could machine learning break the convection parameterization deadlock? *Geophysical Research Letters*, 45(11):5742–5751, June 2018.

[30] Robert Ghrist. Barcodes: The persistent topology of data. *Bulletin of the American Mathematical Society*, 45(01):61–76, October 2008.

[31] Jorge López González, Theodore Chapman, Kathryn Chen, Hannah Nguyen, Logan Chambers, Seraj A.M. Mostafa, Jianwu Wang, Sanjay Purushotham, Chenxi Wang, and Jia Yue. Exploring machine learning based atmospheric gravity wave detection. Technical report, UMBC GESTAR II, 2022.

[32] Karlheinz Gröchenig. *Foundations of Time-Frequency Analysis (Applied and Numerical Harmonic Analysis).* Birkhäuser Boston, 2000.

[33] Jeanpierre Guédon, editor. *The Mojette Transform: Theory and Applications*. ISTE Ltd and John Wiley and Sons, Inc, 1 edition, 2009.

[34] Liam Gumley, Jacques Descloitres, and Jeffrey Schmaltz. Creating reprojected true color MODIS images: A tutorial. Technical Report, January 2010.

[35] Lanqing Guo, Zhiyuan Zha, Saiprasad Ravishankar, and Bihan Wen. Self-convolution: A highly-efficient operator for non-local image restoration. In *ICASSP 2021 - 2021 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. IEEE, jun 2021.

[36] Tzung-Pei Hong, Ming-Jhe Hu, Tang-Kai Yin, and Shyue-Liang Wang. A multi-scale convolutional neural network for rotation-invariant recognition. *Electronics*, 11(4):661, February 2022.

[37] M F Ingham. The light of the night sky and the interplanetary medium. *Reports on Progress in Physics*, 34(3):875–912, September 1971.

[38] ITU-R. Studio encoding parameters of digital television for standard 4:3 and wide-screen 16:9 aspect ratios. Technical Report BT.601, International Telecommunication Union, 2011.

[39] Paul Jaccard. Étude comparative de la distribution florale dans une portion des alpes et des jura. *Bull Soc Vaudoise Sci Nat*, 37:547–579, 1901.

[40] Hannah Kim and Christian Vogel. Deciphering active wildfires in the Southwestern USA using topological data analysis. *Climate*, 7(12):135, November 2019.

[41] Emily J. King, Rafael Reisenhofer, Johannes Kiefer, Wang-Q Lim, Zhen Li, and Georg Heygster. Shearlet-based edge detection: flame fronts and tidal flats. In Andrew G. Tescher, editor, *Applications of Digital Image Processing XXXVIII*, volume 9599 of *Society of Photo-Optical Instrumentation Engineers (SPIE) Conference Series*, September 2015.

[42] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint*, December 2014, 1412.6980.

[43] Andrew Kingston, Imants Svalbe, and Jean-Pierre Guédon. The discrete Radon transform: a more efficient approach to image reconstruction? In Stuart R. Stock, editor, *Developments in X-Ray Tomography VI*. SPIE, aug 2008.

[44] Peter Kovesi. Image features from phase congruency. *Videre: A Journal of Computer Vision Research*, 1(3):1–26, 1999.

[45] Peter Kovesi. Phase congruency: a low-level image invariant. *Psychol. Res.*, 64:136–148, 2000.

[46] Miroslav Kramár, Rachel Levanger, Jeffrey Tithof, Balachandra Suri, Mu Xu, Mark Paul, Michael F Schatz, and Konstantin Mischaikow. Analysis of Kolmogorov flow and Rayleigh–Bénard convection using persistent homology. *Physica D: Nonlinear Phenomena*, 334:82–98, November 2016.

[47] Vladimir M Krasnopolsky, Michael S Fox-Rabinovitz, and Dmitry V Chalikov. New approach to calculation of atmospheric model physics: Accurate and fast neural network emulation of longwave radiation in a climate model. *Monthly Weather Review*, 133(5):1370–1383, May 2005.

[48] Peter Lawson, Andrew B Sholl, J Brown, Brittany Terese Fasy, and Carola Wenk. Persistent homology for the quantitative evaluation of architectural features in prostate cancer histology. *Scientific reports*, 9(1):1–15, February 2019.

[49] Tristan S. L'Ecuyer, H. K. Beaudoing, M. Rodell, W. Olson, B. Lin, S. Kato, C. A. Clayson, E. Wood, J. Sheffield, R. Adler, G. Huffman, M. Bosilovich, G. Gu, F. Robertson, P. R. Houser, D. Chambers, J. S. Famiglietti, E. Fetzer, W. T. Liu, X. Gao, C. A. Schlosser, E. Clark, D. P. Lettenmaier, and K. Hilburn. The observed state of the energy budget in the early twenty-first century. *Journal of Climate*, 28(21):8319–8346, October 2015.

[50] Clément Maria, Jean-Daniel Boissonnat, Marc Glisse, and Mariette Yvinec. The gudhi library: Simplicial complexes and persistent homology. In *Mathematical Software – ICMS 2014*, pages 167–174. Springer Berlin Heidelberg, 2014.

[51] Amy McGovern, Imme Ebert-Uphoff, David John Gagne, and Ann Bostrom. Why we need to focus on developing ethical, responsible, and trustworthy artificial intelligence approaches for environmental science. *Environmental Data Science*, 1:E6, 2022.

[52] Amy McGovern, Ryan Lagerquist, David John Gagne, G Eli Jergensen, Kimberly L Elmore, Cameron R Homeyer, and Travis Smith. Making the black box more transparent: Understanding the physical implications of machine learning. *Bulletin of the American Meteorological Society*, 100(11):2175–2199, November 2019.

[53] Richard Brian Merritt. Visualizing planetary rossby waves with topological data analysis. Master's thesis, Statistics, University of Georgia, 2021.

[54] Yuriy Mileyko, Sayan Mukherjee, and John Harer. Probability measures on the space of persistence diagrams. *Inverse Problems*, 27(12):124007, November 2011.

[55] Steven D. Miller, William C. Straka, Jia Yue, Steven M. Smith, M. Joan Alexander, Lars Hoffmann, Martin Setvák, and Philip T. Partain. Upper atmospheric gravity wave details revealed in nightglow satellite imagery. *Proceedings of the National Academy of Sciences*, 112(49), November 2015.

[56] Stephen Mills and Steven Miller. VIIRS day/night band—correcting striping and nonuniformity over a very large dynamic range. *Journal of Imaging*, 2(1):9, March 2016.

[57] Michel Misiti, Yves Misiti, Georges Oppenheim, and Jean-Michel Poggi. *Wavelets and their Applications*, volume 330. Iste London, UK:, 2007.

[58] Davide Moroni and Maria Antonietta Pascali. Learning topology: Bridging computational topology and machine learning. *Pattern Recognition and Image Analysis*, 31(3):443–453, July 2021.

[59] Grzegorz Muszynski, Karthik Kashinath, Vitaliy Kurlin, and Michael Wehner. Topological data analysis and machine learning for recognizing atmospheric river patterns in large climate datasets. *Geoscientific Model Development*, 12(2):613–628, February 2019.

[60] Dorcas Ofori-Boateng, Huikyo Lee, Krzysztof M. Gorski, Michael J. Garay, and Yulia R. Gel. Application of topological data analysis to multi-resolution matching of aerosol optical depth maps. *Frontiers in Environmental Science*, 9, June 2021.

[61] Stephan Rasp, Michael S Pritchard, and Pierre Gentine. Deep learning to represent sub-grid processes in climate models. *Proceedings of the National Academy of Sciences*, 115(39):9684–9689, September 2018.

[62] Stephan Rasp, Hauke Schulz, Sandrine Bony, and Bjorn Stevens. Combining crowdsourcing and deep learning to explore the mesoscale organization of shallow convection. *Bulletin of the American Meteorological Society*, 101(11):E1980–E1995, 2020.

[63] Jan Reininghaus, Stefan Huber, Ulrich Bauer, and Roland Kwitt. A stable multi-scale kernel for topological machine learning. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 4741–4748. IEEE, June 2015.

[64] Rafael Reisenhofer. The complex shearlet transform and applications to image quality assessment. Master's thesis, Technische Universität Berlin, 2014.

[65] Rafael Reisenhofer, Johannes Kiefer, and Emily J. King. Shearlet-based detection of flame fronts. *Exp. Fluid.*, 57(3):41, February 2016.

[66] Rafael Reisenhofer and Emily J. King. Edge, ridge, and blob detection with symmetric molecules. *SIAM Journal on Imaging Sciences*, 12(4):1585–1626, January 2019.

[67] Olaf Ronneberger, Philipp Fischer, and Thomas Brox. U-net: Convolutional networks for biomedical image segmentation. *arXiv preprint*, May 2015, 1505.04597.

[68] Cynthia Rudin. Stop explaining black box machine learning models for high stakes decisions and use interpretable models instead. *Nature Machine Intelligence*, 1(5):206–215, May 2019.

[69] Timothy J. Schmit, Paul Griffith, Mathew M. Gunshor, Jaime M. Daniels, Steven J. Goodman, and William J. Lebair. A closer look at the ABI on the GOES-r series. *Bulletin of the American Meteorological Society*, 98(4):681–698, April 2017.

[70] MG Schultz, Clara Betancourt, Bing Gong, Felix Kleinert, Michael Langguth, LH Leufen, Amirpasha Mozaffari, and Scarlet Stadtler. Can deep learning beat numerical weather prediction? *Philosophical Transactions of the Royal Society A: Mathematical, Physical and Engineering Sciences*, 379(2194):20200097, February 2021.

[71] Craig S. Schwartz and Ryan A. Sobash. Revisiting sensitivity to horizontal grid spacing in convection-allowing models over the central and eastern united states. *Monthly Weather Review*, 147(12):4411–4435, November 2019.

[72] Roy Schwartz, Jesse Dodge, Noah A Smith, and Oren Etzioni. Green AI. *Communications of the ACM*, 63(12):54–63, November 2020.

[73] Ignacio Segovia-Dominguez, Zhiwei Zhen, Rishabh Wagh, Huikyo Lee, and Yulia R. Gel. TLife-LSTM: Forecasting future COVID-19 progression with topological signatures of atmospheric conditions. In *Advances in Knowledge Discovery and Data Mining*, pages 201–212. Springer International Publishing, 2021.

[74] Caio Átila Pereira Sena, João Antônio Recio da Paixão, and José Ricardo de Almeida França. A topological data analysis approach for retrieving local climate zones patterns in satellite data. *Environmental Challenges*, 5:100359, December 2021.

[75] William Skamarock, Joseph Klemp, Jimy Dudhia, David Gill, Dale Barker, Wei Wang, Xiang-Yu Huang, and Michael Duda. A description of the Advanced Research WRF version 3. NCAR TN-475+STR. Technical report, UCAR/NCAR, 2008.

[76] Ryan A. Sobash, David John Gagne, II, Charlie L. Becker, David Ahijevych, Gabrielle N. Gantos, and Craig S. Schwartz. Diagnosing storm mode with deep learning in convection-allowing models. In review at *Monthly Weather Review*, 2023.

[77] Ryan A. Sobash, Glen S. Romine, and Craig S. Schwartz. A comparison of neural-network and surrogate-severe probabilistic convective hazard guidance derived from a convection-allowing model. *Weather and Forecasting*, 35(5):1981–2000, October 2020.

[78] Bjorn Stevens, Sandrine Bony, Hélène Brogniez, Laureline Hentgen, Cathy Hohenegger, Christoph Kiemle, Tristan S L'Ecuyer, Ann Kristin Naumann, Hauke Schulz, Pier A Siebesma, Jessica Vial, Dave M. Winker, and Paquita Zuidema. Sugar, gravel, fish and flowers: Mesoscale cloud patterns in the trade winds. *Quarterly Journal of the Royal Meteorological Society*, 146(726):141–152, November 2020.

[79] Kristian Strommen, Matthew Chantry, Joshua Dorrington, and Nina Otter. A topological perspective on weather regimes. *Climate Dynamics*, July 2022.

[80] Hu Sun, Ward Manchester, and Yang Chen. Improved and interpretable solar flare predictions with spatial and topological features of the polarity inversion line masked magnetograms. *Space Weather*, 19(12), December 2021.

[81] The RIVET Developers. Rivet, 2020.

[82] Chad M Topaz, Lori Ziegelmeier, and Tom Halverson. Topological data analysis of biological aggregation models. *PLOS ONE*, 10(5):e0126383, May 2015.

[83] Sarah Tymochko, Elizabeth Munch, Jason Dunion, Kristen Corbosiero, and Ryan Torn. Using persistent homology to quantify a diurnal cycle in hurricanes. *Pattern Recognition Letters*, 133:137–143, May 2020.

[84] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention is all you need. *arXiv preprint*, June 2017, 1706.03762.

[85] Lander Ver Hoef, Henry Adams, Emily J. King, and Imme Ebert-Uphoff. A primer on topological data analysis to support image analysis tasks in environmental science. *Artificial Intelligence for the Earth Systems*, pages 1–38, December 2022.

[86] Martin Vetterli, Jelena Kovačević, and Vivek K Goyal. *Foundations of Signal Processing*. Cambridge University Press, September 2014.

[87] Jingjing Xu, Wangchunshu Zhou, Zhiyi Fu, Hao Zhou, and Lei Li. A survey on green deep learning. *arXiv preprint*, November 2021, 2111.05193.

[88] Janni Yuval and Paul A. O'Gorman. Stable machine-learning parameterization of subgrid processes for climate modeling at a range of resolutions. *Nature Communications*, 11(1), July 2020.

[89] Xiao Xiang Zhu, Devis Tuia, Lichao Mou, Gui-Song Xia, Liangpei Zhang, Feng Xu, and Friedrich Fraundorfer. Deep learning in remote sensing: A comprehensive review and list of resources. *IEEE Geoscience and Remote Sensing Magazine*, 5(4):8–36, December 2017.