A Representational Response Analysis Framework For Convolutional Neural Networks

by

Andrew Hryniowski

A thesis
presented to the University of Waterloo
in fulfillment of the
thesis requirement for the degree of
Doctor of Philosophy
in
Systems Design Engineering

Waterloo, Ontario, Canada, 2024

Examining Committee Membership

The following served on the Examining Committee for this thesis. The decision of the Examining Committee is by majority vote.

External Examiner:	Neil Bruce Associate Professor, School of Computer Science, University of Guelph
Supervisor:	Alexander Wong Professor, Dept. of Systems Design Engineering, University of Waterloo
Internal Member:	John Zelek Associate Professor, Dept. of Systems Design Engineering, University of Waterloo
Internal Member:	David Clausi Professor, Dept. of Systems Design Engineering, University of Waterloo
Internal-External Member:	Fakhri Karray Professor, Dept. of Electrical and Computer Engineering, University of Waterloo

Author's Declaration

This thesis consists of material all of which I authored or co-authored: see Statement of Contributions included in the thesis. This is a true copy of the thesis, including any required final revisions, as accepted by my examiners.

I understand that my thesis may be made electronically available to the public.

Statement of Contributions

The follow papers are used in this thesis. I was the primary co-author on all of these works, including: research, design, implementation, experimentation, and writing.

A. Hryniowski, A. Wong "Inter-layer Information Similarity Assessment of Deep Neural Networks Via Topological Similarity and Persistence Analysis of Data Neighbour Dynamics", New In ML, Conference on Neural Information Processing Systems (NeurIPS), 2020.

This paper is incorporated in Chapters 3 and 8 of this thesis.

A. Hryniowski, A. Wong. "Systematic Architectural Design of Scale Transformed Attention Condenser DNNs via Multi-Scale Class Representational Response Similarity Analysis", Fourth workshop on Neural Architecture Search, Conference on Computer Vision and Pattern Recognition (CVPR), 2023.

This paper is incorporated in Chapters 6 and 7 of this thesis.

Abstract

Over the past decade, convolutional neural networks (CNNs) have become the defacto machine learning model for image processing due to their inherent ability to capitalize on modern data availability and computational resources. Much of a CNN's capabilities come from their modularity and flexibility in model design. As such, practitioners have been able to successfully tackle applications not previously possible with other contemporary methods. The downside to this flexibility is that it makes designing and improving upon a CNN's performance an arduous task.

Designing a CNN is not a straightforward process. Model architecture design, learning strategies, and data selection and processing must all be precisely tuned for a researcher to produce even a non-random performing model. Finding the correct balance to achieve start-of-the-art can be its own challenge requiring months or years of effort. When building a new model, researchers will rely on quantitative metrics to guide the development process. Typically, these metrics revolve around model performance characteristic constraints (e.g., accuracy, recall, precision, robustness) and computational (e.g., number of parameters, number of FLOPS), while the learned internal data processing behaviour of a CNN is ignored.

Some research investigating the internal behaviour of CNNs has been proposed and adopted by a niche group within the broader deep learning community. Because these methods operate on extremely high dimensional latent embeddings (between one to three orders of magnitude larger than the input data) they are computationally expensive to compute. In addition, many of the most common methods do not share a common root from which downstream metrics can be computed, thus making the use of multiple metrics prohibitive.

In this work we propose a novel analytic framework that offers a broad range of complementary metrics that can be used by a researcher to study the internal behaviour of a CNN, and whose findings can be used to guide model performance improvements. We call the proposed framework Representational Response Analysis (RRA). The RRA framework is built around a common computational kNN based model of the latent embeddings of a dataset at each layer in a CNN. Using the information contained within these kNNs, we propose three complementary metrics that extract targeted information and provides a researcher with the ability to investigate specific behaviours of a CNN across all of its layers.

For this work we focus our attention on classification based CNNs and perform two styles of experiments using the proposed RRA framework. The first set of experiments

revolve around better understanding RRA hyper-parameter selection and the impacts on the downstream metrics with regards to observed characteristics of a CNN. From this first set of experiments we determine the effects of adjusting specific RRA hyper-parameters, and we propose general guidelines for selecting these hyper-parameters. The second set of experiments investigates the impact of specific CNN design choices. To be more precise, we use RRA to investigate the consequences on a CNN's latent representation when training with and without data augmentations, and to understand the latent embedding symmetries across different pooled spatial resolutions. For each of these experiments RRA provides novel insights into the internal workings of a CNN.

Using the insights from the pooled spatial resolution experiments we propose a novel CNN attention-based building block that is specifically designed to take advantage of key latent properties of a ResNet. We call the proposed building block the Scale Transformed Attention Condenser (STAC) module. We demonstrate that the proposed STAC module not only improves a model's performance across a selection of model-dataset pairs, but that it does so with an improved performance-to-computational-cost tradeoff when compared to other CNN spatial attention-based modules of similar FLOPS or number of parameters.

Acknowledgements

I would like to thank my advisor Dr. Alexander Wong for being my mentor, providing me with guidance, and being patience with me as I completed my PhD part-time.

I would like to thank both Prof. John Zelek, Prof. David Clausi, and Prof. Fakhri Karray for serving on my PhD examining committee, and Prof. Neil Bruce for serving as my external examiner.

I would to thank DarwinAI for providing me with some of the computational resources I used to perform experiments in this work, and for allowing me flexibility in my working hours when required.

Finally, I would like to thank my loving parents and friends who have always given me the emotional support I have needed.

Table of Contents

List of Figures

List of Tables

List of Abbreviations

CCA Canonical Correlation Analysis

CKA Centered Kernel Alignment

CNN Convolutional Neural Network

HSIC Hilbert-Schmidt Independence Criterion

ID Intrinsic Dimensionality

LP Linear Probing

LS Layer Similarity

MS-RRA Multi-Scale Representational Response Analysis

NDR Neighbour Density Ratio

NNLS Nearest Neighbour Layer Similarity

RR Representational Response

RRA Representational Response Analysis

STAC Scale Transformed Attention Condenser

TDA Topological Data Analysis

List of Symbols

E A set of relationships between contiguous functions in a CNN

G A CNN

H A k-nearest neighbour graph

K A set of ordered sets of neighbours in a k-nearest neighbour graph

M A manifold

Q A nearest neighbour class similarity function

R A distance function used in k-nearest neighbour graph

V A set of functions within a CNN

W A set of all weights that parameterize the functions within a CNN

X A set of data

Y A set of output feature vectors produced by some function within a CNN

d The dimensionality of a vector

e A relationships between two contiguous functions in a CNN

n The number of samples in a set of data

v A function within a CNN

w A set of weights that parameterize a single function within a CNN

x A single sample of data

y A feature vector produced by some function within a CNN for a single data sample

Chapter 1

Introductions

Convolutional Neural Networks (CNNs) are an important tool for solving image-based challenges for modern machine learning practitioners. Inspired by the human visual system, modern CNNs started with solving problems in handwritten digit recognition [50] and have since demonstrated incredible results across several applications, including: distinguishing between 1000 classes [14], localizing objects within an image [67], and synthesizing new images [28]. The spread of CNNs can be attributed to multiple factors, including: the wide spread availability of high quality images [14], an increase in CNN model sizes [85], advanced CNN model architectures [78], and improved learning procedures [75]; all of which are downstream consequences of cheap computational resources. Despite these recent advances, CNNs are in principle large computational black boxes [56], making it prohibitive for a practitioner to compare CNNs beyond basic performance metrics, improve a CNN's performance, or adapt existing methods to new problem spaces.

1.1 Motivation

Designing a new CNN or improving upon a one can be an intensive task involving a wealth of specialized knowledge, skill in hypothesis generation, and a time intensive cycle of experimental trial and error. For a given problem, determining which components or processes should be modified, removed, or added generally comes down to a practitioner's intuition and quantitative experimentation measured by performance-based metrics (e.g., accuracy, recall, precision). However, the use of other analytic metrics that assess intrinsic qualities of a CNN (e.g., explainability [54, 57, 68], statistical [49, 65], adversarial robustness metrics [29, 54, 60], 2D/3D latent feature projections [58, 76]) are infrequently employed,

reported on, or used in a substantive manner. The modern CNN design life cycle contains no ubiquitous measure that plays an important role in the CNN design process. No intrinsic analytic metric has been found to be widely applicable, computationally tractable, and low effort to apply. Methods that fill this gap can propel the field of deep learning forward by streamlining the CNN development life cycle.

1.2 CNN Analytic Metrics

The choice of metrics used for evaluation directly affects the nature of potential insights. In this work we put forward a conceptual three tier nested hierarchy of CNN evaluation methods.

1. **Zeroth Order**: Metrics which rely on the direct output of a model. These metrics provide targeted knowledge about a model's output and are the most common form of analysis. Output of CNN classification models can include logits, a probability distribution, or an energy map. Example metrics derived using a CNN's output include accuracy, recall, precision, and training loss.

2. **First Order**: Metrics which directly use the internal latent representations of data produced by a CNN. For CNNs the latent representations can be the input or output of a layer in the CNN, or even a representation from within a specific layer of a CNN. These metrics provide knowledge about latent distributions and a mechanism for comparing them (often focusing on similarity metrics). Examples include statistical measures such as Canonical Correlation Analysis (CCA) [34], Hilbert-Schmidt Independence Criterion (HSIC) [32], and Centered Kernel Alignment (CKA) [49].

3. **Second Order**: Metrics which build models on top of a CNN's latent representations, and then extract meaningful insights. These metrics typically use models to distil information present in first order methods but provide a more convenient and targeted set of tools for analyzing a CNN. Examples include Intrinsic Dimensionality (ID) [3], linear probing [2], Topological Data Analysis (TDA) [7], and adversarial detection [60]. Most of the novel methods proposed in this work fall into this category.

Figure 1.1 depiction of the hierarchy of methods and where the proposed common analytic model (highlighted in red) is situated.

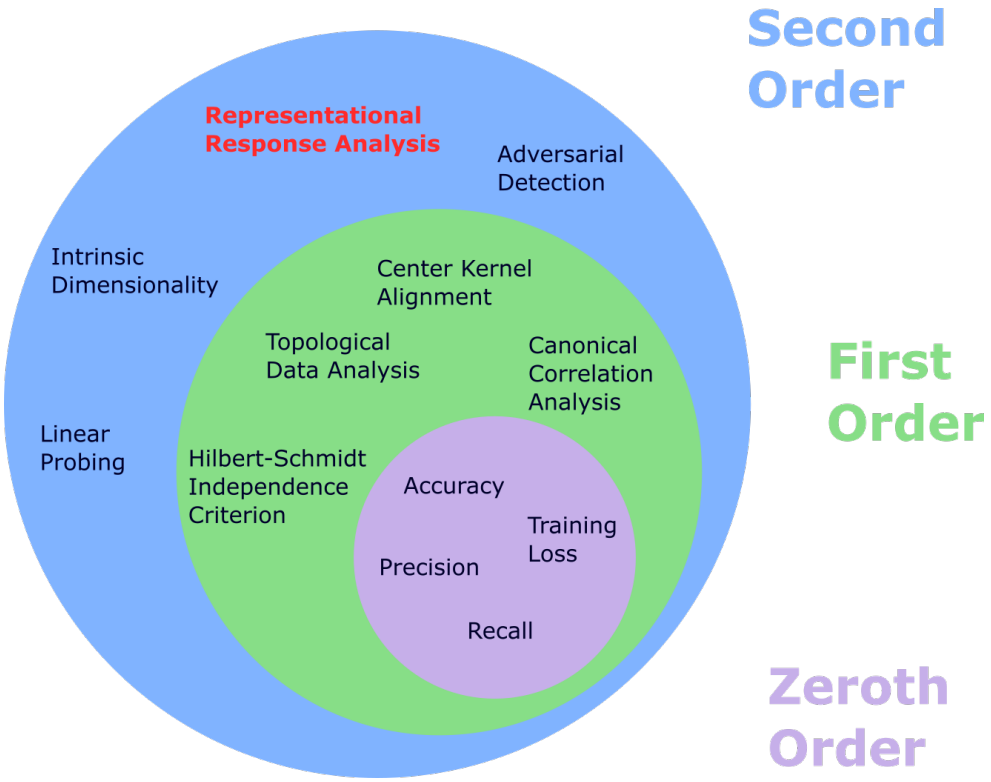


Figure 1.1: A depiction of the nested hierarchy analytic methods used on CNNs. Each order contains representative methods. Novel research in this work falls in the second order category.

1.3 Challenges and Objectives

The three categories of metrics each come with their own set of benefits and challenges. Zeroth order metrics are great iterative tools, often require only a single forward pass of a CNN (per sample), and are generally limited to operating on feature vectors smaller than a few thousand dimensions. These benefits require that CNNs be treated as black box entities. First order methods do away with the black box constraint by directly using a CNN's internal latent representations which allows for the direct analysis of specific sections, blocks, layers, or even functions within any given CNN. Operating directly on these representations can require orders of magnitude more system memory and computational overhead. Because of the added development and compute time required, anything beyond zeroth order methods are often overlooked, or are used in a limited capacity (e.g., only operating on a few select layer outputs). Another difficulty using first order methods is that they are operating on

Table 1.1: Qualitative high level tradeoffs between analytic metrics

Metric Order	Initial Cost	Repeat Cost	Is CNN Black Box	Interpretability
Zeroth	Low	Low	Yes	High
First	High	High	No	Medium
Second	High	Low	No	High

raw CNN features making the metrics difficult to interpret. Second order methods extend first order methods by imposing a model on top of the raw CNN latent representations. The modelling process moves much of the computational cost incurred by first order methods to a one-time cost of modelling the latent representations. Using the common latent model then allows for derivative models to be applied in a tractable manner. The tradeoffs between these methods are qualitatively summarized in Table 1.1.

Each category of metric has its own unique place in the development of CNNs, but knowing in which situation to use them presents its own challenge. Zeroth order metrics are great for rapid iteration, first order metrics are a direct reflection of a CNN's latent representation, and second order metrics allow for more targeted and interpretable aspects of a CNN's latent representation to be distilled. However, each of these categories contains multiple metrics with their own focus and that may require a different set of calculations to be computed.

In general, there is no one metric that will suffice for building or improving a classification CNN. A model's designer will have to choose from a selection of approaches spanning the different categories of metrics. Though, in practice modern classification CNN's are developed with little more in mind than classification accuracy. The overarching goal of this work is to propose a set of complementary analytic metrics (comprising of second order metrics) that are feasible to use during CNN model development, and which demonstrate the ability to provide actionable insights for CNN model improvement. More concretely, the core objectives of this work are:

- To propose a unified model of a CNN's latent representations from which analytic metrics (both existing and novel) can be efficiently computed,
- To perform a multi-modal analysis to see how these metrics are complementary, and
- To generate novel CNN structures using outcomes from the proposed analytic methods.

This work achieves these objectives by formalizing an analytic framework called Representational Response Analysis (RRA) (Chapter 1.4). The proposed framework utilizes a

kNN to model each data manifold within a given CNN (Chapter 3). Using the common model we develop a series of novel metrics for measuring specific aspects of a CNN's latent representations (Chapters 3, 6). Using insights from our RRA analysis we propose an improved ResNet attention module and demonstrate that it generalizes on different classification datasets and ResNet style architectures (Chapter 7).

1.4 Representational Response Analysis Framework

All three orders of CNN analysis rely on extracting information from the latent representations of a dataset (zeroth order trivially so). In this section we formally lay out a framework from which most other CNN analysis can be derived; we call this framework RRA. How other forms of analysis are derived from the RRA framework is discussed throughout this work.

Let $\mathbf{x}_i \in \mathbf{X}$ be a set of input samples of shape $n \times d$, and let $\mathbf{G} = (\mathbf{V}, \mathbf{E}, \mathbf{W})$, represent a CNN, where $\mathbf{V} = \{v\}$ is a set of sub-functions (e.g., layers), $\mathbf{E} = \{e\}$ is a set of edges that represent the sub-function's i/o relationships, and $\mathbf{W} = \{w\}$ is a set of weights that parameterize the sub-functions. Let the output of some sub-function v_v for the $\mathbf{x}_i$ sample be defined as $\mathbf{y}_{vi} = v_v(\mathbf{x}_i; G_v)$, where $G_v \subseteq G$ contains all required sub-functions, edges, and weights to calculate $\mathbf{y}_{vi}$. Let the output of a given function $\mathbf{Y}_v = \{y_v\}_{x \in X}$.

Data passed through a CNN G is mapped from one space to another; at each layer salient information is preserved and combined to form new representations, while irrelevant information is discarded. The relationship between samples in each layer forms an intrinsic structure known as a manifold. In this work we study the structure of individual manifolds, how manifolds between layers in a CNN can change, and the relationship between manifolds of different CNNs.

For a given CNN G and data samples $\mathbf{X}$, let the Representational Response (RR) of a CNN G acting on a set of data $\mathbf{X}$ be defined as the set of manifolds $\{M_v\}_{v \in V}$ generated by G at each layer (or sub-layer) as data samples are propagated throughout G (including the base manifold of the raw dataset $\mathbf{X}$).

$$RR^G_{\mathbf{X}} = M_{\mathbf{X}} \cup \{M_v\}_{v \in V} \tag{1.1}$$

Throughout this work, when discussing the RR of a CNN, assume it is about the CNN operating on some dataset unless explicitly stated otherwise. For notational convenience we assign the input manifold $M_X = M_0$ and all other manifolds to some index $i > 0$.

$$RR^G_{\mathbf{X}} = \{M_i\}_{0 \leq i < |V|} \tag{1.2}$$

As previously noted, aspects of a $RR^G_{\mathbf{X}}$ have been studied in a piecemeal manner through metrics such as CKA, ID, etc. The contents of this work furthers the study of the RR of a CNN through a systematic approach using the proposed RRA framework, an analytic model of $RR^G_{\mathbf{X}}$, and a collection of novel metrics which extract information from $RR^G_{\mathbf{X}}$.

From a practical perspective $RR^G_{\mathbf{X}}$ is a set of latent space models that approximate the underlying data manifold's interactive with a CNN. We approximated each layer's manifold using the latent embeddings of a dataset at that layer. Throughout this work we model each layer's manifold using kNNs trained on the dataset's latent embeddings at each layer. Then, we measure aspects of the set of kNNs using three core metrics. See Figure 1.2 for a generalized depiction of the modelling approach used in this work.

Using preexisting terminology, the closest phrase to describe a CNN's representational response would be something like "a dataset's latent embeddings within a CNN". While this is an adequate description in many cases, it falls short in a few areas:

1. The phrase doesn't explicitly include the input data embedding (e.g., raw images),
2. The term "latent embedding" is often used to refer to the final feature embeddings of a CNN, and
3. There is no implied focus on studying the interrelations between progressively deeper latent embeddings.

Throughout this work we use the term latent embeddings to describe specific instantiations of feature vectors produced at any given layer in a CNN by passing a data sample (or a set of data samples) through a CNN. The term representational response is used to specifically imply the study of how a collection of latent embedding relates to one another.

1.5 Contributions

The following are the key contributions of this work:

1. **Representational Response Analysis (RRA)**(Chapter 3, 5) A novel framework for analyzing CNNs. The proposed RRA framework utilizes a manifold modelling technique that enables a set of multi-modal complementary analytic methods to be efficiently calculated and used to study targeted aspects of the internal workings of a CNN. To the best of the author's knowledge this is the first deep learning

analytic framework to explicitly consider an efficient approach for using multiple complementary metrics in a computationally efficient manner. The three metrics are: 1) Intrinsic Dimenionality (ID), 2) Nearest Neighbour Class Similarity (NNCS), and 3) Nearest Neighbour Layer Similarity (NNLS). NNCS and NNLS are novel methods proposed in this work.

2. **RRA on the Effects of Applying Augmentation During Training** (Chapter 4) We apply RRA to compare CNNs trained with and without augmentations to demonstrate the utility of RRA and to better understand the impacts of augmentation on the latent representation structure of CNNs. By using RRA we are able to detect novel phenomena within a CNN's latent structure, show how this structure compares between training and validation data partitions, and how the novel phenomena changes when augmentations are either applied or withheld during training.

3. **Multi-Scale Representational Response Analysis (MS-RRA)**: (Chapter 6) A novel type of representational response analysis that identifies the importance of features at varying spatial scales across the layers of a CNN. Built on top of RRA, MS-RRA compares the latent manifolds of a dataset at different spatial scales, and investigates how the similarity of the manifolds changes as the spatial resolution of each manifold is altered.

4. **Scale Transformed Attention Condenser (STAC)**: (Chapter 7) Using insights gained from MS-RRA we propose STAC modules, a novel instantiation of attention condenser modules which are designed to emphasize the spatial similarity found in latent embeddings. When a STAC module is used within a CNN it provides improved model performance while requiring fewer parameters and FLOPS compared to existing attention mechanisms.

1.6 Thesis Structure

Chapter 2 presents a formulation of CNNs and provides a literature review of common deep learning design methodologies used to evaluate CNNs. Chapter 3 formulates the proposed RRA framework and derives three complementary metrics from the RRA framework; each of which are each designed to study targeted aspects of a CNN's RR. Chapter 4 uses RRA to compare CNNs trained with and without augmentation during the training process. Chapter 5 investigates how the number of samples and number of neighbours per sample used to construct the underlying nearest neighbour graph affects observations from the RR

metrics. Chapter 6 proposes multi-scale representational response analysis (MS-RRA), an approach which applies RRA across multiple spatial scales. Chapter 7 proposes a novel CNN module derived from insights from MS-RRA. Chapter 8 ties together the observations from this work, draws conclusions, and suggests future directions for this research.

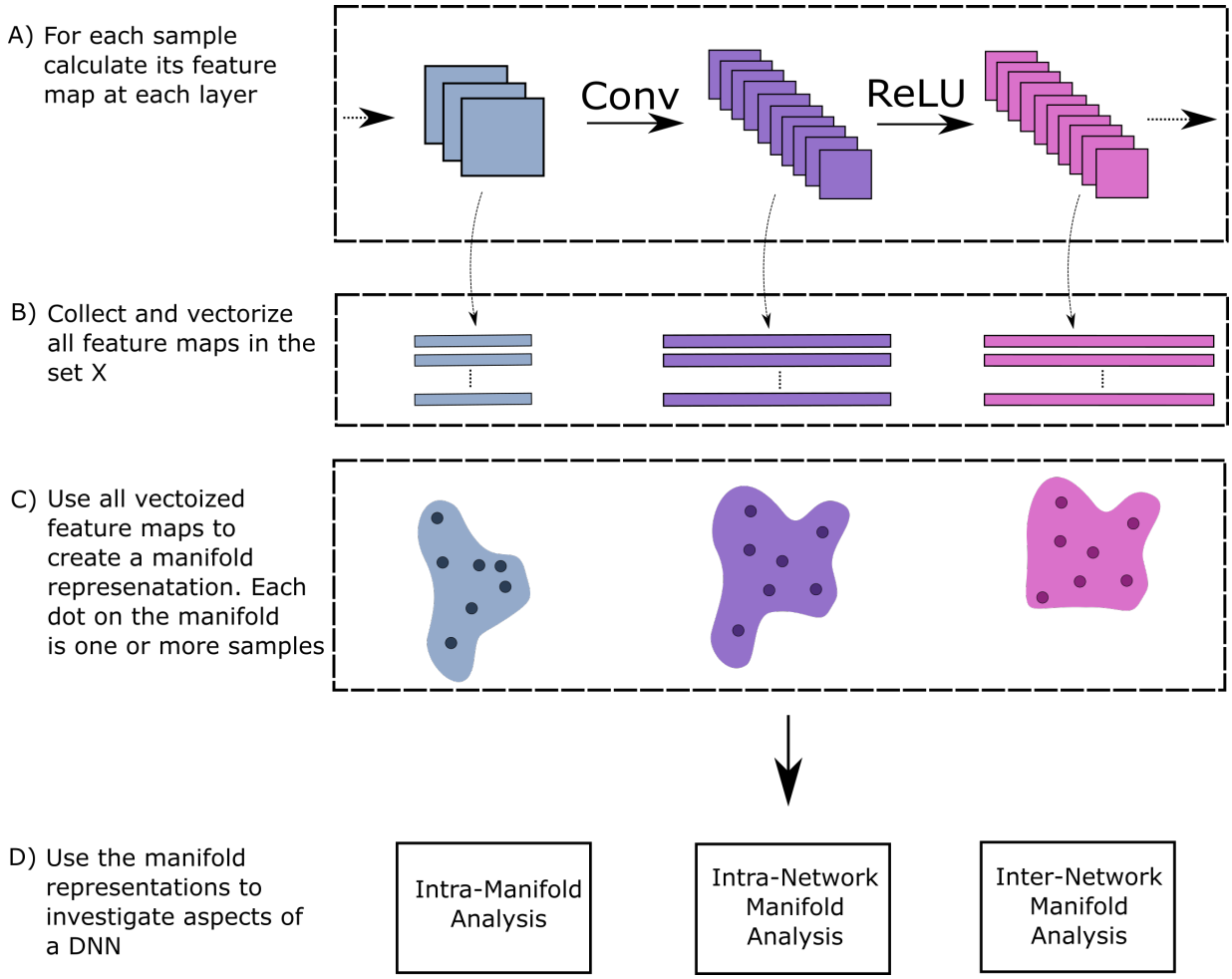


Figure 1.2: An overview of the framework used in this work to analyze the RR of a CNN. The following steps are performed every time an aspect of the CNN is changed: A) All samples in a dataset X are run through a CNN. At each layer (or at each operation in each layer, depending on the level of analysis) the feature maps for each sample in X are recorded. B) All feature maps at each layer are vectorized. C) The set of vectorized feature maps for each layer are used to create a model of the layer's latent manifold. D) The model of each manifold is used to investigate properties of a CNN in one of three categories of analysis: 1) aspects of an individual representation manifold, 2) differences between manifolds within a single CNN, and 3) differences between manifolds of different CNN's.

Chapter 2

Background

2.1 Neural Networks

Neural networks are a type of machine learning model designed to learn a mapping function f of information from one domain $\mathbf{x} \in \mathbf{X}$ to another domain $\mathbf{y} \in \mathbf{Y}$. The mapping function can be decomposed into a graph of functions $G = (V, E, W)$. The process for selecting V, E, and W depends on the application. It is standard for V and E to be hand designed and refined through trial and error by a deep learning practitioner, and for the weights W to be trained using a gradient descent-based algorithm [51]. The objective for both procedures is described by

$$V, E, W = \arg\min_{V,E,W} \sum_i \mathcal{L}(\mathbf{y}_i, \hat{\mathbf{y}}_i) \tag{2.1}$$

where $\mathcal{L}(\cdot)$ is some error metric for the task at hand (e.g., percent error, cross-entropy, mean squared error, number of connections). The specific form of training W depends on the application. Applications fall into one or more of the following categories: i) supervised learning, ii) unsupervised learning, and iii) reinforcement learning. Supervised learning applications are when each input in a training set $\mathbf{x}_i$ has a known output $\mathbf{y}_i$. Unsupervised learning is when the outputs for a training set are unknown. Reinforcement learning is when the objective being learnt is unknown or too hard to mathematically describe.

The type of neural network is determined by the sub-functions V and their relations to one another as described by E. Different types of neural networks include feedforward [51], convolutional [52], and recurrent [42]. In this work we only consider primarily supervised feedforward convolutional neural networks (CNNs).

2.2 Feedforward Convolutional Neural Networks

The class of operations within G for a simple feedforward CNN are a combination of atomic operations, including convolution, non-linear functions, pooling, batchnorms [44], matrix multipliers, and a softmax for classification models. Most of these operations appear multiple times within G for most CNNs. The operations within G are connected via feedforward edges E (i.e., no subset of edges $E_i \subseteq E$ can contain a cycle). The tuple (V, E) forms a poset (partial ordered set) of operations. The poset (V, E) often contains repeated structures of atomic operations that only differ in their parameterization (i.e., the weights associated with each operation). A common structure within a CNN is the sequential chain of operations

$$convolution \longrightarrow bias \longrightarrow activation \longrightarrow batchnorm \tag{2.2}$$

Such structures are called *layers* of the CNN. Like atomic operations, layers can also exist within other more complex layers. Below is an example of a ResNet [39] layer (the defacto layer design for most modern image classification CNNs)

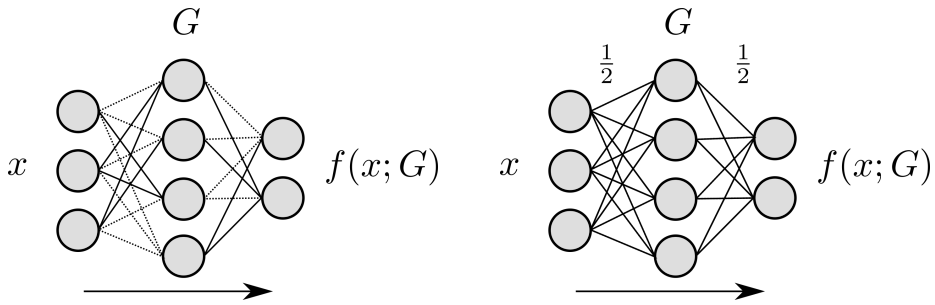
$$\underbrace{layer \longrightarrow layer \longrightarrow sum}_{\curvearrowright} \tag{2.3}$$

As the name denotes, convolution is a central part of a CNN's graph G. Convolutional operations are used in neural networks because convolution exploits translation invariance; a property common in many forms of data, (e.g., natural images, audio signals) thus allowing an efficient encoding of local feature detectors (i.e., weight sharing). In addition, stacking a series of convolutional operations allows for an effective means of detecting hierarchical features.

2.3 CNN Design Methods

2.3.1 Standard Design Methods

Over time the types of layers used within CNNs has grown to provide a diverse set for a practitioner to choose from. Originally, CNNs weren't very deep compared to contemporary designs and took a long time to train, if they trained at all. A large part of the problem was that back propagation (backprop) updates parameters near the input, which can have a large impact on downstream layers. Ioffe *et al.* [44] proposed the batchnorm layer design to alleviate this issue by normalizing the data distribution periodically throughout a network.



(a) Dropout during training (b) Dropout during inference

Figure 2.1: An illustration of dropout during (a) training and (b) inference. In this example the drop rate is 0.5. During training at every update, a random 50% of all connections are set to 0, as depicted by the dotted connections. During inference all connections are scaled by $\frac{1}{2}$ to keep the average activation the same.

Perhaps the greatest paradigm shift in CNN design was a result of the skip connection (also known as a residual connection) design pattern as introduced by He *et al.* [39]. If a layer within a CNN is defined as function $\mathcal{H}(x)$, then a skip connection layer is defined as $\mathcal{F}(x) = \mathcal{H}(x) + x$. Adding the input x to the output of a layer $\mathcal{H}(x)$ allows the information within x to propagate deeper within a CNN. A common use of the skip connection design is shown in Equation 2.3.

A common thread of the above approaches is their use of a regularization technique proposed by Srivastava *et al.* [72] called dropout. Dropout attempts to simulate an ensemble of models within a single model by randomly setting a percentage of activations in a CNN to zero during training. During inference the activations are scaled by the drop percentage to keep the average activation strength consistent. Dropout has been shown to be an effective method for increasing the generalization of a single model at the cost of training time. An example of dropout is shown in Figure 2.1.

The standard form of convolution in most CNNs still follows the same design as presented by LeCun *et al.* [52] over 20 years ago. Some attempts have been made to replace this design in niche applications. For domains where computational limitations are important it is common to decompose the convolution into smaller components. One way to decompose a convolutional operation is to spatially factor convolutional kernels into row and column vectors as proposed by Szegedy *et al.* [74]. Another method for decomposing convolutional operations is decomposing along the channel dimension using depthwise separable convolutions as proposed by Chollet [11]. Figure 2.2 shows an example of decomposing convolution.

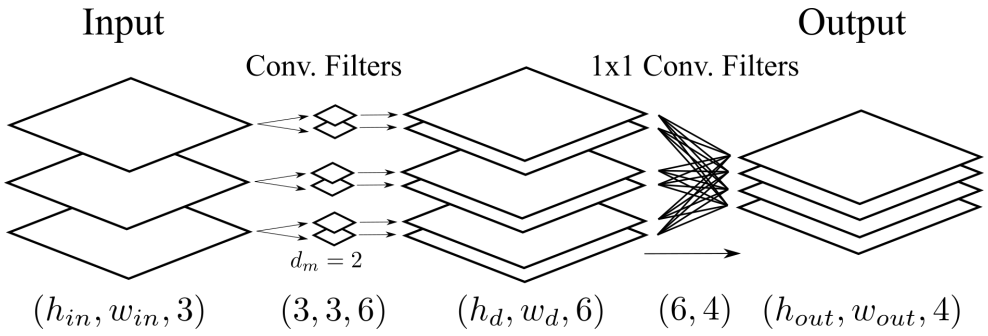


Figure 2.2: An example of a depthwise separable convolution layer. In this example there are two filters, one for spatial convolution of shape $(3, 3, 6)$, and one for inter-channel combination of shape $(6, 4)$. The channel multiplier $d_m = 2$ means that each input channel is convolved with 2 filters. A normal convolution layer with the same input-output channel size would have a single filter of shape $(3, 3, 6, 4)$.

2.3.2 Architectural Design Methods

Designing a CNN can be a time-consuming process due to the number of possible configurations and the time required to train each configuration. One must not only decide on the type of architecture (e.g., CNNs, recurrent neural networks (RNNs) [42], etc.), but the detailed architecture design as well(e.g., number of layers, the type of layers to use, number of neurons per layer, type of activation function to use, etc.). To tackle the bottleneck in network architecture design, recent research efforts have focused on automating the search for optimal network architectures.

There are two general types of automated design approaches for macro-architecture search, a bottom-up approach where one continually adds new operations to a network, and a top-down approach where one starts with an over parameterized model and removes redundant operations. The bottom-up approach can be further divided into evolutionary approaches and reinforcement learning approaches.

Evolutionary approaches follow standard evolutionary based algorithms in which variations on a base architecture are trained, each architecture is measured using a fitness function, and superior architectures are bred with random mutation [73]. The performance of evolutionary methods is heavily tied to the population size, resulting in extreme computational requirements for larger network architectures. Reinforcement approaches, on the other hand, do not directly learn the architecture for a target task, but instead learn a generating network which produces other networks for the target task [87]. Overall, both approaches are prohibitive in practice for practitioners that do not have access to computing clusters.

Top-down approaches use a seed model architecture as a constraint from which to search for new architectures. The most common top-down approaches use an over parameterized model and selectively remove channels. Wen *et al.* [80] added a regularization constraint, called group lasso, during training that slowly decreases the magnitude of multiple groups of kernels. Eventually some kernels are close to zero and can be pruned without consequence. An example of group lasso is shown in Figure 2.3. Shafiee *et al.* [70] takes an evolutionary approach in which every network in a population samples a subset of connections from the seed network, trains to convergence, and inter-breeds. At every iteration the model size of each network in the population progressively decreases.

Instead of only looking at smaller models Gordon *et al.* [31] uses an approach that iteratively shrinks and grows a model. Wong *et al.* [83] proposes a method of fusing successive layers in an architecture instead of relying on pruning to remove a layer from a network thereby allowing for an expedited model reduction and allowing new layer modules to be used.

Unlike automated macro-architecture design, automated micro-architecture design methodologies have not received the same amount of research focus. A recent approach to neuron design is to use an intelligent brute force search scheme in which designs are iteratively tested and improved upon [66]. Other research in this area has focused on extending classic neuron designs by parameterizing the neurons with scaling parameters [38], or by using ensemble methods [35]. These methods are either static or restricted to limited change. To expand the scope of possible learnable neuron designs, piecewise activation functions have also been explored [1, 46, 69]. A common theme to the learnable activation functions is that they combine piecewise coefficients with a combination of non-linear elements.

Often information within a data sample is irrelevant for the task at hand. Attention based models use a specific style of building block that allows a model to more easily focus on a specific subset of features for any given sample. Squeeze and excitation network (SENet) [43] based designs introduce an attention module that allows an easy method of highlighting specific channels of a feature maps using global feature context. With the use of minimal additional computation, the SENet module improves network performance on the ImageNet dataset [14] on all base architectures it is applied too. Several derivative methods, including Bottleneck Attention Module (BAM) [61] and Spatial and Channel-wise CNNs (SCA-CNNs), expand on the SENet module by adding a separate spatial attention mechanism in a parallel and sequential manner, respectively. Attention Condensers (ACs) [82] unify these approaches in a common framework that combines channel attention, and spatial attention into a single module design. The specific parameterization of AC modules is determined using Generative Synthesis [84] based design exploration.

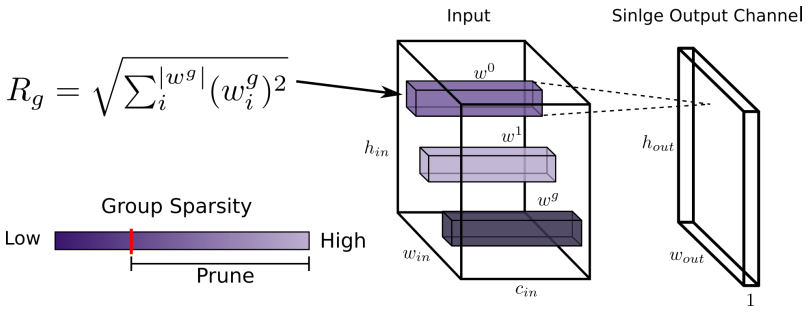


Figure 2.3: An example of group lasso regularization applied to a single layer in a CNN. The group in this example contains all weights used to calculate a single output channel. Each purple rectangular prism is a group. During training each group is sparsified because of having R_g applied to it. After training, groups that are sufficiently sparsified are pruned.

2.3.3 Data Design Methodologies

Without enough data, or without data of sufficient quality, a machine learning model will fail to learn the underlying intrinsic distribution. Techniques like data augmentation implicitly integrates different types of invariances directly into a CNN's model while providing more data points for a model to learn from. For images, common data augmentation techniques include flipping vertically or horizontally, scaling, and randomly cutting out sections from an image [15].

Data augmentation are commonly combined in a stochastic manner. However, it does not always make sense to use each type of augmentation equally across the entire dataset. To alleviate these issues, Cubuk *et al.* [13] proposed a method of dynamically learning desirable augmentations on a per sample basis. Such approaches are still limited to hand designed types of augmentation and significantly increase computational costs during training. Instead of augmenting existing images, many attempts have been made to learn a generative CNN from which one could sample images that follow the same distribution as the base dataset [27].

Augmentation does not only have to be performed on the input data, but can also be performed on the output data. Hinton *et al.* [40] uses a technique called knowledge distillation in which imperfect label predictions of a trained teacher model are used to train another smaller student model. The imperfect labels are not as sparse as the original one-hot labels, but also capture inter-class dependencies. Lin *et al.* [55] extended this work by demonstrating that knowledge distillation can also be used to augment the input data directly. That is, a pre-trained model is used to remove unneeded information from input samples, thereby allowing a student model to learn with improved performance.

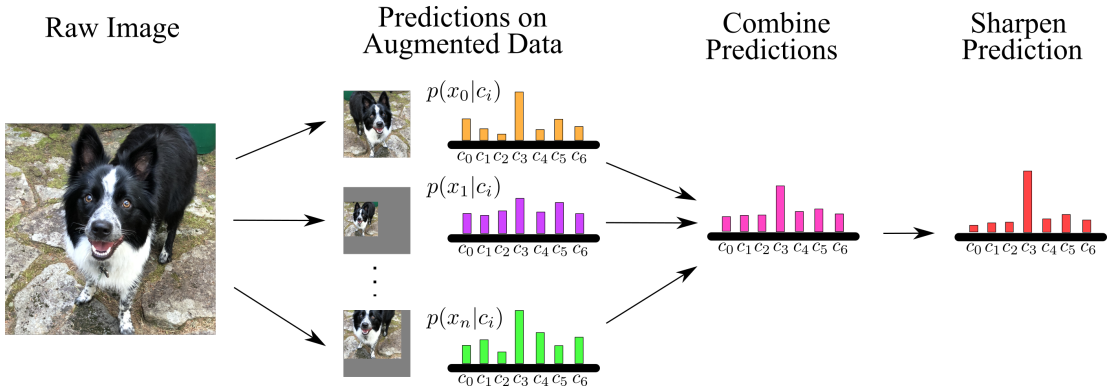


Figure 2.4: An example of using an ensemble of data augmentations to predict an unlabelled sample's true label.

Normally a training dataset is an incomplete representation of the world being modelled; a large part of data augmentation is focused on filling in the missing holes. Often augmentation alone is insufficient, and obtaining more labelled data is required. Using ideas from label smoothing and data augmentation Berthelot *et al.* [6] created a method in which one can label unlabelled data in a semi-supervised manner. By combining multiple augmentations of the same unlabelled sample one can estimate the true underlying label. Figure 2.4 depicts this approach.

The order in which data is presented to a model during training can greatly impact a model's final performance. Bengio *et al.* [5] demonstrated that it can be beneficial to gradually increase the difficulty of samples throughout training using curriculum learning. In a similar approach, Jiang *et al.* [45] uses the loss of a sample as a proxy for difficulty and only performs backprop using difficult samples.

2.4 General CNN Measures

2.4.1 Performance and Heuristic Based Measures

CNNs are generally analyzed with respect to some performance measure. For image classification, typical measures include classification accuracy, precision, recall, F1 score, and Receiver Operating Characteristic (ROC) curves. Such measures are not useful for training during gradient descent as they are not sample independent measures. Loss measures are used to provide a direct proxy method of measuring a model's performance

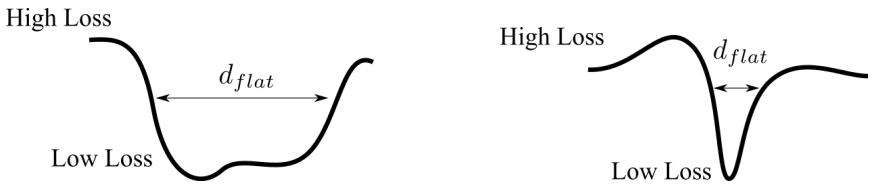
on a single sample. By independently optimizing on a per-sample basis, one indirectly optimizing for a performance measure.

Performance measures are the optimal goal to maximize when designing a model. Unfortunately, they do not generally incorporate aspects of model design. Some heuristic based measures expand the scope of model performance by explicitly including model architecture descriptors, including number of FLOPS, number of parameters, and number of layers in a model. Such measures are easy to calculate but provide limited (and sometimes counterproductive) guidance for improving a model's performance. For example, Nakkiran *et al.* [59] demonstrated uniformly increasing the number of parameters can both increase and decrease a model's performance. To address the issues associated with simplistic measures, Wong [81] proposed a method of combining several measures to determine the tradeoff between different model architectures from a more holistic perspective. These heuristic based measures allow considerations beyond mere decision performance, but are only capable of providing guidance through trial and error.

2.4.2 Loss Surface

An important design consideration of a CNN is the ease with which it can be trained in a tractable manner. A network's loss and the loss's behavior throughout training are important contributing factors to a network's trainability. To better understand these aspects, it is often useful for a practitioner to study the loss surface of a neural network. The loss surface is a scalar field where each point describes the performance of a CNN given a specific model parameterization. The loss surface is of particular interest to CNN practitioners because it is the space in which a learning algorithm must navigate during the training process.

Ideally, the loss surface of a neural network would be convex, thus ensuring a global optimum can be found [26]. Unfortunately, convex loss surfaces are only guaranteed for linear neural networks [26]. For non-linear neural networks (including CNNs) the loss surface is generally highly non-convex [41]. Given the complexity and infinite size of a loss surface researchers have focused on studying a loss surface's local features, including flat minima and sharp minima. Hochreiter *et al.* [41] defines the degree of local flatness as the area of an approximately uniform region around a local minimum below a given threshold in the loss surface. The concept of flatness has been used to describe many phenomena. For example, Hochreiter *et al.* [41] observed that networks that occupy flat regions are shown to correlate with better generalization behavior. Another now well-known example by Kesker *et al.* [47] is that larger batch sizes result in poorer performing models and demonstrated



(a) Flat Loss Surface (b) Sharp Loss Surface

Figure 2.5: An example of two general types of loss surfaces.

that larger batch sizes also cause models to converge to sharp minima. He *et al.* [36] further demonstrated that the convergence to sharp minima is in fact a result of the batch size to learning rate ratio (for specific model designs).

Using loss landscape visualization, one can visually confirm the effect of various components on the loss surface. Goodfellow *et al.* [30] take a visualization approach by uniformly sampling the loss surface between an randomly initialized model and a model after training. They experimentally demonstrated that there exists a smooth and monotonically decreasing path between these two points in the loss surface. Furthermore, it is observed that the loss surface directly between two samples can contain parameter solutions with high loss. Li *et al.* [53] extended the work of Goodfellow *et al.* [30] by using normalized basis vectors to project the loss landscape to a 2D surface. Through visualization they demonstrated that both skip connections and small batch sizes produce smoother surfaces for ResNet models. In addition, Li *et al.* [53] demonstrate that as a model increases in depth the loss surface transitions from a mostly convex shape to a chaotic space filled with convex pockets. The transition between these two states is marked by a significant drop in generalization performance.

2.4.3 Intra-layer and Inter-layer Comparison Methods

The methods discussed so far have primarily been focused on the global nature of a CNN. FLOPS and parameters are generally counted at a network level. Loss surface is investigated from a specific weight instantiation. Methods that study CNN models at a finer resolution are primarily focused on measuring the properties of data as it moves from a network's input to a network's output. One type of method studies how the performance of a CNN progressively changes throughout a CNN. Alain *et al.* [2] proposes a method called linear probing in which a linear classifier is used to measure the classification performance of a CNN at each layer in the model. This style of method provides a rough estimate on which

parts of a model are most impactful to the overall task at hand. The features used to calculate linear separability have also been used for more non-linear kNN based methods. Methods such as Hierarchical Nucleation [17] and methods proposed throughout this work use kNNs to study how the latent representations of samples gradually cluster with like samples as the layers get progressively deeper. Papernot *et al.* [60] use kNNs to study a model's confidence, credibility, and robustness.

Another type of method studies the intrinsic dimensionality (ID) of the data within the layers of a network. ID metrics effectively serve as a measure of complexity; the higher the ID the more relevant degrees of freedom the manifold has. There are many ways to measure the ID of a set of data. One approach is based on the study of dominate eigenvalues in a local region [22], while another approach uses topological methods that focus on calculating the density of data within a local region [20, 63].

Gong *et al.* [25] uses a topological approach to estimate the ID of a given dataset by comparing nearest neighbour geodesic distances of a dataset to that of an m-dimensional hypersphere with samples following a Gaussian distribution. Since the true ID of data is often unknown, Gong *et al.* [25] validate their approach using a k-NN classifier on the intrinsic data representation. Unfortunately, knowing the intrinsic dimension of a dataset is only broadly useful for bounding the size of a CNN required for a given task. A more useful approach for CNN analysis is investigating the ID of dataset representation throughout a network (i.e., at each layer). Ansuini *et al.* [3] demonstrate that the ID of a dataset in a classification model across a variety of architectures generally follows a hunchback pattern (i.e., starts off low, quickly spikes, and gradually decreases towards the output). Another interesting find from this approach is that classification error negatively correlates with the ID of a dataset at the final layer of a CNN.

Intrinsic dimensionality methods are focused on performing analysis on a local scale. Other approaches such as topological data analysis (TDA) study the structure of data across all ranges of localities [7]. TDA uses a structure called a simplicial complex (think sets of high dimensional triangles) to measure how the relation between sets of data changes as the local region under consideration expands. To demonstrate the efficacy of TDA for CNNs Carlsson *et al.* [8] studied the structure of all 3×3 convolutional filters for each layer in a network across a variety of models (instead of the data at each layer directly). Through qualitative and quantitative analysis Carlsson showed that each layer in a CNN displayed a distinct internal structure.

Despite providing a method of looking at the structure of a network across layers, TDA is still a relatively new field of study and has had limited adoption by the deep learning community. Statistical measures, in contrast, have been used in deep learning since its

inception to study many problems, including inter-layer similarity. Dot product-based methods are the basis for most statistical measures that compare a dataset's representation between layers

$$s_{dot}(\mathbf{X}, \mathbf{Y}) = \| \mathbf{Y}^T \mathbf{X} \|^2 \tag{2.4}$$

where s_{dot} is the similarity, $\mathbf{X} \subset \mathbb{R}^{n \times d_0}$ and $\mathbf{Y} \subset \mathbb{R}^{n \times d_1}$ are assumed to have zero mean along the feature dimension, n is the number of samples, and d_0 and d_1 are number of features for set $\mathbf{X}$ and $\mathbf{Y}$, respectively. Here the i^{th} row in $\mathbf{X}$ and $\mathbf{Y}$ are vectorized layer outputs for the x^{th} and y^{th} layer of a network, respectively. If one normalizes Equation 2.4 by dividing by $(n-1)^2$ one gets the linear case of the Hilbert-Schmidt Independence Criterion [32]

$$s_{linear\ HSIC}(\mathbf{X}, \mathbf{Y}) = \frac{\| \mathbf{Y}^T \mathbf{X} \|^2}{(n-1)^2} = \| cov(\mathbf{X}^T, \mathbf{Y}^T) \|^2 \tag{2.5}$$

An attractive property of the similarity measures in Equations 2.4 and 2.5 is that they are invariant to orthogonal transforms on either set of data. In other words s_{dot} and $s_{linear\ HSIC}$ will produce the same similarity even if one or both sets of data are rotated in some higher dimension. On the other hand, a problem with such transforms is that they are not invariant to isotropic scaling, nor to invertible linear transforms. From a CNN perspective, both types of transforms have no impact on the CNN's operational structure. That is, one can easily insert both transforms followed by their inverse anywhere in a CNN without affecting its performance.

A measure that has all three properties is a measure called Canonical Correlation Analysis (CCA) [34]. At a high level CCA works by finding a pair of linear transforms such that $\mathbf{X}$ and $\mathbf{Y}$ maximally correlate with one another. Note that solving for the transforms is subject to orthogonality requirements. To achieve a scalar number one then takes the mean of the correlation coefficients of the transformed sets of data. The CCA measure can be defined as

$$s_{CCA}(\mathbf{X}, \mathbf{Y}) = \frac{tr(\widetilde{\mathbf{Y}}^T \widetilde{\mathbf{X}})}{\min(d_0, d_1)} \tag{2.6}$$

where $\widetilde{\mathbf{X}}$ and $\widetilde{\mathbf{Y}}$ are zero meaned transformed sets of data, and $tr(\cdot)$ is the matrix trace operator. A problem with CCA is that it doesn't distinguish between noisy features and important features. To solve this issue Raghu *et al.* [65] proposed a method call Singular Vector CCA (SVCCA) in which SVD is applied to both $\mathbf{X}$ and $\mathbf{Y}$, such that 99% of the variance is accounted for, prior to applying CCA. Like CCA, SVCCA is invariant to invertible linear transforms, orthogonal transforms, and isotropic transforms.

The statistical approaches discussed so far argue that invariance to invertible linear transforms is an important feature to have. However, from a topological perspective

invertible linear transforms can completely change properties of the manifold that the data sits in. Kornblith *et al.* [49] propose an approach called Centered Kernel Alignment (CKA) to solve this issue. To remove invariance to invertible linear transforms, CKA normalizes the HSIC similarity between two sets of data by the HSIC similarities between each set of data independently. The linear case of CKA is defined as

$$s_{linear\ CKA}(\mathbf{X}, \mathbf{Y}) = \frac{s_{linear\ HSIC}(\mathbf{X}, \mathbf{Y})}{\sqrt{s_{linear\ HSIC}(\mathbf{X}, \mathbf{X}) s_{linear\ HSIC}(\mathbf{Y}, \mathbf{Y})}} = \frac{\| \mathbf{Y}^T \mathbf{X} \|_F^2}{\| \mathbf{X}^T \mathbf{X} \|_F \| \mathbf{Y}^T \mathbf{Y} \|_F} \quad (2.7)$$

Of the statistical approaches discussed, CKA demonstrates the greatest ability to detect similarly between layers in a CNN [49].

Chapter 3

Representational Response Analysis

Introduced in Chapter 1, the Representational Response (RR) of a CNN captures a CNN's behaviour as it operates on a set of data. By definition the RR contains all discrete latent representations that a CNN produces in the form of a set of manifolds for each function in a CNN. In its raw form the wealth of information contained within the RR presents a practical challenge as the sheer volume of data produced by even a shallow CNN operating on a small dataset (in both number of samples and dimensionality of data) can result in gigabytes of information to process. This volume of data limits a researcher's ability to directly interface with the RR and to understand a CNN's behaviour. Thus, distilling the information contained within a CNN's RR is essential for making use of this information.

Many of the existing analytic methods discussed in Chapters 1 and 2 have already been designed to extract targeted information from a CNN's RR. In this chapter we first discuss how these existing methods extract information from a CNN's RR with a focus on limitations when using each of these methods. From this comparison we identify important analytic qualities to include in a generalizable analytic framework for studying CNNs. We then propose a method of modelling the RR of a CNN using these qualities as a guide. Finally, we propose a set of three quantitative metrics, derived from the underlying RR model, which we use throughout this work to study the progression of complexity, similarity, and performance throughout a CNN.

3.1 RRA Framework Scope

The information contained within a CNN's RR can be used to study a variety of key aspects of a CNN's behaviour. The following are broad categories of analysis:

1. **Sequential analysis**: Investigating the change in local or global structure of a RR's manifolds across a CNN (i.e., from input to output of a CNN).
2. **Comparative analysis**: Comparing two discrete RRs to one another. For example, investigating the effects of changing a CNN's architecture.
3. **Training analysis**: Comparing the RR of a given CNN throughout its training process and looking for discrete state changes in the properties of the RR.
4. **Data sensitivity analysis**: Investigating the effect of altering the data distribution a CNN is operating on.
5. **Adversarial analysis**: Looking for how a CNN responds to adversarial attacks.

This list of analytic methods is extensive and highlights why comprehensively studying a CNN is so difficult. In this work we focus on using **sequential** and **comparative** analysis techniques.

3.1.1 Existing RRA Metrics

Many existing deep learning analytic metrics fit neatly into the proposed RR formulation. However, these metrics have practical limitations which prevent their wide scale adoption, including: knowing which metric to use is not necessarily clear, such metrics are generally computationally expensive to compute, the computation of such metrics are disjoint, and there is limited research on using these RR metrics in unison. Below we expand on a selection of existing RR metrics, which were already presented in Chapter 2, to better contextualize benefits and limitations of each metric as relevant to this research.

Intrinsic Dimensionality (ID) methods [3, 20] are an example of sequential RRA. The manifold of each layer in a CNN has its dimensionality independently estimated by aggregating pairwise nearest neighbour distance statistics. Using the ID at each layer one can see the change in ID as the depth of the layer's manifold increases. Knowing the ID of the manifold and how it changes when a CNN's design is changed can illuminat interesting characteristics of a model, like how the manifold of a CNN changes when the brightness of all data used with a CNN is increased [3]. Lacking from ID analysis is any concept of distance between ID measures of different layers. One is left to draw conclusions from the general trend in the ID.

Layer Similarity (LS) measures [32, 34, 49, 65] are examples of comparative RRA. These metrics are designed to directly compare information representations and allows one to gain

direct insight into differences between layers in a CNN in the form of a bounded metric; often between 0 (not similar) and 1 (the same). An important difference between existing LS metrics and ID metrics is that ID metrics use only local information (i.e., comparing nearest neighbours) while LS measures use global information (i.e., all features are compared to every other feature). LS measures are able to use global information since they only measure zeroth degree dependencies. Using global information for higher degree dependencies quickly becomes computationally infeasible. In addition, using global information is not always necessary (e.g., comparing distance between all points on a spiral-like manifold).

Linear Probing (LP) [2] provides both a sequential analysis and comparative analysis component. Linear probing calculates an accuracy score (via a linear classifier) independent of other layers, where the score is normalized and human interpretable allowing a researcher to directly compare accuracy scores between CNN layers. The tradeoff with LP is that it requires a unique linear classifier to be learned for each manifold in a CNN. These auxiliary models can easily be larger than the CNN itself.

Topological Data Analysis (TDA) [7] is another dual-purpose metric providing both a sequential analysis and comparative analysis component. The topological barcodes generated for each stage provide insight into each manifold's topological features, and comparing the barcodes allows the similarity between layers to be measured. An attractive aspect of TDA is that it can be used to compare disjoint manifolds (e.g., manifolds of two different datasets). However, computing these barcodes is $\mathcal{O}(n^3)$ making it computationally prohibitive for larger datasets. In addition, there is limited research into how TDA can be used to guide CNN design choices.

3.2 RRA Framework

One of the goals in this work is to propose a useful framework for analyzing classification based CNNs. As such, the framework must include a variety of metric types that are collectively easy to compute (relatively speaking). Throughout this work we focus on three forms of metrics:

1. A metric that measures the **complexity** of a layer's manifold,
2. A metric that measures the **similarity** between manifolds (either within a single CNN or between CNNs), and
3. A metric that quantifies the classification **performance** (or a proxy) within each manifold.

A complexity measure is used to provide an indication on the scope of information a CNN is computing (e.g., what layers in a CNN are responsible for distinguishing a larger variety of patterns). A similarity measure is used to provide both an intra-model method for layer redundancy detector (e.g., are layers at different points in a CNN computing the same patterns), and an inter-model method of similarity (e.g., do two distinct CNNs have the same progression of pattern detection). Finally, a performance metric is used to better understand which points in a model provide the best absolute classification performance, and which layers contribute the most to relative performance improvements.

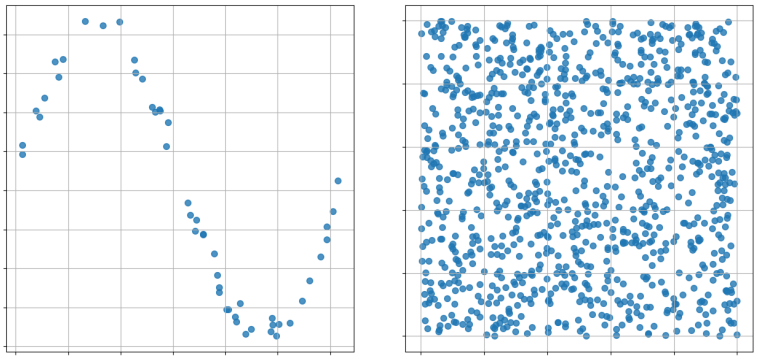
The list of metrics discussed in Chapter 3.1 individually covers either the complexity, similarity, or performance metric types. However, all these methods are expensive to compute individually, and many are calculated disjointly, further increasing the computational burden. To reduce the computational requirements we focus on developing a set of metrics that can all be derived from the same computational model of a layer's manifold.

3.2.1 Manifold Approximation via Neighbour Graphs

Until now we have only discussed the analysis of a CNN's RR in vague terms around using computational models constructed from latent embeddings within a CNN to quantify targeted characteristics of interest. Before introducing the specific methods used in this work, we first elaborate on how we transform a set of latent embeddings into a common model of the CNN's manifolds.

The choice of the underlying model is dependent on the target metrics. While within this work we focus on the use of three metrics, it is important not to limit the proposed RRA framework to obscure metrics, but instead have the framework be generalizable. Looking at the metrics discussed in Chapter 3.1 one common underlying computational requirement is the need to calculate the distance between local neighbours in each of the latent embeddings. For example, the ID metric requires knowing the distance between the two nearest neighbours, and the TDA metric and CKA [49] metric (a type of LS metric) require calculating the distance between all samples.

Given the common requirement of needing to calculate the distance between samples, we base the proposed RRA framework on the concept of nearest neighbour graphs. More formally, let $\mathbf{Y}_v = v_v(\mathbf{X}; G_v)$ be a one-to-one mapping of samples from the input space to the space of layer v_v of a CNN. For a given layer v_v with a set of outputs $\mathbf{Y}_v$, let $H_v = (\mathbf{Y}_v, D_v)$ be the graph of neighbours for layer v_v, where $\mathbf{Y}_v$ are the vertices of the graph, and D_v are the set of all edges between pairs of samples $\mathbf{y}_{vi}, \mathbf{y}_{vj} \in \mathbf{Y}_v$. Let $K_{vi} \subseteq \mathbf{Y}_v$ be an ordered set of nearest neighbours of sample $\mathbf{y}_{vi}$. The distance between two points



(a) 1D manifold in a 2D space (b) 2D manifold in a 2D space

Figure 3.1: Examples of using the TwoNN estimator to approximate the intrinsic dimensionality of each respective manifold. Each image shows a set of points on a manifold with noise added. The left example is of a 1D sine wave in a 2D space. The right example is of a 2D uniform plane in 2D space. The TwoNN estimator predicts an ID of 1, and 2, respectively, for the two distributions.

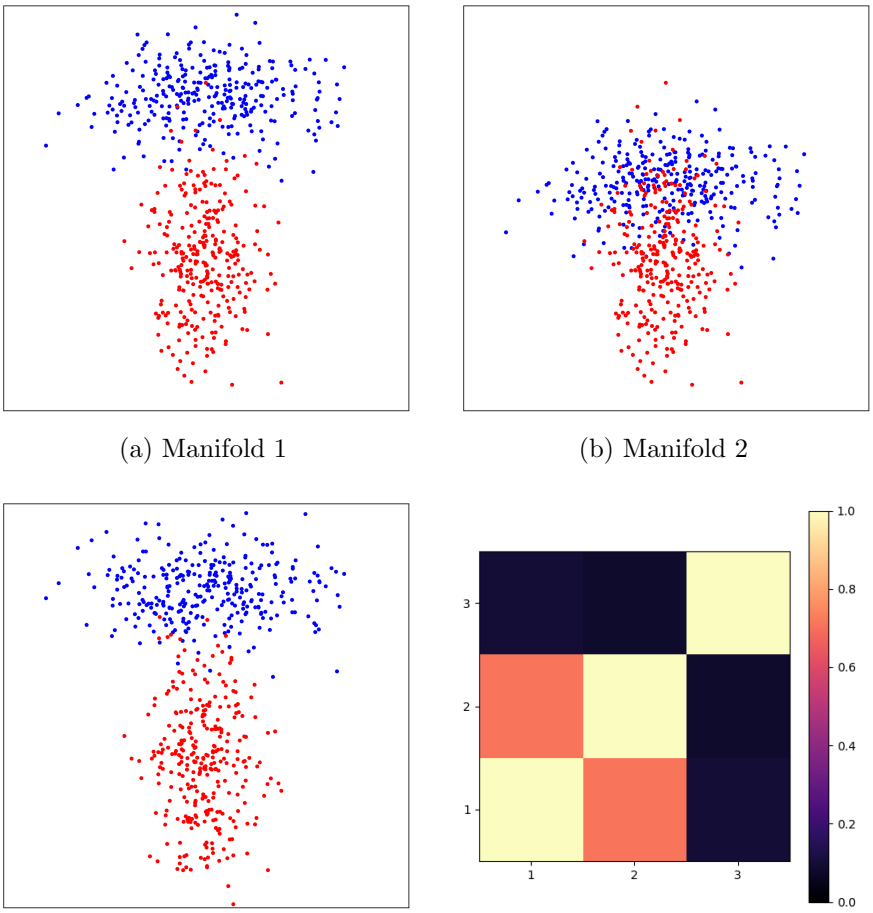
$\mathbf{y}_{vi}$ and $\mathbf{y}_{vj}$ in the ambient space $\mathbb{R}^v$ (i.e., the feature space of layer v_v) is measured by the distance metric $R(\mathbf{y}_{vi}, \mathbf{y}_{vj}) = r^v_{ij}$. In this work we use the Euclidean distance for R. Design considerations around K_{vi} are explored further in Chapter 3.3. Through the remainder of this section, we provide specifics on the complexity, similarity, and performance metrics used for CNN analysis in this work.

3.2.2 A Complexity Metric Through Intrinsic Dimensionality

The purpose of the complexity metric within the framework is to provide a measure for the variety of unique features that a CNN is processing at any given point. A complexity metric like VC dimensionality [77] provides a bound on potential complexity that can be learned but does not necessarily reflect on how the capacity within a layer of a CNN is being used. Naively counting the number of weights of a given layer of a CNN produces a similar result. In this work we use the TwoNN estimator proposed by Facco *et al.* [20] and used by Ansuini *et al.* [3] to study the ID progression of various CNN architectures. The TwoNN ID estimator is defined as

$$ID = \arg\max_{ID} ID^N \prod_i^N \left[\frac{r_i^2}{r_i^1}\right]^{-(ID+1)} \tag{3.1}$$

where r_i^1 and r_i^2 are the distances between the first and second closest nearest neighbour of the i^{th} sample, and ID is the intrinsic dimensionality of the dataset. Figure 3.1 shows examples of using the ID estimator.



(a) Manifold 1

(b) Manifold 2

(c) Manifold 3

(d) NNLS matrix

Figure 3.2: An example of using the NNLS metric to compare three different manifolds. Each set of data has a red class and a blue class. These classes are Gaussian distributions. Manifold 1 has the red and blue class mostly separated. Manifold 2 slides the blue class down so that it has a greater intersection with the red class. Manifold 3 uses the same global class distributions of Manifold 1 but with the samples shuffled within the classes. The bottom right image contains the 3×3 symmetric NNLS matrix. Notice the high degree of similarity between Manifolds 1 and 2, but the dissimilarity between Manifold 3 and other manifolds.

3.2.3 A Similarity Metric Through Nearest Neighbour Layer Similarity

For the similarity measure we are looking for the ability to compare manifolds within a CNN, and manifolds between CNNs. One prominent layer similarity metric CKA [49] provides both of these abilities. CKA comes in two primary variants: RBF CKA and Linear CKA. RBF CKA requires that the distance between every sample is known and stored for the final similarity metric to be calculated; for modern datasets an $n \times n$ distance matrix can quickly overwhelm a computer's memory capacity. However, Linear CKA allows one to avoid direct distance calculation through algebraic simplification. Much like CKA, all other CNN layer similarity metrics also demonstrate similar incompatibilities with the proposed kNN model in that all inter-sample distances are required and not just a subset.

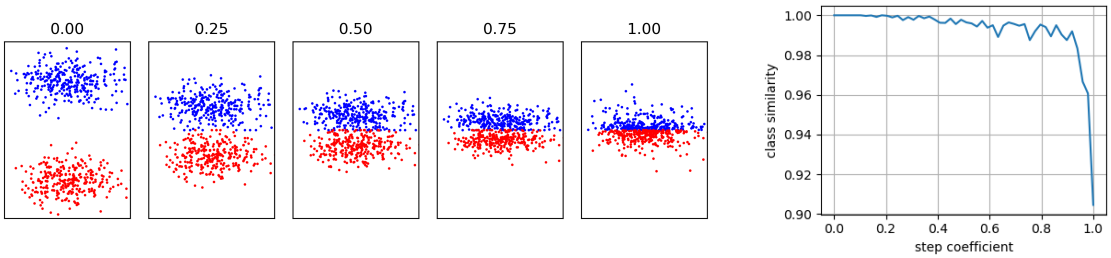
Now we develop a novel similarity metric designed to be directly compatible with the underlying kNN-based manifold model. Let $Q(H_a, H_b) = q_{ab}$ measure the inter-layer similarity between layers v_a and v_b using their respective neighbour graphs H_a and H_b. To compare a single sample across layers we propose a sample-wise similarity function $Q_s(K_{ai}, K_{bi})$ where K_{ai} and K_{bi} are sample $\mathbf{x}_i$'s nearest neighbours in layers a and b, respectively. We bound an individual sample's inter-layer similarity to the range of 0.0 to 1.0 to ensure that samples are equally represented. Let $Q(H_a, H_b)$ be defined as

$$Q(H_a, H_b) = \frac{1}{n} \sum_{i}^{n} Q_s(K_{ai}, K_{bi}) \tag{3.2}$$

Using K_{ai} and K_{bi} one can calculate a variety of useful statistical measures, including the mean sample, and the deviation from the core sample in each layer $\mathbf{y}_{ai}$ and $\mathbf{y}_{bi}$, respectively. However, the neighbours for a given sample may not be the same between layers (for some $k << n$), this makes it difficult to normalize and compare intra-layer measures. To avoid this issue, we directly measure the difference of the neighbours between layers through intersection-over-union. Let the per-sample inter-layer similarity function be defined as

$$Q_s(\mathbf{H}_{ai}, \mathbf{H}_{bi}) = \frac{|K_{ai} \cap K_{bi}|}{|K_{ai} \cup K_{bi}|} \tag{3.3}$$

Note that $Q_s(\mathbf{H}_{ai}, \mathbf{H}_{ai}) = 1$. Due to the formulation of $Q_s(\cdot)$, $Q(\cdot)$ can be framed as the mean IOU (mIOU) of nearest neighbours between layers. Let the layer similarity metric Q be called Nearest Neighbour Layer Similarity (NNLS). Figure 3.2 shows an example of how NNLS is used.



(a) Sample CS Transitions

(b) CS Transition Curve

Figure 3.3: A toy example of how the NNCS is affected as the distance between two linearly separable class clusters changes. Figure 3.3a shows the transition as two linearly separable classes approach one another. On the left at the transition state of 0.0 there is a clear separation between the clusters. As the transition state approaches 1.0 the average intra-class distance between points begins to outweigh the average inter-class distance. Note that at a transition state of 1.0 the two classes are still linearly separable along the x-axis. Figure 3.3b shows the observed NNCS for the toy dataset as the transition state moves from 0.0 to 1.0. When the transition state is 0.0 the NNCS equals 1.0. By the time the transition state reaches 1.0 the NNCS has reduced to almost 0.9.

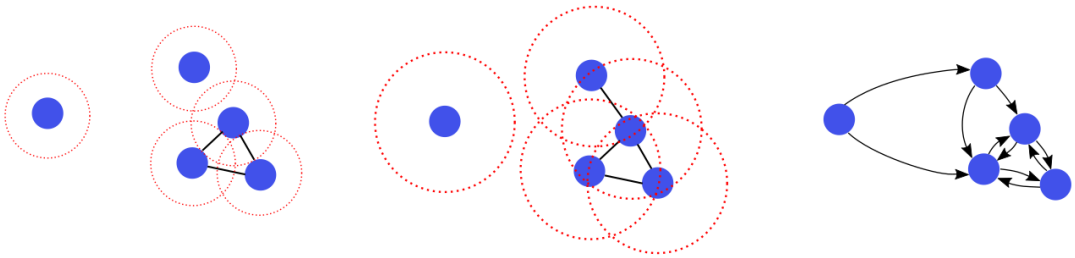
3.2.4 A Performance Metric Through Nearest Neighbour Class Similarity

For the performance metric we are interested in measuring how the classification ability of a CNN changes across it. Linear probing [2] is directly designed to achieve this. However, LP does not use a kNN, but instead requires an auxiliary linear model to be trained. The auxiliary model requires an additional $f \times c$ parameters, where f is the number of features of a given layer and c is the number of classes. Instead of using a linear classifier to measure global class separability, we propose using a kNN based strategy to measure the change in local class separability.

For any given kNN of layer i in a CNN, let the class similarity score $NNCS_i$ be defined by

$$NNCS_i = NNCS(G_i, Z) = \frac{1}{MN} \sum_n \sum_m \mathbb{1}(Z_n, Z_m) \tag{3.4}$$

where $\mathbb{1}(\cdot)$ is an indicator function, and Z_n and Z_m are the ground truth classes for the n^{th} sample in N and the m^{th} nearest neighbour of the n^{th} sample, respectively. Figure 3.3 shows an example of class similarity.



(a) Smaller fixed radius (b) Larger fixed radius (c) Directional

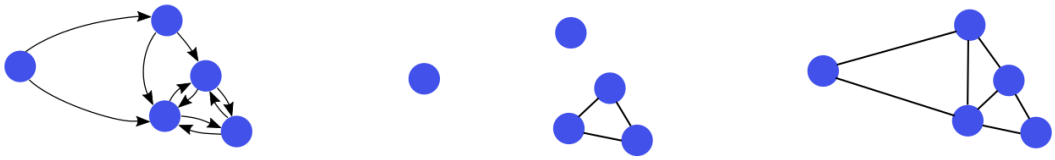
Figure 3.4: Examples of nearest neighbour graphs generated using three different strategies. Left shows a nearest neighbour generated using a fixed small radius. Center shows a nearest neighbour generated using a fixed large radius. Right shows a nearest neighbour generated using a fixed number of neighbours.

3.3 Considerations for Approximating a Manifold

In this work we represent the data manifold using a graph of relations between samples. The effect of how we design the graph of relationships has a potential to change what we can see as we investigate properties of a CNN's RR. Below we discuss the implications of specific design decisions and what practical considerations must be used to create a useful and computationally tractable RRA framework.

3.3.1 Neighbouring Regions

The goal of the kNN graph is to represent localized information from the samples. As such, a metric for measuring distance between two samples in each layer is required. In general, there are two common methods used. The first approach uses a distance threshold to find all samples $\mathbf{y}_{vj} \in K_{vi}$ that are within some fixed radius r_v of sample $\mathbf{y}_{vi}$, where r_v is constant for the entire graph. Using this approach the set K_{vi} for a single layer v_v can contain a variable number of neighbours. The second approach uses a variable radius but with a fixed number of samples k in K_{vi} for each sample $\mathbf{y}_{vi}$. Such an approach is called a k nearest neighbour (kNN) graph. For this work a kNN based approach is used to ensure that each sample $\mathbf{y}_{vj} \in \mathbf{Y}_v$ has a neighbour (i.e., $|K_{vi}| > 0$). Note that it would be possible to find the smallest radius such that every sample has at least one neighbour, but this would also allow for an unlimited number of neighbours (e.g., when there is one extreme outlier). Figure 3.4 shows examples of neighbours graphs generated from the two graph generation methods.



(a) Directional (b) Intersection (c) Union

Figure 3.5: An example of the neighbour graphs generated using three different neighbour selection criteria. Left shows a neighbour graph generated using directional connections. Center shows a neighbour graph generated using an intersection approach. Right shows a neighbour graph generated using a union approach.

3.3.2 Directionality

To build a kNN graph one must choose if connections are directed or un-directed, what distance metric to use, and the number of neighbours. For an un-directed kNN graph, if sample $\mathbf{y}_{vi}$ is a neighbour of $\mathbf{y}_{vj}$, then $\mathbf{y}_{vj}$ must also be a neighbour of $\mathbf{y}_{vi}$. However, the un-directed nature of connections would require a loosening of the fixed number of neighbours inherent to kNN graphs as a kNN graph with k un-directed edges per sample may not exist.

One way to loosen the neighbourhood criteria is to perform an intersection such that two samples are un-directed neighbours iff both samples are directed neighbours of each other; this effectively sets an upper bound to the number of neighbours to k. Such an approach undermines the choice of a kNN graph in that some samples might not have neighbours. Another way to solve the issue is to perform a union where by two samples are un-directed neighbours iff either sample is a directed neighbour of one another; effectively setting k as the lower bound to the number of neighbours. This approach can result in some samples having orders of magnitude more neighbours than other samples. A third option to loosen the neighbourhood criteria is to just use directed edges, thereby ensuring every sample has the same number of neighbours. In this work directed edges are used for nearest neighbour graph construction. Figure 3.5 shows kNNs generated using each of the three neighbour criteria.

3.4 Summary

In this chapter we introduced the core focus of this work, a RRA framework that emphasizes the use of complementary RR metrics to extract useful information about a CNN's behaviour. The proposed framework consists of first modelling the manifolds of a CNN using kNN

Parameterization	kNN Calculation	ID	NNCS	NNLS
$N_{samples}$	$\mathcal{O}(s^2)$	$\mathcal{O}(s)$	$\mathcal{O}(s)$	$\mathcal{O}(s)$
$N_{neighbours}$	$\mathcal{O}(n \log n)$	$\mathcal{O}(1)$	$\mathcal{O}(n)$	$\mathcal{O}(n)$
N_{layers}	$\mathcal{O}(l)$	$\mathcal{O}(l)$	$\mathcal{O}(l)$	$\mathcal{O}(l^2)$
Data Dimensionality	$\mathcal{O}(d)$			

Table 3.1: The computational runtime of the RRA framework. For a typically use case the kNN calculation is the most expensive computational operation by a large margin, and is primarily gated by the number of samples for large dataset, or the data dimensionality.

graphs, and then analyzing the kNN graphs using a set of three metrics. These metrics focus on 1) the complexity of each manifold using the TwoNN ID estimator, 2) the similarity between layer manifolds using NNLS, and 3) the performance of each layer using NNCS. To the best of our knowledge, this is the first proposed CNN-focused analytic framework which tries to unify a heterogeneous set of RR metrics. The proposed framework is designed to front load the majority of the computation into modeling each of the manifolds, and then running the proposed RRA metrics directly on the kNN graphs. This order of operations is designed to allow for a more seamless CNN analysis when compared to using similar metrics but which are calculated in a disjoint manner. Table 3.1 includes the computational runtime of a the proposed RRA framework.

Chapter 4

Applying Representational Response Analysis

Chapter 3 formulated both representational response analysis and the three metrics that will be used throughout this work. We now apply RRA to study the behavior of ResNet20 [39] and its latent representations as data travels through the model under various conditions. The focus of this chapter is to demonstrate the utility of RRA. We do this by studying the effects that applying data augmentation to a dataset during the training process has on the resulting latent structure of a CNN.

Applying augmentations during a CNN's training process is a standard technique used to prevent overfitting on the training dataset and to improve a model's generalization. If the augmentations are chosen correctly, they typically improve a model's performance on unseen data. When new augmentations are introduced, standard practice is to use the augmentations on a variety of model-dataset combinations and perform a series of hyper-parameter ablation tests. However, these tests often focus on measuring model performance, and perhaps a metric for which the new augmentations were designed to improve (e.g., robustness). Often the internal representational workings of a model are ignored. In this chapter we demonstrate the utility of RRA by investigating how a model's internal representational behaviour changes when augmentations are used during the training process.

Table 4.1: ResNet20-CIFAR10 Layer Feature Map Size

	Input	Neck	Stage 1			Stage 2			Stage 3			GAP	Classifier
Layer	0	1	2	3	4	5	6	7	8	9	10	11	12
channels	3	16	16			32			64			64	10
h, w^*	32	32	32			16			8				
features	3072	16384	16384			8192			4096			64	10

*h and w are the height and width, respectively

Table 4.2: ResNet20-CIFAR10 Training Augmentations

Augmentation	Setting
Translate	4 pixels
Horizontal Flip	50%
Random Rotation	±15 degrees
Brightness*	0.4 @ 80%
Contrast*	0.4 @ 80%
Saturation*	0.2 @ 80%
Hue*	0.1 @ 80%
Grayscale	20%
Gaussian Blur	(0.1, 2.0)

*the coefficients correspond to PyTorch [62] augmentation hyperparameters

4.1 Experimental Setup and Results

In this experiment we compare a ResNet20 model trained on two different versions of the CIFAR10 dataset. The first model (model A) is trained on the CIFAR10 dataset without using data augmentation, and the second model (model B) is trained on the CIFAR10 dataset with random data augmentations applied throughout training (See Table 4.2 for a complete list of augmentations). Both model variations are trained 8 separate times (using different seeds for each run). The results in this chapter (and throughout this work) are the averaged between these 8 runs unless otherwise noted. For each model type we investigate the RR for both the training partition and the validation partition. No augmentations are applied during the actual RRA.

The number of layers and specifics on each layers' feature map size is shown in Table 4.1. The following is the training procedure used on both model A and model B: the optimizer is stochastic gradient descent with a momentum rate of 0.9, the base learning is 0.01, with a cosine decay rate that ends at 1% the initial learning rate, training for 100 epochs, and a linear warm up period of 10 epochs.

Table 4.3: ResNet20-CIFAR10 Training Results

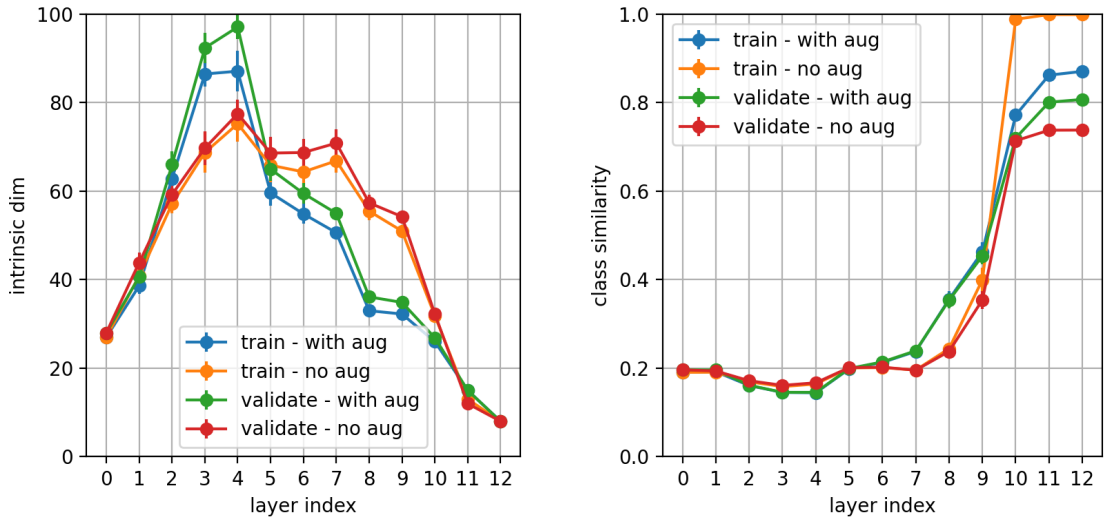
Model	Training Acc. (%)	Validation Acc. (%)
Model A (No Aug.)	1.0 ± 0.0	81.0 ± 0.46
Model B (With Aug.)	95.6 ± 0.2	89.8 ± 0.2

Table 4.4: ResNet20-CIFAR10 Augmentation Ablation Results

Augmentation	No Aug.	Translate	Rotation	H-Flip	Colour Jitter	Grayscale	Blur
Mean Acc.	81.0	88.0	87.0	85.1	82.0	80.6	78.8
Std.	0.46	0.21	0.33	0.37	0.40	0.44	0.34

The specific parameterization of the kNN used to model the manifolds will have consequences for the measure quantities of the three RR metrics. Such parameters include: directionality of connections, the number of data samples used to model the manifold, and the number of neighbours per sample. The consequences of these considerations are explored in Chapter 3.3 for directionality, and Chapter 5. In this section we limit our investigation to a single parameterization of the kNNs to a class balanced random selection of 15000 samples with each sample having 150 neighbours for the training partition, and 10000 samples (i.e., the entire partition) with each sample having 100 neighbours for the validation partition; note that this parameterizations keeps the ratio between the number of samples and the number of neighbours consistent between the two partitions (the importance of keeping this ratio consistent is explored further in Chapter 5).

The training accuracy results for the two models on both of the data partitions are shown in Figure 4.3. Unsurprisingly, model B has a higher performance on the validation set (i.e., data not seen during training) by 6.7% while also overfitting less. In a typical deep learning paper proposing a novel set of data augmentations (or some other model improvement) one might perform an ablation test on how the different augmentations impacted performance results, leaving an investigation on the internal workings of the model narrowly addressed, or left to be addressed in *future work*. Throughout the remainder of this chapter, we investigate how training with and without data augmentations manifests itself in the underlying RR of each model. For the sake of completeness, the results of an augmentation ablation test are shown in Table 4.4. Note that *Colour Jitter* contains all four colour augmentations in Table 4.2. In the remainder of this chapter we look at the incites that each of the three RRA metrics provide. We separate NNLS into two comparisons: one that compares layers within a model, and one that compares layers between models.



(a) ResNet20-CIFAR10 ID Comparison

(b) ResNet20-CIFAR10 NNCS Comparison

Figure 4.1: Example of applying the RRA metrics to ResNet20 CNNs trained on the CIFAR10 dataset with and without data augmentation applied during training. The left image shows the ID curves, and the right image shows NNCS curves. Each figure contains curves for both the CIFAR10 training partition and validation partition.

4.2 Intrinsic Dimensionality

The ID training and validation curves for model A (w/o augmentation during training) and model B (w/ augmentation during training) are shown in Figure 4.1a. Overall, there are a few key patterns that stand out. The first noticeable pattern is that the training and validation ID curves for a given model closely follow one another. For most layers, regardless of the model, the validation ID is $\sim$5% above the training ID curves. While interesting, the validation ID curve being consistently higher than the training ID curve could be a numerical artifact from our choice of hyper-parameters for RRA and will need further examination. For most layers, the variation is near 0.

The similarity between the training and validation curves holds true for both model A and model B despite the differences in classification performance; model A overfits by 19% and model B overfits by 5.8%. The difference in performance between the training partition and validation partition means that neither model fully generalize (i.e., they overfit), however the similarity between the training and validation ID curves indicates that each of the models is not inherently processing the two partitions of data differently. Thus, implying that the distributions of the training partition and the validation partition are sufficiently like one another as seen by the model. We further validate this observation by progressively adding Gaussian noise to the training partition until the partition consists of

only images of random noise. The resulting ID curves progressively grow larger, loses its characteristic shape, and diverges from the validation ID curve. The important take away is that if the training ID curve and validation ID curve differ significantly then there is likely data drift between the partitions (or the model has learned a pathological behaviour during training).

Looking closer at the stage structure of the ResNet20 model (shown in Table 4.1), one can see that many of the major changes in ID are associated with a transition between the stages. Both models peak at layer 4 (the end of stage 1), and experience large drops in ID between layers 7 and 8 (the transition to stage 3) and between layers 10 and 11 (the transition to the fully connected stage). Note that the association of changes does not exclusively happen between stages. For example, model A experiences a large drop in ID between layers 9 and 10, both of which layers are in stage 3. We will see in subsequent sections that the stage transitions also affect our other RR metrics.

Looking at the ID curves within a model for different data partitions can clearly be informative, but it is not until we compare the curves between models that the impact of using augmentation during training is apparent. In general, both models exhibit the hunchback shape but with unique characteristics. Model A's peak is lower but held at near peak levels for over twice as long (in number of layers) as model B. Conversely, the ID of Model B has a higher peak (also at layer 4) value but immediately decreases by over 25% and remains lower than model A for most of the remaining layers.

What is interesting is that the increase in ID happens in model B on the same peak ID layer as model A. At this point in the CNN the focus is on detecting the low-level features [51] (e.g., line, edges, parts of objects). Ansuini et al. [3] showed that a key function of the early layers in a CNN is to remove numerically dominant but irrelevant information (e.g., brightness of an image for number classification). From this we can conclude exactly which parts of a CNN are responsible for filtering out the additional irrelevant information to achieve better generalizable performance. Note that this does not necessarily imply that model A was unable to identify those additional features as it may have happened later in the model. While it makes intuitive sense that the early layers could be capable of removing the basic data variations introduced through data augmentations, it is not clear from the ID alone why the front loading of low-level feature removal is inherently learned at the initial stages in the ResNet20 models. One hypothesis is that the model naturally learns this to make better use of the overall learning capacity of CNN by allowing the deeper layers to focus more directly on higher level object features.

4.3 Nearest Neighbour Class Similarity

The NNCS curves for model A and model B are shown in Figure 4.1b. Each of the NNCS curves at layer 0 start at just below 0.2, experience a decrease in NNCS during stage 1, increases back to approximately 0.2 at layer 5, begins to increase and sees a sharp increase between layers 9 and 10, and finally levels off between layers 10 and 12. Like with the ID curves, there is a high degree of similarity between the training and validation NNCS curves, which are near identical until layer 9 for model A and layer 10 for model B. Once again, the variation on each of the curves is near 0.

Interestingly, the upward trend in NNCS is not monotonic across the model; between layers 0 to 3 we see a small but non-trivial decrease in the NNCS curves before the NNCS curves begin to trend upwards. We refer to this phenomenon as the **initial CS contraction**. The initial CS contraction is experienced by both models, with model B seeing a slightly greater decrease in ID during the contraction. Looking more closely, we see that the location of the initial NNCS contraction coincides in the same layer of the CNN as the initial ID peak observed in Figure 4.1a. Considering both ID and NNCS, we are given three pieces of novel insights into a CNN's behavior through the first stage of a ResNet20 CNN. First, that the removal of low-level irrelevant information is harmful to the local separability of classes. Second, the more aggressively a model removes the irrelevant features, the larger the negative impact on class similarity (i.e., in the early layers as ID increases, NNCS decreases). And finally, that the SGD learning algorithm allows a model to learn functional operations that are locally destructive to class similarity.

Importantly, this third insight appears to conflict with the finding from Alain et al. [2] which states, that "... the level of linear separability increases monotonically as we go to deeper layers" (with respect to ResNet style architectures). But we must remember that linear probing is looking at linear separability while NNCS is reflective of local (non-linear) separability. This distinction is important because it is direct evidence that the SGD learning algorithm (and its interactions with a ResNet style architecture) inherently allows a model to balance making seemingly local negative changes to the latent space while allowing the global objective function to be optimized. Keep in mind that this tradeoff happens early in the model and still allows for the model to optimize for both local and global separability at the deeper layers. But this observation leaves one to wonder what a learning strategy that allowed short term declines in both local and global separability could achieve or if such a strategy is even desirable.

At layer 5, all the curves have returned to approximately their initial NNCS values, while having larger IDs. From layer 5 onward the NNCS curves for each model begin to

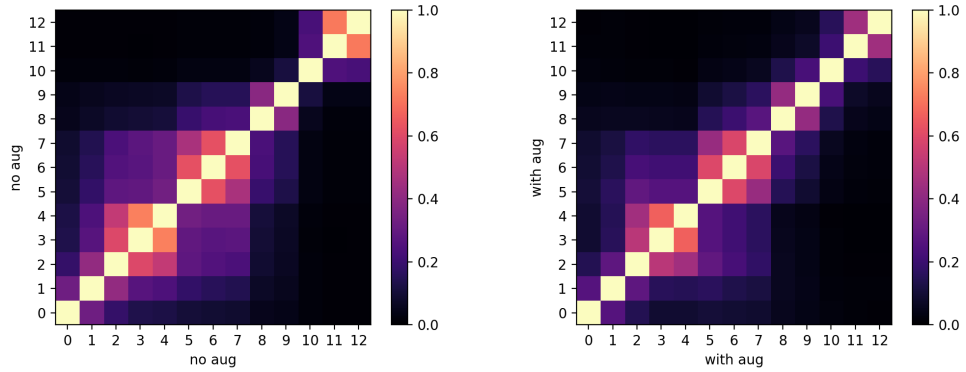
exhibit different trends. The curves for model A experience a minor decrease until stage 7, and at layer 8 the training and validation curves for model A diverge. Both curves experience a large spike between layers 9 and 10, at which point the two curves have diverge substantially. While the training NNCS hits 1.0 (the same as model A's training accuracy), the validation NNCS plateaus at 0.74, which is 6% lower than model A's validation accuracy. Unlike model A, after layer 5 the curves for model B immediately begin to increase, and do not begin to diverge until the dramatic increase in NNCS between layers 9 and 10. For model B the training curve and the validation curve go on to differ by up to 0.07.

Comparing the validation curves between models we see two regions of interest after layer 5. The first region is between layers 5 and 10, and the second being layers 10 and onward. In the first region the validation curve for both models begins and ends at approximately the same location, but during the second region model B's validation curve once again pulls a head. The increased NNCS curve during the first region for model B (relative to model A) supports the hypothesis that the upfront removal of the low-level feature allows the model to focus on improved classification throughout the remaining layers. While model B has higher NNCS for both curves during this first region, it is not until layer 10 that we see a large difference in how the two distributions of data are being processed in each CNN. For model A the training and validation curves differ significantly, while the difference for model B is under a third as big. The beginning of the large difference between the two curves for model A is a clear indication where in the model is responsible for overfitting.

As noted above, there is a large increase in the observed NNCS between layer 8 to layer 10. We refer to this change as the **NNCS cluster spike**. Unlike the initial NNCS contraction, the NNCS cluster spike begins in a layer whose output is structurally different from its input layer, with layer 8 having half the height and width while having double the number of channels, thus resulting in half the total number of features per sample than layer 7 has. Like with the initial NNCS contraction, the NNCS cluster spike corresponds with noticeable change in ID across the layers of interest. In this instance the NNCS cluster spike is inversely correlated with the change in ID. However, this correlation does not continue on into layer 11 and layer 12, while the ID continues to decrease changes in the NNCS curve are minimal.

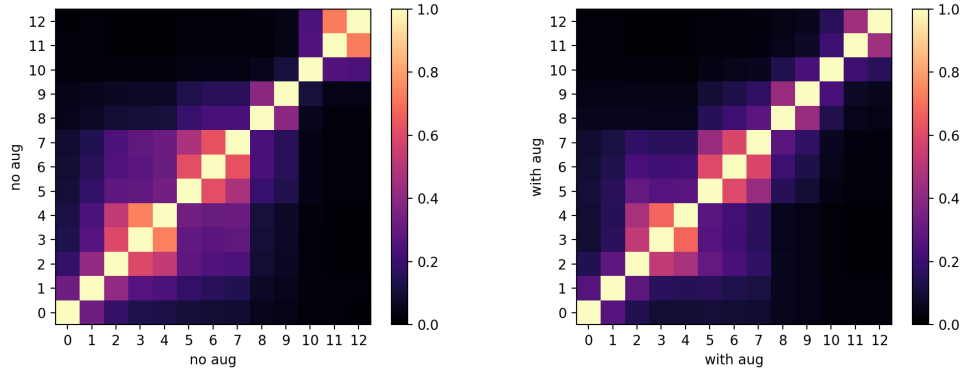
4.4 Intra-Layer Nearest Neighbour Layer Similarity

The intra-NNLS matrices for model A and model B are shown in Figure 4.2 for both the training and validation partitions. A given cell in any of the matrices represents the similarity between a given layer-pair's RR. When a layer is compared to itself it has a



(a) NNLS - Train, No Aug.

(b) NNLS - Train, With Aug.



(c) NNLS - Val., No Aug.

(d) NNLS - Val., With Aug.

Figure 4.2: An example calculating the NNLS matrix using four different combinations for model-dataset pairs. The left figures are of a model trained with no augmentations, the right figures are of a model trained with augmentations, the top figures are generated using the CIFAR10 training partition, and the bottom figures are generated using the CIFAR10 validation partition.

similarity of 1.0 as we are applying NNLS analysis between layers of a single model. As such, the bottom left to top right diagonal cells are equal to 1.0 since each layer is self-similar. We refer to this diagonal as the self-layer diagonal.

Figure 4.2a depicts the NNLS matrix for model A generated from the training dataset. In general, the closer a cell is to the diagonal the higher the similarity. Looking at the backbone layers (i.e., stages 1, 2, and 3) we observe a high degree of similarity between stages 1 and 2 (layers 2 to 7). Within this region the layers in stage 1 have an additional degree of self-similarity, and layers within stage 2 show the same pattern. Interestingly, the intra-stage self-similarity trend does not hold for stage 3 even though the structure of stage 3 is almost identical to stages 1 and 2. While layers 8 and 9 share a similarity of 0.44, the similarity between layers 9 and 10 drops to 0.16. The lack of similarity between layers in

stage 3 correlates to both the drastic decrease in ID and a drastic increase in NNCS (we will see in Chapter 6 why this difference exists).

Comparing the NNLS matrices between the training and validation partitions we see that the difference between them is surprisingly small. The largest absolute difference between the matrices is 0.0023 for model A, and 0.002 for model B. The similarity between the NNLS matrices of the training and validation partition is a strong indication that the models are processing the two distributions in a near identical manner. We now have two pieces of information that indicates that the distributions between the two partitions of data are highly correlated. Firstly, the similarity between the ID curves is calculated using distance ratios, and secondly the NNLS matrices which uses a ratio of relative neighbour similarities. In addition, we have the observation from NNCS which showed the training and validation class similarities are near identical until at least layer 8.

Another interesting observation is that the similarity between the training and validation partitions persists regardless of whether augmentations are used during training. Thus, we have an indication that the augmentations used during training were not necessarily specialized to the data symmetries found exclusively in the training partition. If the augmentations were specialized it would be expected that the model would learn specialized behaviour for only samples in the training set and would result in NNLS matrices that differ between the two respective partitions for a given model. Remember that the NNCS curves are still different between the partitions in the deeper sections of the models for both models. This is because the models are *drawing* the decisions boundaries and learning feature detectors in a way that is overfit to the specifics of the training distribution, and not to the overall target distribution shared between the training and validation partitions. The NNLS matrix does not notice the overfitting because the models are still manipulating the two near-identical partition distributions in a consistent manner that does not affect the distributions of nearest neighbours.

When comparing the matrices for model A with model B we can see that the qualitative block structure is still present. However, there are a few notable differences. The most extreme difference in similarity is that between layers 11 and 12. For the training partition in model A the layer similarity is 0.72 while for model B the corresponding layer similarity drops to 0.45. The high layer similarity for model A tells us that the linear classifier is not changing the neighbour structure of each sample on average. Since it is a linear operation there are only a few things that it can do to the 64-dimensional embedding vector as it gets mapped to a 10 class logit. Specifically, the linear operation could be stretching (multiplying), shifting (add a bias), collapsing (zeroing out), and/or skewing (multiplying features differently) the embeddings. Given that we are using Euclidean distance to calculate the kNN, only collapsing or skewing could be causing the large decrease in similarity between

layers 11 or 12. Why augmentations are reliably causing this change between layers 11 and 12 is not clear from our experiments.

The general similarity decrease experienced between layer 11 and 12 is also experienced between most consecutive layers when augmentation is applied during training. The overall decrease in similarity between neighbours supports the above hypothesis that the model is making fuller use of its discriminating capabilities, as the lower similarity between layers implies a greater degree of changes in local manifold structure. Interestingly, only the similarity between layers 9 and 10 experiences a non-negligible increase in similarity when augmentation is applied during training. Looking at the ID and NNCS curves in Figure 4.1 we see that the ID curves for model A experience a significant drop, and the NNCS curves experience part of the class cluster spike. But for model B the ID curves only experience a slight change, and the NNCS curves still experience part of the class cluster spike. Our hypothesis is that model A must learn to impose a large drop in ID to reach the ID required 10 classes at the output of the model. This large drop in ID caused by a drastic non-linear alteration to the underlying manifold is then observed by a small NNLS value (in cell 9-10 in Figures 4.2a and 4.2b). But because model B already has a low ID a drastic change in the underlying manifold is never learned.

4.5 Inter-Layer Nearest Neighbour Layer Similarity

In the last section we used NNLS analysis for comparing layers within a model to one another. We now consider NNLS analysis for comparing the manifolds between different models. The NNLS for a single layer, as shown in Equation 3.2, is

$$Q(H_a, H_b) = \frac{1}{n} \sum_{i}^{n} Q_s(K_{ai}, K_{bi}) \tag{4.1}$$

Now the nearest neighbour graphs H_a and H_b are no longer generated from the same CNN. Note, that K_{ai} and K_{bi} are an order set of nearest neighbor embeddings sample data, but now contain latent embeddings generated from different models. For each partition of data we perform three different comparisons:

1. Comparing models trained **without** augmentation to all other models trained **without** augmentation (for a total of 56 model comparisons).
2. Comparing models trained **with** augmentation to all other models trained **with** augmentation (for a total of 56 model comparisons).

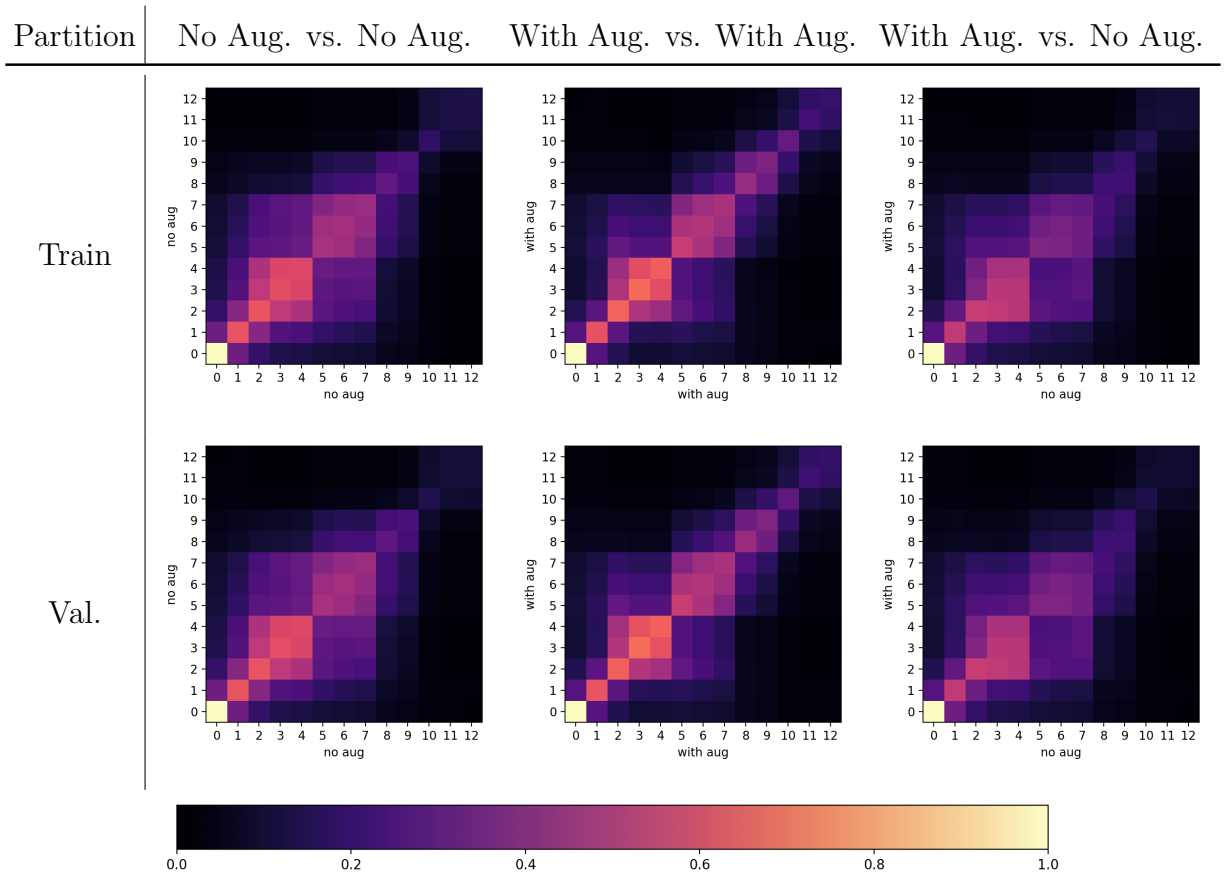


Figure 4.3: Results from an inter-model similarity comparison using NNLS matrices. For each dataset partition 3 comparisons are performed. On the left are matrices comparing models trained without augmentations to one another, in the middle are matrices comparing models trained with augmentations to one another, and on the right are matrices comparing models trained without augmentations to models trained with augmentations. For the left and center matrices no models are compared to themselves.

3. Comparing models trained **without** augmentation to all other models trained **with** augmentation (for a total of 64 model comparisons).

Figure 4.3 shows the average NNLS matrices for each comparison on both the training and validation partitions. Figure 4.4 shows a selection of data extracted from the NNLS matrices in Figure 4.3. Given the similarity between the training and validation matrices we focus our investigation on the NNLS matrices for the training partition unless otherwise stated.

In the previous intra-NNLS experiment models were only ever compared to themselves. Now that we are comparing the similarity between different models (i.e., models produced through separate training runs) the self-layer diagonal is no longer the identity (i.e., equal to

1.0). The only cell that is still 1.0 is the input layer. In this experiment we see a gradually diminishing similarity between layers in all the NNLS matrices the deeper the layer is. Despite this difference the block structure between stages is still present. Here we learn that the local structure learned between models with different initialization after training is relatively stable throughout the model, but to a diminishing degree relative to depth. The takeaway is that the ResNet20 models are not finding some arbitrary structure in the dataset, but instead the models learn to consistently find similar RR.

Continuing to focus on the self-layer diagonal we see that models trained with augmentation have a similarity that is equal to or higher than models trained without augmentation across all layers. The self-layer diagonals for the training partition are shown in Figure 4.4a. The difference between the curves gradually (but erratically) grows the deeper the layer. Figure 4.4c shows the difference between with and without augmentation curves for both the training and validation partitions. Here we learn that training with augmentations allows a model to learn the same RR more consistently. Note that the absolute difference is not particularly important here as it's an artifact of our choice in the number of samples relative to the number of neighbours used to construct the kNN.

There is a small difference between the training and validation matrices in general, but the difference is not as small as it was in the previous intra-NNLS experiment. Looking at the differences in similarity along the self-layer diagonal between training and validation partitions we see that training without augmentation results in a lower similarity later in the model, specifically from layers 10 and onward. The difference between the training and validation curves is shown in Figure 4.4d. The larger difference in these final layers corresponds to the observation made about the NNCS in the final layers of the model trained without augmentation which is that the models are overfitting primarily in the final layers of a model. Thus, NNLS is still capable of detection overfitting when used to compare models.

Comparing models trained with augmentations to models trained without augmentations we see an asymmetry in the matrices. The asymmetry in these matrices is expected as the model along the x-axis and y-axis use different training strategies. What is interesting about the asymmetry is that earlier layers in model B remain more like later layers in model A, but not vise-versa. The most noticeable occurrences are 1) within stage 1, 2) between stage 1 of model B and stage 2 of model A, and 3) between stage 2 of model B and stage 3 of model A. More surprising is that the similarity between layers 3 and 4 in model B and the layers in stage 2 of model A increase the deeper the layers in model A are. For example, layer 4 in model B is more similar to layer 7 than to layer 6 by 0.03 which is a 10% relative increase in similarity. These are the only examples in our NNLS matrices of a non-negligible increase in similarity when depth between layers increases. The increased

similarity earlier on in model B is another point of evidence which supports our hypothesis that augmentation allows a model to learn to *shift* certain feature detectors to earlier in the model. Note, we suspect that the shift is greater than what we can measure, as new feature detectors are likely being learned to take the place of ones that got shifted to earlier in the model thereby obscuring the impact.

4.6 Discussion

In this chapter we compared a CNN trained with and trained without augmentations to one another on both the training and validation partitions. By using RRA we gained several novel insights into the behavior of a ResNet20 model trained on CIFAR10, and on the impacts to the model when using augmentations during training.

The first novel insight is that the training and the validation RRs have a high degree of similarity as measured by ID, NNCS, intra-NNLS, and inter-NNLS. This informs us that the model is not inherently processing the two data distributions differently from one another. For ID, the ID curves closely followed one another, with the validation curves remaining consistently higher than the training curves. For NNCS, the training and validation curves are nearly identical for the first two thirds of the model. And for intra-NNLS and inter-NNLS, the NNLS matrices differed by under 1%. The similarities are independent of whether the model was trained with augmentations. The one significant difference between the training and validations curves is observed using NNCS in the last third of the model, where we see evidence of overfitting occurring. Using these observations, we know exactly where to focus our efforts to improve model performance.

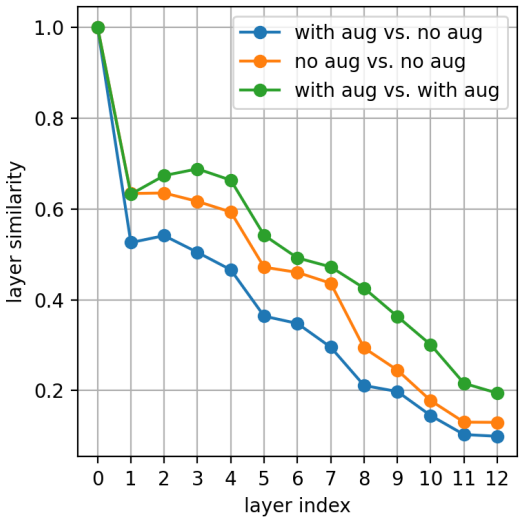
The second novel insight from our application of RRA is that using augmentations during training allows a model to learn the same RR more consistently. By using inter-NNLS the average similarity between different runs of the same model configuration increased when augmentation is applied. The increased similarity is important because it tells us that the model is not learning different behaviour from the augmentations but instead converging on a more general way of extracting useful latent information. On the other hand, if augmentations were used and the model similarities diverged, we would know that the learning processes were travelling unique paths in the weight space.

The third insight from using RRA is that the layer design of each layer has an impact on the transition between distinct phases in a model's RR. While this has been shown using other layer similarity metrics, to the best of our knowledge no other work as shown that the effects of the model architecture are not consistent between latent embedding-based

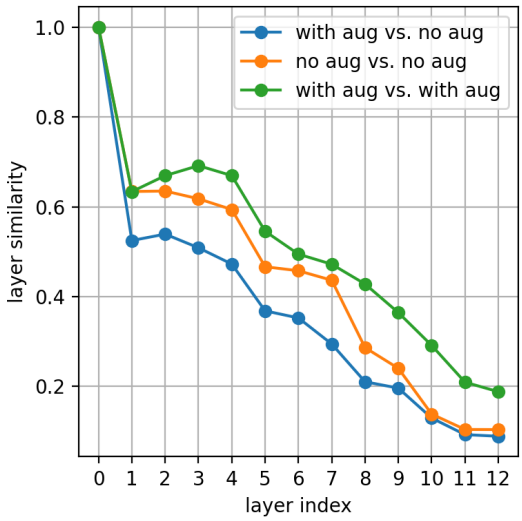
metrics, and that the depth of the given layer is a crucial factor on how the metrics are affected.

Finally, the fourth and most significant insight from our RRA is that we show compelling evidence that adding augmentations during the training process allows a CNN to better learn lower-level feature detectors earlier in the model's layers and opens model capacity to learn additional feature detectors in the deeper layers. This observation is demonstrated through the increase in ID in the early layers, the increase NNCS in the middle layers, the decrease in intra-NNLS between sequential layers, and elevated inter-NNLS between early layers of a model trained with augmentation and the deeper layers of a model trained without augmentation. Knowing the internal effects that augmentations have on a CNN could allow for a more cohesive design strategy to be employed instead of only considering performance-based outcomes.

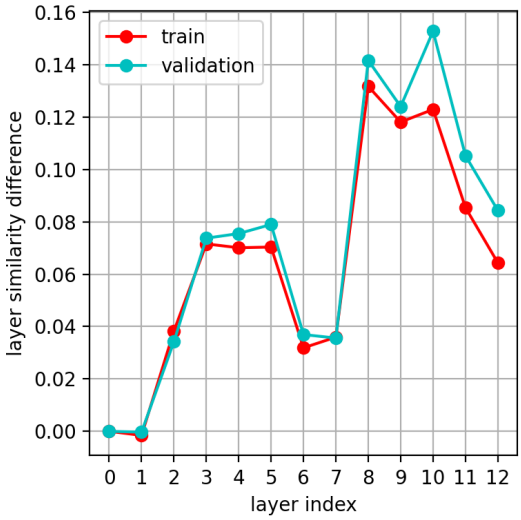
Overall, the experiments and the corresponding observations within this chapter show the need for a cohesive analytic framework capable of measuring a wide variety of aspects of a CNN's behaviour. Had we only relied on a single metric for analysis many of the effects of augmentation on the training and validation partitions would have been missed. While our analysis covered many aspects of a CNN, the underlying kNN model still allows for many additional pieces of information to be extracted. But first we focus on better understanding how the construction of the kNN effects the subsequent metrics used in this chapter.



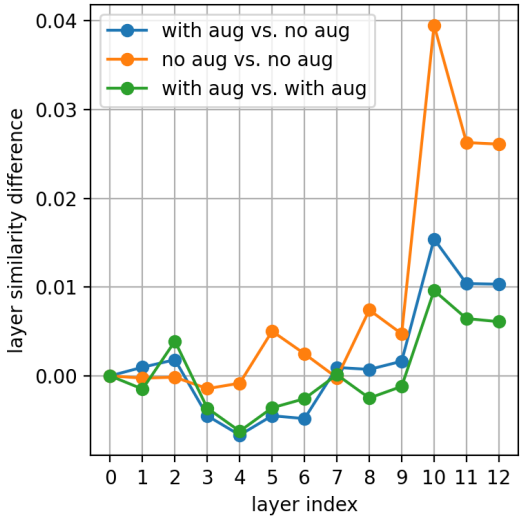
(a) NNLS Train



(b) NNLS Val



(c) NNLS No Aug. vs. With Aug.



(d) NNLS Train vs. Val

Figure 4.4: Trends extracted from the NNLS matrices in Figure 4.3. Figure 4.4a and Figure 4.4b compares the self-layer similarities (i.e., the bottom left to top right cells) for the training partition and validation partition, respectively. Figure 4.4c shows the difference between the self-layer similarity of models trained with augmentations and without augmentation. Figure 4.4d shows how the training and validation self-layer similarities compare to one another.

Chapter 5

The Effects of the Number of Sample and Number of Neighbours on the Observed Representational Response

When building the kNN for RRA one must choose the number of samples and the number of neighbours per sample. In Chapter 4 we used a specific number of samples and number of neighbours to construct the kNN used for RRA when analyzing the ResNet20 CIFAR10 model. Before progressing with our study of a CNN's RR, we first explore how the choice of these kNN hyperparameters impact the downstream RR metrics. In this chapter we focus on the interplay between the number of samples, the number of neighbours, and how altering these properties of the kNNs affects the quantities observed from each RR metric under consideration. Performing this analysis is critical for three reasons. First, to better understand how the RRA metrics will behave as the characteristics of a dataset changes. Second, to see if using a complete dataset is required for RRA analysis or if a representative subset is sufficient allowing for substantial computational savings. Third, to understand how the considered locality (controlled through the number of neighbours) affects the RRA metrics. Note that the constructions used in Chapter 4 were derived from results of this chapter. We felt it appropriate to introduce the reader to a concrete example of RRA before analyzing the intricacies of RRA kNN construction.

We extend our analysis to using both the CIFAR10 dataset and a 50-class subset of the ImageNet64x64 dataset [12]. ImageNet64x64 is a spatial down sampling of all images in the ImageNet dataset to 64 by 64 pixels. We refer to our 50-class subset as either ImageNet64x64-50 or Img64-50 for short. ImageNet64x64-50 has 1000 samples for each

Table 5.1: ResNet20 Training Results

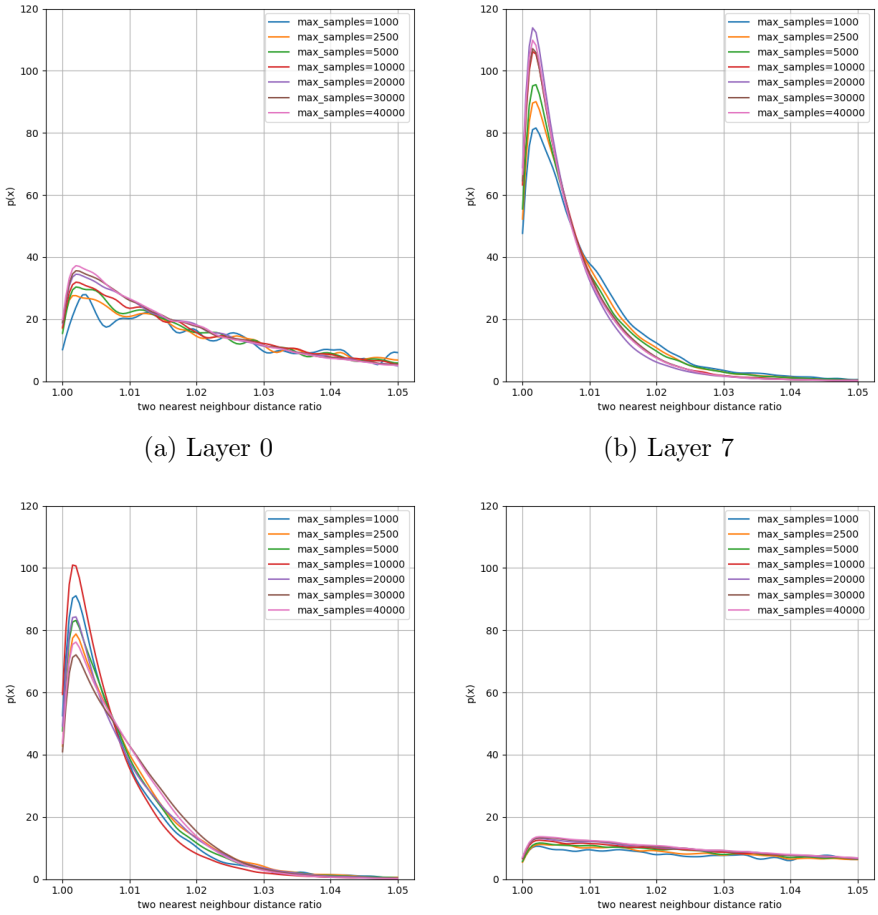
Model	Training Acc. (%)	Validation Acc. (%)
CIFAR10	1.0 ± 0.0	81.0 ± 0.46
Img64-50	1.0 ± 0.0	57.8 ± 0.1

class as to maintain a class balance. We selected 50 classes so that the total number of samples is the same as the CIFAR10 dataset. We use a ResNet20 for both datasets. Images in ImageNet64x64-50 have a height and width of 64. As such, the resulting feature maps for all layers in the ResNet20 model with a spatial dimension have double the height and double the width compared to applying the ResNet20 on CIFAR10. See Table 4.1 for the number of features per layer when applying ResNet20 to the CIFAR10 dataset. No augmentations are used during training for the experiments in this chapter as to allow for a more controlled set of experiments, however all samples are still used for training regardless of the number of samples used to construct the kNN. Like with most figures in this work, figures and numbers are averaged across 8 unique runs for any given model configuration. In this chapter we leave out the standard deviation bars in the figures given the number of curves per figure. The training and validation accuracy for both datasets are shown in Figure 5.1.

5.1 Effects on Intrinsic Dimensionality

We begin our analysis by investigating the effect the number of samples has on the measured ID at various points in a CNN. Since the ID approximator we use is only concerned with a sample's two nearest neighbours it will be unaffected by changes in the number of neighbours. As such we limit our investigation to changes in the number of samples. The seminal work by Ansuini *et al.* [3] used ID on a CNN to only investigate the effects of changing the number of samples had on the later layers of a CNN. Specifically, they noted that the number of samples used to construct the kNN has minimal effect on the calculated ID. A limitation of their analysis is that they limited their scope to the last few layers of a CNN. We extend this initial investigation to a spanning subset (from the input data to the last layer) of feature representations. A core component to the TwoNN estimator is the reliance on ratio between the second closest neighbour to the first closest neighbour for every sample in the kNN. Our first experiment investigates the distribution of TwoNN neighbour ratios as the number of samples is increased. Figure 5.1 visualizes the likelihood distributions for the 0^{th} (raw data), 7^{th}, 8^{th}, and 12^{th} layers' two-nearest-neighbour ratios $\mu = \frac{r_2}{r_1}$ for different number of samples used to construct the kNNs.

For each of the layers we visualize the two-nearest-neighbour ratio distributions between 1 and 1.05 as over 90% of the distributions lie within this bound. The distributions for each



(a) Layer 0

(b) Layer 7

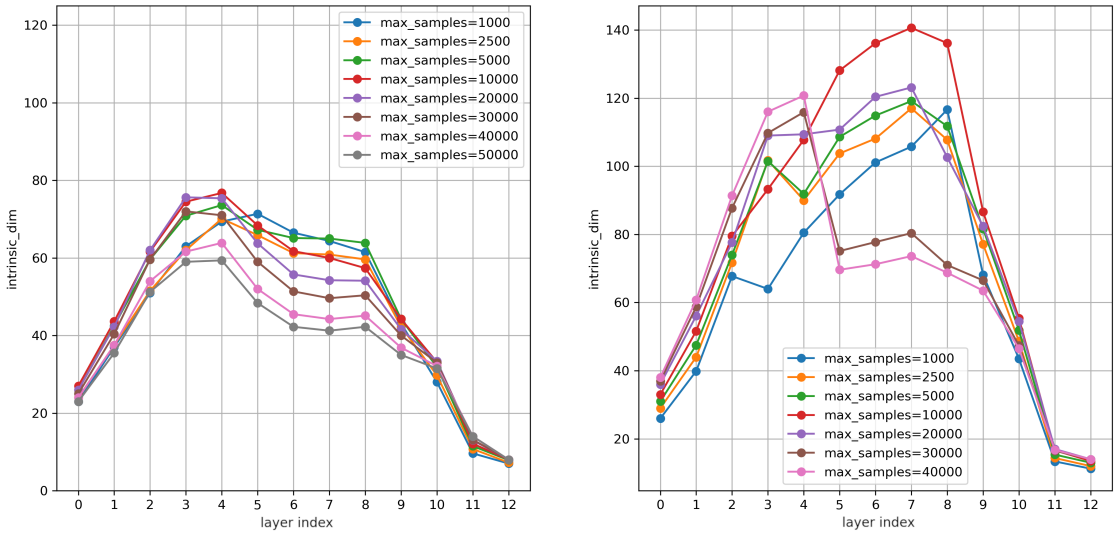
(c) Layer 8

(d) Layer 12

Figure 5.1: A visualization of the two nearest neighbour distance ratios for a selection of layers in a ResNet20 model applied to a CIFAR10 dataset. The distribution for each layer is shown when using a varying number of samples to generate the underlying kNN.

layer all follow a similar trend starting midrange in the likelihood values at a ratio of 1, quickly peaking at approximately 1.002, then quickly diminishing towards a likelihood of 0. Of interest is the fact that the ratio at which the likelihood peaks is consistent across all layers despite different feature map dimension sizes, intrinsic dimension of the layer (discuss below, and shown in Figure 5.2), and feature representations within each layer (i.e., what the feature within a layer represents). This trend is consistent across all layers not just the ones shown in Figure 5.1.

One aspect where the trends differ between layers is the magnitude of the initial likelihood (at a ratio of 1.0), the peak likelihood (near a ratio of 1.002), and the size of the likelihood's tail (as the ratio moves towards infinity). Initial layers and later layers in the CNN have



(a) CIFAR10 (b) ImageNet64x64-50

Figure 5.2: ID curves vs. the number of samples used to generate the underlying kNN. The left figure shows the ID curves for the CIFAR10 dataset, and the right figure shows the ID curves for the ImageNet64x64-50 dataset.

'shorter' and 'wider' ratio distributions while layers in the middle of the CNN have 'taller' and 'narrower' ratio distributions. A notable difference between our visualization of the ratio distribution and the theoretical underpinnings of the TwoNN ID approximator is the assumed form of the distribution. The TwoNN ID approximator assumes that the nearest neighbour ratio distributions have an exponential form (i.e., $\lambda e^{-\lambda x}$ for $x > 0$). However, our experiments deviate from an exponential distribution in the lower ratios with a lower likelihood than the distribution's peak. This observation is consistent for ResNet20 models trained on CIFAR10 and ImageNet64x64-50 datasets (not depicted).

Another difference between the likelihood distributions between layers is how these distributions change as the number of samples used to construct the kNN increases. Looking specifically at peak likelihood, the early layers and later layers follow a similar trend where the peak likelihood gradually monotonically increases as the number of samples used to construct the kNN increases; such a tread is visible in layer 0 (Figure 5.1a) and layer 12 (Figure 5.1d). However, we observe a mix of patterns in the middle layers of the CNN. Layer 7 (Figure 5.1b) follows the monotonic trend until the number of samples used is over 10000, while Layer 8 (Figure 5.1c) shows no such trend. On average, most layers stick closely with the monotonically increasing trend with minor deviation. The results for the CIFAR10 model are more consistent with monotonically increasing trends than ImageNet64x64-50.

Next, we look at ID curves for the ResNet20 model trained separately on CIFAR10 (Figure 5.2a) and on ImageNet64x64-50 (Figure 5.2b), respectively. The focus of this

experiment is to determine whether it is necessary to use all the samples from a dataset to accurately measure the ID of a CNN. Previous works [3] focus on consistency was limited to the last few layers of the network they investigated. Moreover, previous work limited such inquiry to a synthetic dataset on a limited number of samples (i.e., under 1440 samples [3]). We intentionally focus on more realistic image classification datasets and consider datasets consisting of up to 50000 samples.

The set of ID curves for CIFAR10 follow the known hunchback pattern where the ID is lower early in the model, quickly increases and peaks in the first few layers, then gradually decreases as one moves towards the later layers in the network. The middle of the model exhibits more dynamic patterns as the number of samples increases. Starting with 1000 samples (the blue curve) we observe that the ID curve still captures the main hunchback trend. In the first 4 layers (layers 0 to 3) the ID curve increases from 24 to 63. The ID curves increase as the number of samples used increases beyond 2500. However, at 40000 samples the ID curve for the early layers once again settles closer to the initial 1000 sample ID curve. Moving to the middle layers of the CNN, the general trend in the middle of the CNN has the ID curve's magnitude progressively decrease as the number of samples increases. Unlike the initial layers and end layers, the middle layers show no sign of converging to consistent ID values indicating that the CIFAR10 data might not be representative of the true underlying distribution. The consistent downward trend in the middle of the CNN only appears above 10000 samples. Looking at global ID curve trends on curves 10000 and above we see the emergence of a previously unobserved pattern in which the ID curve levels off for layers 6, 7, and 8 before decreasing again. We call this pattern the double-hunchback pattern. The emergence of this new pattern as the number of samples is increased demonstrates how the choice in the kNN's parameterization affects what phenomenon are observable.

The set of ID curves for ImageNet64x64-50 in Figure 5.2b follows a similar hunchback progression (in that the curves starts low, increase throughout the CNN, and finally decreases towards the minimal ID) as CIFAR10. In the initial layers for ImageNet64x64-50 (layers 0 to 2) there is a consistent increase in ID curves; the curves do not decrease as the number of samples reaches the size of the dataset.

The main differences between CIFAR10 and ImageNet64x64-50 are primarily in how the middle layers' ID curves change as the number of samples used to construct the kNNs increase. Curves constructed with samples between 1000 and 5000 all follow a similar pattern, except for a deviation in the 3^{rd} layer where the ID is lower than the other two curves. The curve constructed with 10000 has a significantly higher ID between layer 5 and layer 8, but this difference again diminishes with the curve constructed with 20000 samples. The most interesting curve transition is between 20000 and 30000 samples. At 30000

samples there is a dramatic change in layers 5 to 9 where the ID curve has lower values. It is during this transition that the same double hunchback emerges. The characteristic plateau of the double hunchback pattern appears in the same region of the CNN as it did with the CIFAR10 dataset (albeit one layer earlier). The initial progression of the double hunchback for both datasets starts with a decrease in observed ID between the 4^{th} and 5^{th} layers. For ImageNet64x64-50 the drop off is more proportionally pronounced.

Like the tests performed on CIFAR10 by Ansuini *et al.* [3], their tests on ImageNet are also limited to a small subset of 3.5k samples out of the entire dataset of over 1.2M samples. While ImageNet64x64-50 is still a subsampled dataset (in both feature map size and number of samples), we demonstrate that expanding the ID calculation to a more representative subset over 14x the size (50k samples instead of 3.5k) reveals the existence of the same double-hunchback pattern that was observed in the ResNet20 model trained on CIFAR10. Overall, our experiments reveal that ID is indeed sensitive to a change in the number of samples used to construct the kNNs.

5.2 Effects on Nearest Neighbour Class Similarity

Now we investigate how the choice in number of samples and the number of neighbours used to construct the kNN affects the observed NNCS in the ResNet20 model. Because we are using a classification CNN it is expected that samples within a class become more clustered as the features move deeper in the CNN. We must determine under what parameterization and to what degree this phenomenon is observable to ensure we do not obscure basic information contained within the kNN while studying a CNN's representational response.

As the parameters under investigation vary between extremes, the observed NNCS will undergo large changes. As one transitions from the beginning to end of a CNN it is expected that the NNCS transitions from a low number, near $\frac{1}{N_{classes}}$, to a value near the classification accuracy of the model. More specifically, we expect the initial NNCS values to be slightly above $\frac{1}{N_{classes}}$ since some inter-class similarity in the raw pixel values will correlate. As the data representation progresses deeper in the CNN, we expect the NNCS to increase in value and approach the performance of the CNN's classification accuracy (as seen with the related approach linear probing [2]).

For many of the trends we predict that the nature of the change will be conditional on the relative ratio between the number of neighbours per samples $N_{neighbours}$ and the number of samples $N_{samples}$ in the dataset. Such a ratio represents important information with respect to the local verse global inherent bias within the kNN. For example, if $N_{neighbours}$

is decreased, a more local region in the feature manifold is modelled and a corresponding increase in the overall trend line (across layers) of NNCS is expected since the closest neighbours to a given samples are more likely to be from the same class in a non-random classifier. If $N_{samples}$ is then decreased to keep the ratio consistent then it is expected to have the observed NNCS trend to adjust back to near the original observed trend line. For a given kNN let the ratio between the number of neighbours per sample to the number of samples in the data partition, which we call the Neighbour Density Ratio (NDR), be defined as

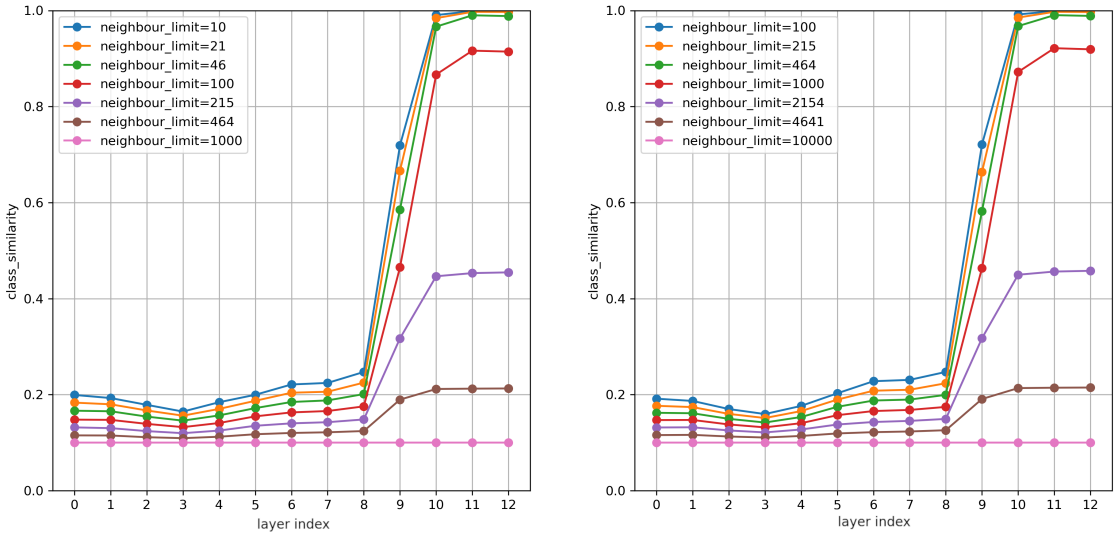
$$NDR = \frac{N_{neighbours}}{N_{samples}} \tag{5.1}$$

where the NDR is bounded between $\frac{2}{N_{samples}}$ and 1.0.

One trend where NDR will have a large effect on the observed NNCS is in the later layers of the CNN when the NDR is proportional to or lower than $\frac{1}{N_{classes}}$ (assuming a balanced dataset). When NDR is above $\frac{1}{N_{classes}}$ then the enforced clustering of classes (learned through training) will be imprecisely measured. For example, with a perfect classifier with large margins between class clusters it is guaranteed that the neighbours of any given sample will contain at least one sample from another class if the NDR is above $\frac{1}{N_{classes}}$. As we consider more locally constrained kNNs (i.e., a smaller NDR) it is expected that the observed NNCS will increase to represent the true NNCS of the underlying data manifold (i.e., the NNCS measured if an infinite i.i.d. data distribution was sampled).

We expect the number of samples used in a kNN to have gradually diminishing effects on the change in observed NNCS while keeping NDR constant. The reasoning for this is that $N_{samples} >> N_{classes}$ for both the CIFAR10 dataset and the ImageNet64x64-50 dataset. But since ImageNet64x64-50 has more classes, we suspect that the stabilization period of the NNCS will take additional samples when compared to CIFAR10. An extension of NDR could be to include the number of classes as a relevant variable, but we leave this consideration for future experiments as we do not control which classes are sampled from ImageNet64x64 to create ImageNet64x64-50 in this set of experiments.

Figure 5.3 depicts the results for CIFAR10 and Figure 5.4 depicts the results for ImageNet64x64-50. The plot in each figure contains the NNCS trend lines for a fixed number of neighbours across all layers in the CNN. Each dataset is tested on using a 1000 sample class balanced subset, and a 10000 sample class balanced subset. The set of number of neighbours used to construct the kNN for CIFAR10 follows the exponential pattern $\frac{total\_samples}{10^{x/3}}$ where x ranges from 0 to 6 in integer increments; at $x = 3$ the number of neighbours per sample equals the number of samples per class. Since ImageNet64x64-50 has more classes we adjust the increment pattern to $\frac{total\_samples}{50^{x/2}}$ where x ranges from 0 to 4



(a) NNCS - CIFAR10 - 1000 Samples

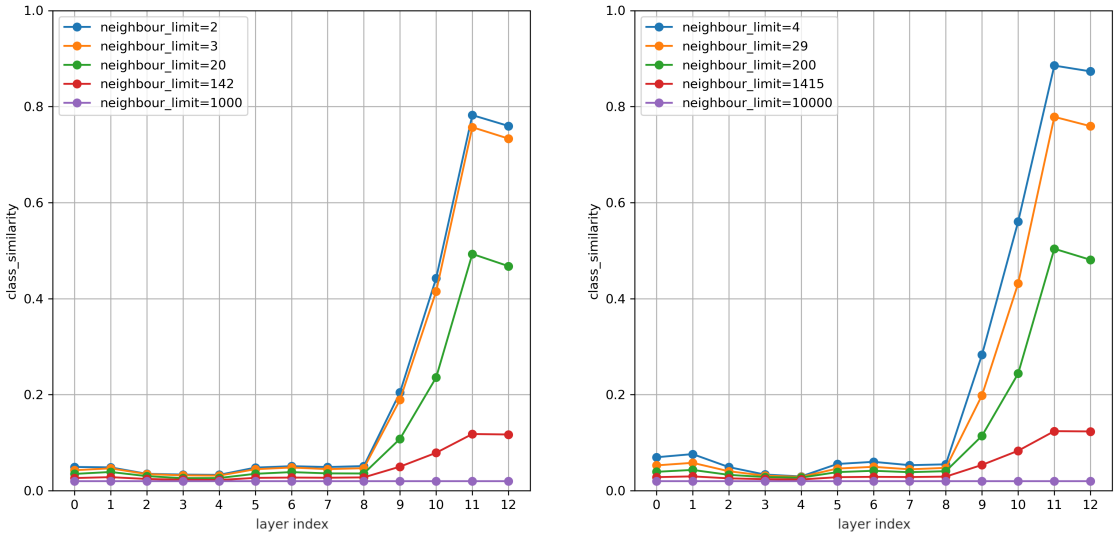
(b) NNCS - CIFAR10 - 10000 Samples

Figure 5.3: NNCS curves vs. the number of kNN neighbours used to generate the underlying kNN for the CFIAR10 dataset. The left and right figures shows the NNCS curves when 1000 and 10000 data samples are used in the kNN, respectively.

in integer increments. Now the number of neighbours per sample is equal to the samples per class at $x = 2$. Less steps are taken for ImageNet64x64-50 as to have each step remain meaningful.

Looking at Figure 5.3a we see the seven NNCS curves, one for each $N_{neighbours}$ under consideration. As $N_{neighbours}$ increases there is a monotonic decrease in the observed NNCS. The degree of the decrease varies depending on the layer of the CNN. The layers between layer 0 to layer 8 experience moderate amounts of change. Amongst this group of layers, layer 3 experiences the least change, while layer 8 experiences the greatest change. Each successive step in $N_{neighbours}$ results in an approximately uniform decrease in NNCS. All layers in the CNN settle at a NNCS of 0.1 (as expected in a balanced dataset with 10 classes). Layer 9 breaks the moderate change in NNCS trend as it starts with a higher NNCS of 0.72, has a larger variation in NNCS increments, but then still settling at a NNCS of 0.1. Layers 10 and onward starts at or near a NNCS of 1.0, but these layers do not see a significant change in NNCS until the third $N_{neighbours}$ value of 46. There is a major inflection point in NNCS increments when $NDR_{1000}^{i\geq 10} = 0.1$; that is, the relative difference between the NNCS at $N_{neighbours} = 46$ and $N_{neighbours} = 100$ is over five times smaller than the difference between the NNCS at $N_{neighbours} = 100$ and $N_{neighbours} = 215$.

Moving from layer 0 to layer 12, there is an upward trending change in the NNCS curves. The initial NNCS contraction is present in all of the non-pathological curves (i.e., when the number of samples equals the number of neighbours). Of note is that the initial NNCS



(a) NNCS - Img64-50 - 1000 Samples

(b) NNCS - Img64-50 - 10000 Samples

Figure 5.4: NNCS curves vs. the number of kNN neighbours used to generate the underlying kNN for the ImagenNet64x64-50 dataset. The left and right figures shows the NNCS curves when 1000 and 10000 data samples are used in the kNN, respectively.

contraction is present when as little as 1000 samples are used, while the peak in ID only becomes consistently observable with 5000 or more samples. Like with the contraction, the NNCS cluster spike is also visible in all non-pathological curves.

When moving from 1000 samples (Figure 5.3a) to 10000 samples (Figure 5.3b) we make the same corresponding order of magnitude increase to the respective number of neighbours used for each NNCS curve. For example, 10 turns into 100, and 215 turns into 2154, etc. Note that the 10000 sample subset of the CIFAR10 dataset is not a strict superset of the 1000 sample CIFAR10 subset. Comparing the respective curves, we see a limited change in the NNCS trend curves, with each respective point on each of the NNCS curves changing less than 0.01 NNCS on average. All major trends observed for NNCS curves generated using 1000 samples are also present in the NNCS curves generated using 10000 samples, including the initial NNCS contraction and the NNCS cluster spike.

The NNCS trend curves for ImageNet64x64-50 are shown in Figure 5.4a and Figure 5.4b, with 1000 samples and 10000 samples used to construct the kNNs, respectively. Looking at the ImageNet64x64-50 NNCS curves in Figure 5.4a, one can see several commonalities when compared to the respective curves generated using the CIFAR10 dataset in Figure 5.3a. Note the curve generated with $N_{neighbours} = 2$ breaks the establish exponential cadence discussed above as to ensure that the ID can still be calculated from the kNN.

At layer 0 the NNCS curves start low. When $N_{neighbours} = 2$ the NNCS is equal to 0.042, approximately double $\frac{1}{N_{classes}}$. At layer 1 there is a small upward step in the observed NNCS

before experiencing a decrease in NNCS. With respect to the initial NNCS contraction the minimum NNCS in the contraction is now at layer 4 instead of layer 3. The shift in the contraction's minimum also lengthens the number of layers with which the contraction takes place. After the contraction's minimum, the NNCS curves experience a small increase to where the NNCS is about equal to the NNCS values before the contraction. Once the feature maps hit layer 8 the NNCS curves undergo a quick rise in NNCS values. Like with the initial NNCS contraction, the rise in NNCS values appears similar to the NNCS cluster spike observed in the CIFAR10 experiments. In the CIFAR10 NNCS cluster spike the rate of change between the 10th and 11th layer is minor compared to the rate of change between the previous layers. But with ImageNet64x64-50 the rate of change between the 10th and 11th layer is observed to be greater than the preceding rates of change. In the last two layers of the CNN the rate of NNCS change between layers is substantially reduced. Unlike with CIFAR10 the NNCS values do not come close to 1.0, nor do they remain as stable (at least when $N_{neighbours}$ is greater than or equal to the neighbour inflection point) despite the classifier itself achieving a perfect accuracy on the training data.

Comparing the NNCS curves for ImageNet64x64-50 generated with 10000 samples to 1000 samples we see the same transitory patterns discussed above but with some subtle differences. First there is a general increase in the observed NNCS between all respective non-pathological curves. The increase is non-trivial for the three curves constructed with $N_{neighbours} \leq 20$, with $N_{neighbours} = 4$ experiencing the most pronounced change. The change for $N_{neighbours} = 4$ is likely because the neighbourhood size of 4 now more closely fits in to the exponential cadence between neighbourhood sizes. The increase at layer 0 no longer follows the observed trend in which the NNCS was approximately double $\frac{1}{N_{classes}}$.

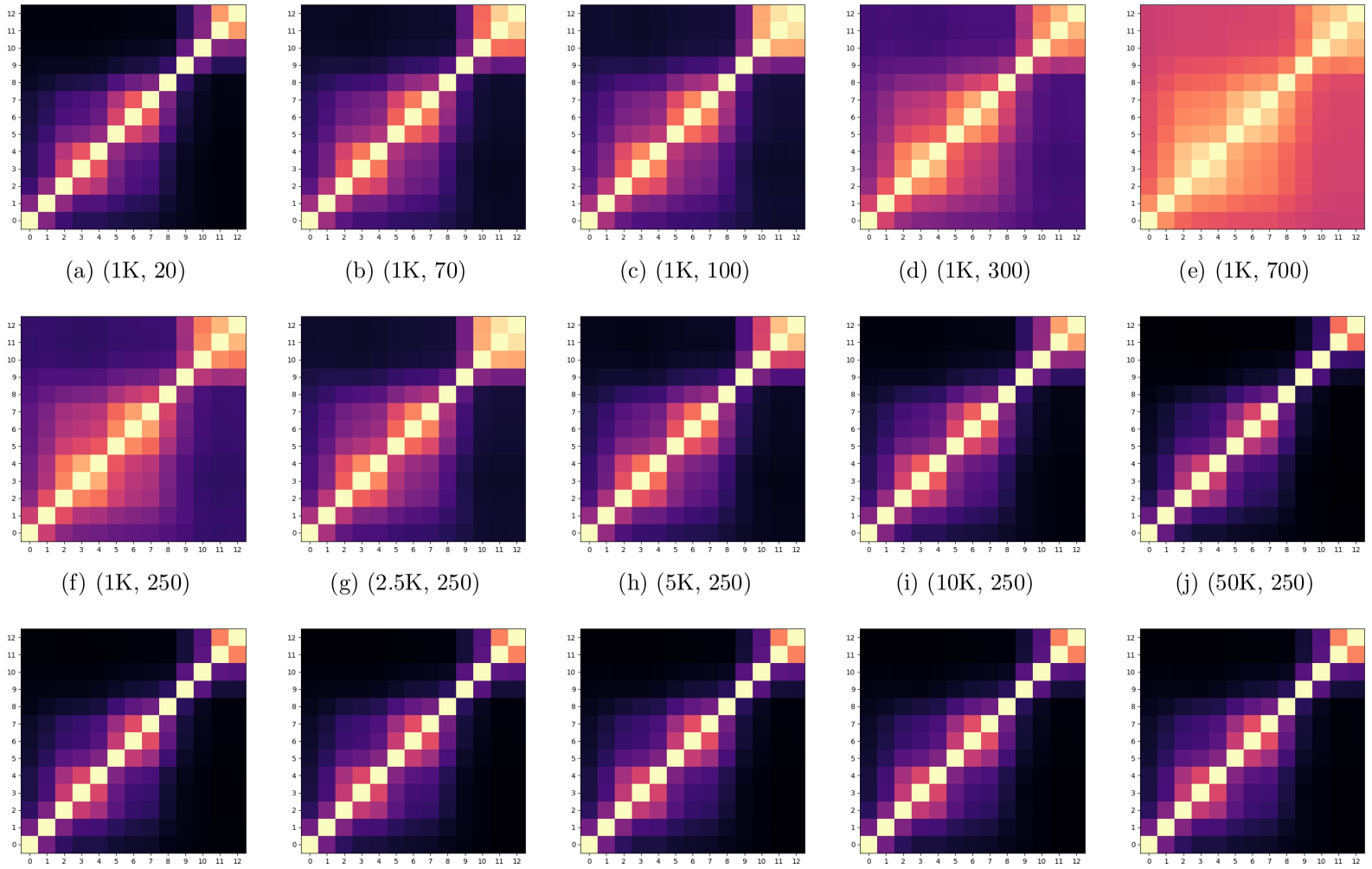
Looking closer at $N_{neighbours} = 4$ at layer 6 and layer 7, a new trend becomes more apparent in which the NNCS curves show a decrease greater than 0.02 NNCS. While the decrease is also present when 1000 samples are used, the decrease is relatively insignificant when compared to other phenomenon observed along the NNCS curves. At 10000 samples the new phenomenon is still relatively small, but is now no longer hidden by the lack of precision. At 10000 samples all non-pathological NNCS curves experience a greater difference between layer 6 and layer 7. The final notable difference is a large increase in observed NNCS at layer 11 and layer 12, but despite the increase the observed NNCS still fails to approach the performance of the classifier on the training data.

5.3 Effects on Nearest Neighbour Layer Similarity

So far, we have demonstrated that the ID and NNCS metrics observe different qualities depending on the number of samples used to construct the kNN and the number of neighbours used for each sample. We continue our analysis by looking at the effects these hyper-parameters have on the NNLS's ability to observe properties of a CNN's RR. In Figure 5.5 we look at the qualitative properties of the intra-NNLS matrix when $N_{samples}$ and $N_{neighbours}$ are changed. During this qualitative comparison we will be looking at general trends as one or both variables are altered, and what are the global trends within the NNLS matrix and which if any regions of the matrix are most impacted. The goal is to understand what patterns of a CNN's RR are visually observable at various kNN parameterizations.

In the top row of Figure 5.5 $N_{samples}$ is held constant while $N_{neighbours}$ is increased from 20 on the left to 700 on the right. Here we see that most high similarity layers lie on or near the diagonal. With 20 neighbours per sample, we see the same block structure that reflects the stage structure of the ResNet20 model. The further a layer comparison is from the self-layer diagonal the lower the similarity. Layers on opposite sides of the CNN have near 0 similarity. As the number of neighbours increases to 70 in Figure 5.5b there is an overall increase of average similarity between layers. The largest difference is the inter-layer similarities between layers 9 to 12 which now appear to form a more definitive block like group. Unlike other block groups (e.g., layer 2 to 4, and layers 5 to 7) layers 9 to 12 do not all have the same feature map dimensionality. At 100 samples in Figure 5.5c the NDR is now equal to the $\frac{1}{C}$. Much like the last transition there is an overall increase in both average inter-layer similarity and minimum inter-layer similarity. All the main block structures are still visually intact. The same pattern continues for $N_{neighbours} = 300$ and $N_{neighbours} = 700$ where both the average and minimum increase, but with the block structure becoming less pronounced as both the average and minimum similarity trend towards 1.0. Note, when $N_{neighbours} = N_{samples}$ each layer comparison in NNLS matrix would be 1.0.

In the middle row of Figure 5.5 $N_{samples}$ is increased from 1000 samples to 50000 samples while $N_{neighbours}$ is held constant. Qualitatively from the perspective of the NNLS matrix increasing $N_{samples}$ has the same effect as decreasing the $N_{neighbours}$. On the right of the middle row in Figure 5.5j we see a similar pattern to that of Figure 5.5a. As $N_{neighbours}$ decreases to 1000 in Figure 5.5f the NNLS matrix more closely resembles Figure 5.5d. In general, the closer two matrices' NDR values are the more visually similar they appear to one another. For example, Figure 5.5c in the top row and Figure 5.5g in the middle appear visually identical; both NNLS matrices are built from kNNs with NDR ratios of 0.1. The figures in the bottom row of Figure 5.5 provide additional evidence of this observation. Each figure in the bottom row is built with a kNN constructed with an NDR value of 0.01.



(a) (1K, 20) (b) (1K, 70) (c) (1K, 100) (d) (1K, 300) (e) (1K, 700)

(f) (1K, 250) (g) (2.5K, 250) (h) (5K, 250) (i) (10K, 250) (j) (50K, 250)

(k) (1k, 10) (l) (2.5k, 25) (m) (5k, 50) (n) (10k, 100) (o) (20k, 200)

Figure 5.5: Examples of intra-NNLS matrices, each calculated from a different kNN parameterization. The tuple $(N_{samples}, N_{neighbours})$ in the title for each plot indicates the number of samples used to construct the kNN and the number of neighbours for each sample in the kNN. The indices along each axis specify the ID of the layer in a ResNet20 model. The colour spectrum of these matrices are the same as other NNLS figures.

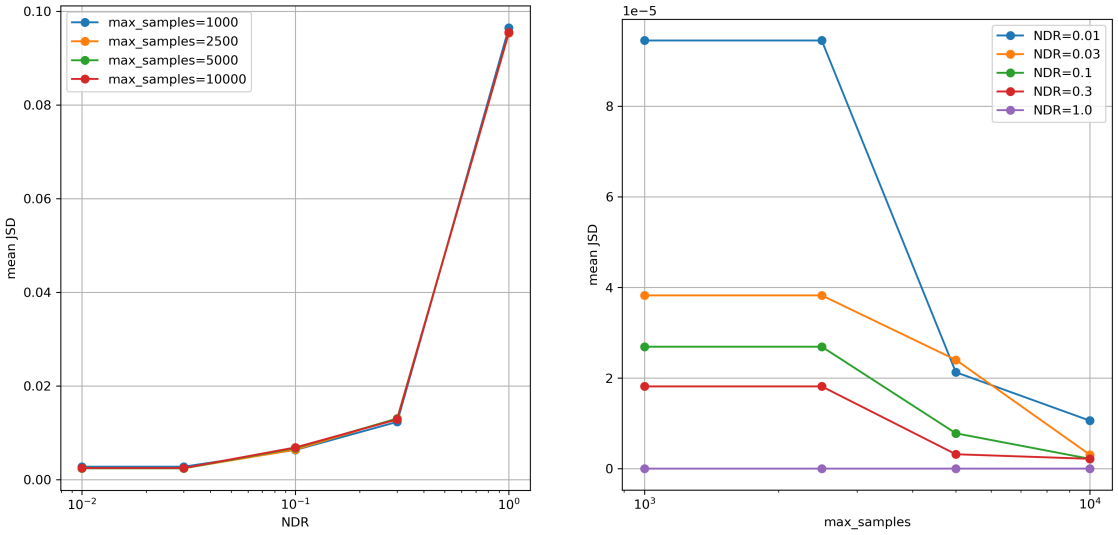
From results in Chapter 5.2 it was demonstrated that NNCS is invariant to changes in either $N_{samples}$ and $N_{neigbhours}$ as long as the NDR remained constant, and as long as $N_{samples}$ is sufficiently large. In Figure 5.5 the same observation is made across a set of different NNLS matrices. We now take a more rigours look at the NDR's role in the kNN's ability to measure a CNN's NNLS. To measure the difference between a given NNLS matrix M_p with parameterization p, and another NNLS matrix M_q with parameterization q we use the Jenson-Shannon Divergence (JSD) to compare individual values within the matrix, and then take the mean of all the JSD calculations. Let the similarity between two NNLS matrices M_p and M_q be defined as

$$NNLS\_SIM(M_p, M_q) = \frac{1}{l_p l_q} \sum_{i,j} JSD(M_p(i,j)||M_q(i,j)) \tag{5.2}$$

where the parameterizations p and q describe the CNN, dataset, and kNN settings used to generate each NNLS matrix, respectively. Let $M_p(i,j)$ and $M_q(i,j)$ be the NNLS of layer i in CNN p and layer j in CNN q. Finally, let l_p and l_q be the number of layers under consideration for the CNNs p and q, respectively. We compare NNLS matrices to one another using two methods: first, by holding $N_{samples}$ constant and comparing NNLS matrices with similar $N_{neighbours}$ to one another (Figure 5.6a), and second by holding NDR constant and comparing NNLS matrices with similar $N_{samples}$ to one another (Figure 5.6b).

Let $\{s_i\} \in S$ be an ascending ordered set where each element is the number of samples used to construct a kNN, and let $\{n_j\} \in N$ be an ascending ordered set where each element is the number of neighbours used to construct a kNN. For given (CNN, dataset) pair a NNLS matrix is calculated for each (s_i, n_j) tuple. For the tuple (s_i, n_j) let n_j be derived using some function f which takes the index in the set j and the number of samples in the tuple s_i as input. In other words, let the tuple for a given NNLS matrix parameterization be defined as $(s_i, n_j = f(s_i, j))$. Let the NDR for a given tuple (s_i, n_j) be defined as $r_{i,j} = \frac{n_j}{s_i}$. Let the set of NDR which exists for a given s_i be defined as $R = \{r_{i,j}|s_i\}$. In the following experiments let each NDRs for any s_i be equal. For example, if $S = \{10, 100\}$ and $R = \{0.3, 0.5, 0.7\}$, then $N_{10} = \{3, 5, 7\}$, and $N_{100} = \{30, 50, 70\}$, for a total of 6 unique NNLS parameterizations.

We aggregate the set of all NNLS parameterizations into sub-groups using two different methods. The first approach groups the NNLS matrices based on the number of samples used to generate the kNN, while the second approach groups the NNLS matrices based on the NDR. Using the above example, the first aggregation approach would contain 2 groups of 3 matrices, while the second approach would contain 3 groups of 2 matrices. In the following experiment $NNLS\_SIM$ is applied between neighbouring pairs of matrices. For



(a) NNLS_SIM - Holding $N_{samples}$ Constant

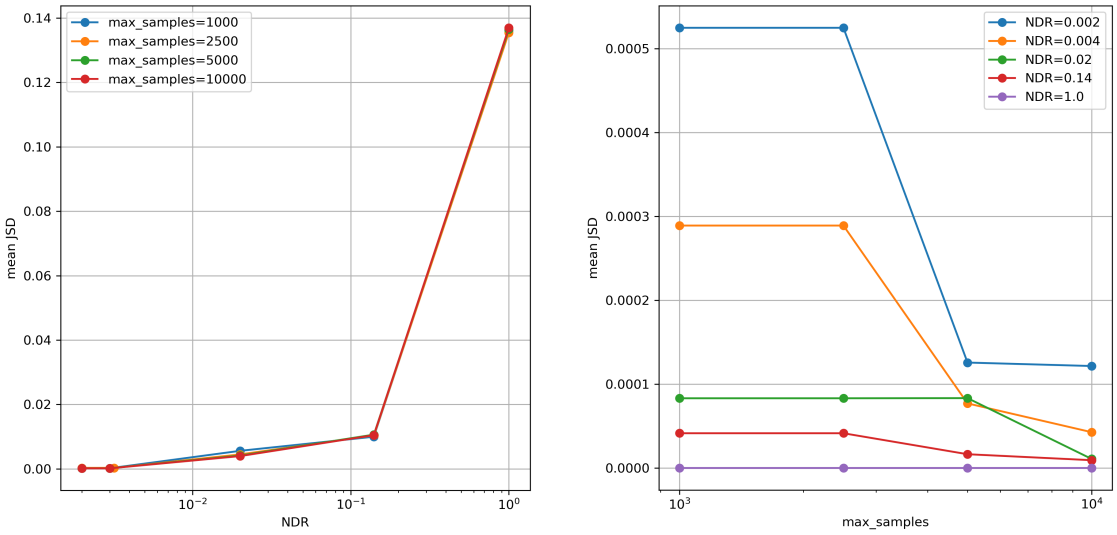
(b) NNLS_SIM - Holding NDR Constant

Figure 5.6: The JSD between CIFAR10 intra-NNLS matrics as the number of samples and the number of neighbours used to generate the underlying kNN are changed. The left figure holds the number of samples constant and changes the number of neighbours resulting in different NDR values. The right figure holds the ratio between the number of samples and number of neighbours constant.

a given subset of matrices, a NNLS matrix's neighbour is the NNLS matrix to the left if it along the non-static parameterization variable (e.g., for the first approach the non-static variable is NDR). In the boundary case the NNLS matrix's right neighbour is used. In the following experiments $S = \{1000, 2500, 5000, 10000\}$, $N = \{0.01, 0.03, 0.1, 0.3, 1.0\}$ is the set of the NDRs for the CIFAR10 dataset, and $N = \{0.002, 0.004, 0.02, 0.14, 1.0\}$ is the set of the NDRs for the ImageNet64x64-50 dataset. Note that the median value of each set N equals $\frac{1}{C}$.

The results for the first aggregation approach on the CIFAR10 dataset is shown in Figure 5.6a. The curve for each subset of data has a high degree of overlap. Moving from left to right there is an approximately linear increase in the $NNLS\_SIM$ between neighbouring NNLS matrices. Note that the x-axis is logarithmic, meaning that the relative difference in NDR between neighbouring matrices grows exponentially. For smaller NDR there is negligible change in the information content of the NNLS matrix even though the number of samples used to construct the kNN is tripled. The only significant change is between the two right most NNLS matrices. Thus, it is the change in the number of neighbours relative to the number of samples (i.e., the absolute change in NDR) which has the principal impact on the divergence between NNLS matrices.

The results for the second aggregation approach on the CIFAR10 dataset are shown in



(a) NNLS_SIM - Holding $N_{samples}$ Constant

(b) NNLS_SIM - Holding NDR Constant

Figure 5.7: The JSD between ImageNet64x64-50 intra-NNLS matrices as the number of samples and the number of neighbours used to generate the underlying kNN are changed. The left figure holds the number of samples constant and changes the number of neighbours. The right figure holds the ratio between the number of samples and number of neighbours constant.

Figure 5.6b. Two main trends emerge when the NDR is kept constant in a given subset of NNLS matrices. The first trend is that as the number of samples increases the observed dissimilarity between neighbouring NNLS matrices decrease. The second trend is that the rate of change in $NNLS\_SIM$ between neighbouring NNLS matrices is negatively correlated with the NDR. That is, changing the number of samples used to construct a kNN has a larger impact on the NNLS matrix the smaller the NDR. Overall, the difference between any NNLS matrix with the same NDR is less than 1e-4, an order of magnitude smaller than any change observed between NNLS matrices with different NDR in Figure 5.6a.

The results for ImageNet64x64-50 for the first aggregation approach and for the second aggregation approach are shown in Figure 5.7a and Figure 5.7b, respectively. Note for ImageNet64x64-50 the step progression for the NDR is changed such that the smallest NDR results limit the number of neighbours per samples to 2 for all kNNs. In Figure 5.7a we see the same trends that are observed with CIFAR10 with respect to the increasing divergence between NNLS matrices the larger the NDR, and that the divergence trend is still relatively invariant to the number of neighbours used to construct the kNN. The main variance in the trends is with NDRs at or below $\frac{1}{C}$. This means when you look at a local region the more samples you have the more reliable the NNLS matrix will be. For ImageNet64x64-50 it is likely that the smaller number of samples used per class (due to there being more classes) is related to this observation. Figure 5.7a also demonstrates many similarities with

the respective CIFAR10 figure. First the NDR negatively correlates with the magnitude of the divergence between NNLS matrices of the same NDR. Second, the NNLS matrices for ImageNet64x64-50 for a given NDR are more like one another the larger the number of samples. Finally, the relative divergence between NNLS matrices with the same NDR but different number of samples is over an order of magnitude smaller than the relative divergence between NNLS matrices with the same number of samples but with different NDRs.

For this work we choose JSD as the core metric to compare $NNLS\_SIM$. There were many other viable options which we did not explore beyond a cursory review. Alternatives metrics to JSD include Kullback–Leibler divergence, L1 distance, and L2 distance. JSD was chosen because it is bounded, symmetric, and qualitatively showed a good balance being sensitive to larger differences while reducing, but not ignoring the impact of small differences. On the limitations of using JSD, naively taking the mean across all JSD values has the potential to hide, or at least under represent, the effects of changing either $N_{samples}$ or $N_{neighbours}$. Looking at any of the plots in Figure 5.5 one will see that the most similar NNLS values lay along layers that are compared to themselves. As one moves further from the diagonal the similarity drops off dramatically. As such, comparing elements far from the identity will naturally result in a lower divergence as measured by JSD. For example, layer 3 and layer 11 will naturally have a low NNLS value. The low similarity will be true for most parameterizations of the kNN since the layers are far apart in the CNN.

5.4 Discussion

Throughout this chapter we produced four novel contributions to better understanding the proposed RRA framework. First, that the proposed RRA generalizes beyond a ResNet20 CIFAR10 combination. Second, that the three proposed RRA metrics are sensitive to changes in $N_{samples}$ and $N_{neighbours}$. Third, holding a consistent neighbour density ratio results in consistent measurements for NNCS and NNLS. Fourth, that representative RR information can be measured using a fraction of the overall dataset.

The two datasets used in this experiment were both used in a classification context and trained on a ResNet20 model. While CIFAR10 and ImageNet64x64-50 consist of the same number of samples, ImageNet64x64-50 has 5 times the number of classes and is also 4 times the total number of pixels per image. These similarities and differences between these two datasets were detected (to some extent) by the proposed RRA framework. Overall, the general trends of each of the three RR metrics across the two datasets were consistent with one another and behaved in the same manner as $N_{samples}$ or $N_{neighbours}$ was

changed. However, ImageNet64x64-50 still showed important differences when compared to CIFAR10. For ID ImageNet64x64-50 showed a higher peak ID and required more samples before the double hunchback pattern emerged. For NNCS ImageNet64x64-50 showed curves consistently lower when compared to CIFAR10. Finally, for NNLS ImageNet64x64-50 showed diminished similarity between neighbouring layers. Each one of these differences is expected given the increased complexity required to distinguish 5x more classes, the extra number of classes in the dataset, and the reduced performance of the ResNet20 model on the ImageNet64x64-50 dataset. These observations strengthen the evidence for the applicability of RRA. Future work will include testing RRA on additional model-dataset pairs.

The ID curves experienced unique behaviour depending on the depth of the layer as $N_{samples}$ is increased. In the initial layers the ID increased before returning to initial values, the middle layers experienced some increase before decreasing with no sign of converging, and the final layers experienced a slight increase. Noticeably, in contrast to prior work, our experiments identify the importance of using enough samples when calculating the ID. If too few samples are used then the measured ID is not reflective of the true underlying ID distribution, which results in key trends in the ID curves being obscured. However, once enough samples are used, the trend in the curve showed minor change. Identifying the appropriate number of samples to use for ID calculation is left for future work.

Unlike the ID curves, both NNCS and NNLS are sensitive to changes in both $N_{samples}$ and $N_{neighbours}$. For each of these metrics increasing $N_{samples}$ or decreasing $N_{neighbours}$ results in a similar change to the measured quantities. For NNCS, an increase in $N_{samples}$ results in an increase to the observed NNCS, and an increase in $N_{neighbours}$ results in a decrease in the observed NNCS. Conversely, for NNLS an increase in $N_{samples}$ results in a decrease in similarity between neighboring layers, and an increase in $N_{neighbours}$ results in a decrease in similarity between layers. By knowing how ID, NNCS, and NNLS are sensitive to changes in $N_{samples}$ and $N_{neighbours}$ we are better equipped to pick appropriate values to observe the desired quantities. Through our investigation we identified key features that only emerge under specific values of $N_{samples}$ and $N_{neighbours}$ (e.g., the ID double hunchback, or the initial NNCS contraction). But more analysis is required to know what other features of interest are relevant for analysis of CNNs and under what $N_{samples}$ and $N_{neighbours}$ values they emerge.

A key observation in this chapter is that if $N_{samples}$ and $N_{neighbours}$ are changed proportionally to one another (i.e., if the same NDR is used) then NNCS and NNLS produce near identical results (but where the larger $N_{samples}$ is, the more stable the observed values) even when the NDR is below 0.01 or close to 1.0. The invariance of NNCS and NNLS to $N_{samples}$ and $N_{neighbours}$ for a constant NDR informs us that we can reliably use a reduced number

of samples during the kNN calculation. Unfortunately, this potential for computational savings is hindered by the need of ID to have a sufficiently large number of samples to get a stable measurement. Where the ID relies on actual distances between neighbours, the NNCS and NNLS simply rely on discrete relations between neighbours (i.e., binary relations of are they the same class, or are they neighbours in more than one layer). Perhaps a more suitable complexity measure that is more invariant to the number of samples exists or can be derived (e.g., the distribution of nearest neighbour occurrences [64]). We leave these considerations to future work.

While we do not propose a generalized strategy for picking $N_{samples}$ and $N_{neighbours}$ for all datasets, we observed the point at which most trends became qualitatively stable for the datasets under consideration. For ID, the curves for CIFAR10 and ImageNet64x64-50 stabilized at 10000 samples and 30000 samples, respectively. For NNCS, the curves stabilized at NDR values sufficiently above $\frac{1}{C}$ (where C is the number of classes in the dataset). For NNLS, the NNLS matrices preserved similarity along the diagonal but showed minimal similarity between layers on opposing ends of a model when the NDR was below 0.01. Given this information we set $N_{neighbours} = \frac{N_{samples}}{10 * N_{classes}}$ for the remainder of this work, and leave a more rigorous selection process for future work. This heuristic provides a balance between limiting computational costs while providing relevant and informative information on the RRA for the models under consideration. Based on when the ID curves stabilize for each dataset, we use 15000 randomly selected samples with 150 neighbours per sample for CIFAR10, and 30000 randomly selected samples with 60 neighbours per sample for ImageNet64x64-50. The samples used are kept consistent between each of the 8 models for a given model-dataset pair.

Chapter 6

The Effects of Feature Dimensionality on the Observed Representational Response

In the last chapter it was shown that changing either the $N_{samples}$ or $N_{neighbours}$ affects the values observed by the RR metrics. We now expand our analysis to include alterations to the dimensionality of individual samples. The size of the feature maps throughout a CNN take on a variety of sizes. For the ResNet20 model trained on CIFAR10 the smallest feature map size is only 10 features at the model's head. The largest feature map in this model is $(32, 32, 16) = 16,384$ features large. When the model is trained on ImageNet64x64-50 the largest feature map is 4 times as large. In practice, when large CNNs are used on higher resolution images, the size of a feature can easily surpass tens of millions of features. If the dataset has a few thousand samples then calculating the kNN using these large feature maps requires expensive computational resources.

Other approaches that work directly with the feature maps throughout a CNN, such as linear probing [2], have also identified the large dimensionality of CNN features to be computationally problematic and that reducing the size of the feature map is a necessary practical compromise. In this chapter we investigate the consequences of using various dimensionality reduction operations on a dataset's features for a given layer in a CNN, and how different degrees of dimensionality reduction affects 1) the quality of the kNN, and 2) the values observed by the RR metrics.

6.1 Dimensionality Reduction

There are a host of potential data dimensionality reduction techniques, some require the calculation of a transformation matrix like PCA [21]. Methods like PCA require the calculation of eigenvectors which is an expensive calculation that also requires more unique samples than there are features. Random projection addresses this issue by randomly sampling a transformation matrix from a specific distribution that guarantees that the distance between samples is preserved within an upper bound. However, both PCA and random projection require storing a large transformation matrix. For deeper CNNs with larger feature maps, the storage of numerous transformation matrices can be storage and memory prohibitive. Non-parametric methods like TNSE [76] and UMAP [58] avoid the storage of the transformation matrix by reducing the dimension of the data in an iterative process that considers both the local and global structure of the dataset. These methods are computationally expensive to use, and require the entire mapping to be re-calculated when additional samples are added to the dataset. Other approaches like pooling make use of simple, easy-to-compute heuristics that leverage apriori knowledge of the ambient manifold for which the individual dataset samples are embedded in. For images, pooling assumes a spatial relationship between features, and assumes some level of local redundancy of the information in features near one another. In this chapter we investigate the effects of applying average pooling and max pooling to a layer's feature maps prior to the kNN calculation in RRA.

6.2 Multi Scale Representational Response Analysis

We now introduce Multi-Scale Representational Response Analysis (MS-RRA). At a high level the proposed method investigates how the similarity between latent representations at different spatial scales changes. More specifically, for a given set of data X we first extract latent representations of X for each operation of interest in the CNN. Each latent representation Y_i is independently spatially down sampled for each scale $s \in S$. Let $Y_i^s = P(Y_i, s)$ be the scaled latent representation of X output by operation v_v and down sampled by scale s using some function P. Then for each Y_i^s a kNN K_i^s is calculated and used for RRA. The complete process is described in Algorithm 1.

For the purposes of this work we only explore spatial pooling-based operations. By spatial, we mean the pooling operation is only applied along the height and width of the feature maps, and not the channel dimension. We investigate two spatial pooling-based operations 1) average pooling, and 2) max pooling. For each of these methods we explore

Algorithm 1 MS-RRA

Input: X, Z, F
Output: $\{ID_i^s\}, \{NNCS_i^s\}, \{NNLS_i^s\}$

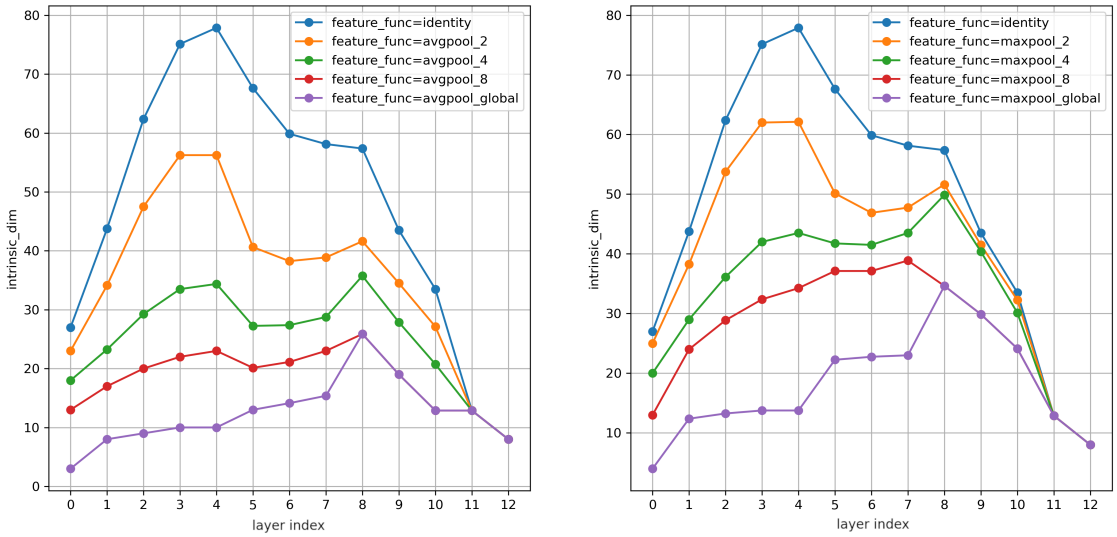
for i **do** ▷ For each operation in CNN F of interest
  $Y_i \leftarrow f_i(X)$ ▷ Extract features at operation i
  for s **do** ▷ For each scale
    $Y_i^s \leftarrow P(Y_i, s)$ ▷ Scale the features
    $K_i^s \leftarrow g(Y_i^s)$ ▷ Calculate the kNN
    $ID_i^s \leftarrow ID(K_i^s)$ ▷ Calculate ID
    $NNCS_i^s \leftarrow NNCS(K_i^s, Z)$ ▷ Calculate class similarity
    $NNLS_i^s \leftarrow NNLS(K_i^s, K_i^s)$ ▷ Calculate intra-NNLS
  end for
end for

the following non-overlapping windows sizes: 2, 4, 8, and global, where the window size refers to the side length of a square window. For global pooling the entire feature map height and width is pooled to a single value per channel. Each size is independently applied to the full-sized features at each layer in the same ResNet20 model used in the last section. No pooling operation is applied if a given feature map only has a channel dimension. The algorithm is shown in Algorithm 1.

Within this chapter we limit our investigation to a single parameterization of the kNN. For the CIFAR10 dataset $N_{samples} = 15000$, and for ImageNet64x64-50 data $N_{samples} = 30000$. For both datasets we use $N_{neighbours} = \frac{N_{samples}}{10 * N_{classes}}$; while somewhat arbitrary, this choice of $N_{neighbours}$ allows the novel phenomenon observed in Chapter 5 to be observable in the baseline (i.e., unpooled) RR metrics values. Thus, for CIFAR10 $N_{neighbours} = 150$, and for ImageNet64x64-50 $N_{neighbours} = 60$.

6.3 MS-RRA and ID

Conceptually, it is expected that the pooling operations do two things, first that it will directly reduce the ambient dimension of the feature map it was applied to, and second it will either smooth out or directly remove local features thereby reducing the intrinsic dimensionality of the feature space. As such, we hypothesize that all layers for which pooling is applied will see a reduction in the observed ID. Comparing average pool to max pool, it is expected that average pooling will result in an overall greater reduction in



(a) CIFAR10 - Average Pooling

(b) CIFAR10 - Max Pooling

Figure 6.1: The ID curves for a ResNet20 trained on CIFAR10. The left and right figures show the ID curves when pooling is applied to the features maps at different window sizes for average pooling and max pooling, respectively.

observed ID as the averaging operation will smooth out the intrinsic manifold the dataset is embedded in, whereas max pooling will naturally preserve the salient features for a given window size to a greater extent. By extension, as the window size of the pooling operation is increased it is expected that max pooling will also retain the salient features for longer.

Figure 6.1 shows the impact to the observed ID when the respective variants of pooling are applied to the base feature maps of the ResNet20 models. Each plot has 5 curves: the blue curve is the ID for unaltered features, and the orange curve, green curve, red curve, and purple curve are IDs for 2x, 4x, 8x, and global pooling, respectively. In general, the greater the pooling window size the large the decrease in observed ID. However, the decrease in ID is neither proportionally uniform across all layers, nor strongly correlated with the pooling window size.

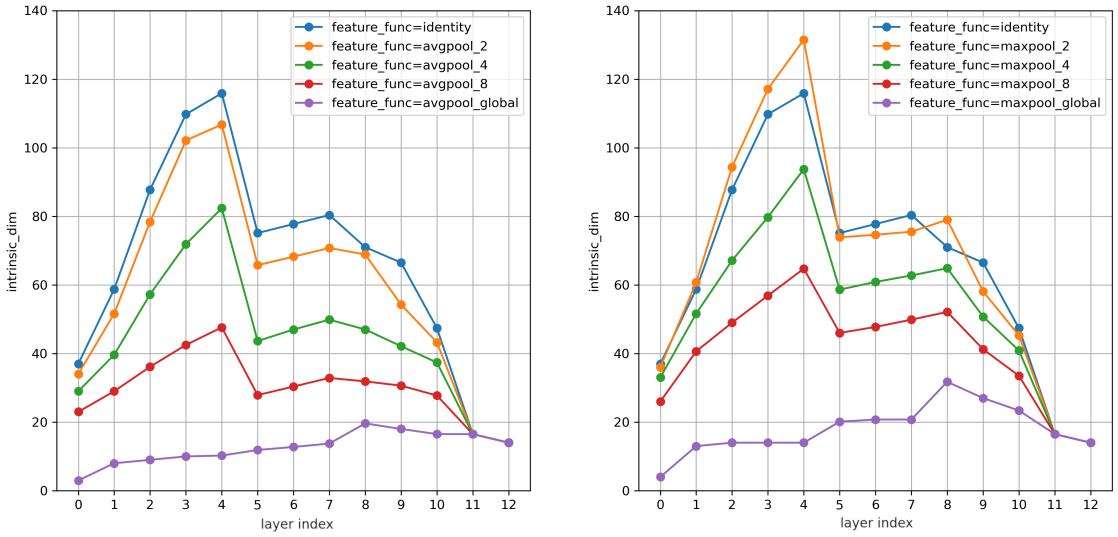
Looking at average pooling to CIFAR10 in Figure 6.1a, one can see the double hunchback pattern in blue that was discussed in previous experiments. When 2x average pooling is applied there is a general decrease in the magnitude, but the double hunchback pattern is still well defined. The ID decrease is most substantially in the middle layers of the CNN with the largest decline occurring at layer 5. Note, layer 5 already contains a natural 2x reduction in the feature map height and width. Moving to 4x average pooling, all layers continue to experience a decline in observed ID; now the general shape of the ID curves is altered. Layers between layer 0 to layer 4 see a large enough relative drop where the peak of the first *hunch* drops below the peak of the second hunch, with layers 5, 6, and 7 now

firmly forming an ID valley between the two peeks. The decrease in the latter layers of the CNN experience the smallest relative drop.

When 8x average pooling is applied the ID curve continues to lose the double hunchback shape, though it does not disappear entirely. When 8x is applied to layers 8, 9, and 10 they are reduced to features maps with only a channel dimension of size 64. Now the features in each of these three layers can be viewed as containing only global information. Between layer 8 to layer 12 (i.e., the feature maps containing only global information) the ID monotonically decreases. Note, applying 8x average pooling to layer 10 is the same as the GAP operation performed at layer 11. Finally, when global average pooling is applied to layer 0 it only has 3 dimensions, layer 1 to 4 have 16 dimensions, layer 5 to layer 7 have 32 dimensions, layer 8 to 11 have 64 dimensions, and layer 12 has 10 dimensions. The first 3 of these layer groups are associated with a narrow range of IDs. Layer 0 has an ID of 3, layer 1 to layer 4 have an ID between 8 to 10, and layer 5 to layer 7 have an ID between 23 to 35. Across this set of layers, the ID monotonically increases with large jumps in ID happening between feature maps of different dimensions. It is not until the final ResNet stage that a decrease in ID is observed. For global average pooling the trend for layer 8 and beyond is the same as 8x average pooling.

When global average pooling is applied to a feature map, it inherently removes all spatial information. So, when we calculate the ID for these feature maps, we are considering the complexity of the global presents of the features within a layer. The smaller the ID the more dominated a set of features are relative to the entire set. Conversely, the larger the ID the more balanced the features are being used. The increasing ID up to layer 8 indicates that the model is gradually increasing its effective use of unique features on average. As expected, there is a larger increase in ID as the model has more channels to work with. What is interesting is that the ID (at a global level) begins to decrease after the first layer in stage 3, just like the ID curve with no pooling applied. The decrease in the GAP curve tells us that the model is focusing on a specific subset of features to make the final class prediction. If we were to subtract the base ID curve from the global average pool curve, we could approximate what component of the feature space complexity in each layer is global information, and what component is local information.

Like average pooling, max pooling causes an overall decrease in the observed ID; the larger the window the greater reduction in observed ID. However, for a given window size, the reduction in ID is less substantial for max pooling. For 2x max pooling the double hunchback shape is still well defined. The magnitude in reduction for every layer that experiences a reduction is smaller than the reduction experienced during 2x average pooling. The layers 5, 6, and 7 drop enough relative to layer 8 to make the local maximum of layer 8 more pronounced than it was for 2x average pooling. Layers 9 and beyond experience a



(a) Img64-50 - Average Pooling

(b) Img64-50 - Max Pooling

Figure 6.2: The ID curves for a ResNet20 trained on ImageNet64x64-50. The left and right figures show the ID curves when pooling is applied to the features maps at different window sizes for average pooling and max pooling, respectively.

maximum reduction of 3 dimensions, showing that the key features the CNNs are considering are sparse. When 4x max pooling is applied, the first hunch is almost flattened to the point where the global maximum ID is now in the second hunch peak at layer 8. Layers 8 onward continue to experience negligible reduction in ID. It is not until 8x max pooling that the double hunchback pattern completely disappears. Now, with 8x max pooling, the ID curve follows a single concave like shape, which is much closer in shape (but not magnitude) to ID curves created with a smaller number of samples which are shown in Figure 5.2a. Unlike previous transitions, layers 8 and onward experience a non-negligible decline; the decline is significant enough to make layer 7 the peak ID. Note, the feature map at layer 7 still has a height and width, while layer 8 is reduced to having only a channel dimension. With 8x max pooling layer 10 is not equivalent to layer 11 like it is with average pooling. This shows the contrast between smoothing out features verse keeping only dominate ones. A similar step wise pattern emerges when global max pooling is applied to all layers between layer 0 and layer 7. However, now the steps have higher ID but less variation in ID.

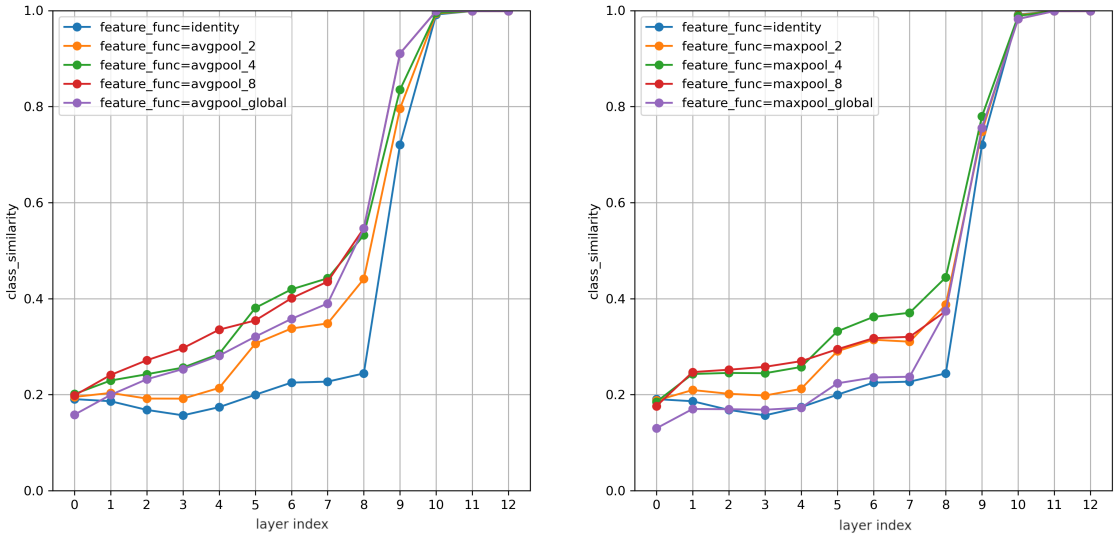
The results for average pooling applied to the feature maps of ImageNet64x64-50 are shown in Figure 6.2a. As a reminder the ImageNet64x64 images have double the height and width of CIFAR10. When 2x average pooling is used the ID curve sees a small but consistent reduction across all layers. Like with CIFAR10 2x average pooling the double hunchback pattern is still well defined. However, in contrast to with CIFAR10 2x average pooling, the relative reduction experienced by ImageNet64x64-50 is significantly smaller.

The limited change does not persist to 4x average pooling where the ID curve experiences 20 or above decrease in ID for layers between layer 1 and layer 9. The more substantial change still leaves the double hunchback pattern intact. 8x average pooling continues to see a drop in ID across all layers. While the double hunch back pattern is still visible, its general shape is diminished. By this point with CIFAR10 the double hunchback pattern is less visible than it is with ImageNet64x64-50. Throughout all the fixed average pooling windows, the layer with the global peak in layer 4 and local peak in layer 7 does not change. It is not until global average pooling is applied across the entire CNN for these peaks to either disappear or be shifted. We observe the same global average pooling trend as we observed with CIFAR10, where the ID monotonically increases until it peaks at layer 8, from which the ID monotonically decreases until the final layer in the CNN.

The ID curves for max pooling ImageNet64x64-50 are shown in Figure 6.2b. 2x maxpooling on ImageNet64x64-50 is the first and only example of the ID curve increasing at any layer above the original pooling-free ID curve. This happens between layer 1 to layer 4 and at layer 8. The increase at layer 8 results in shifting the local peak in the double hunchback to this layer. Note that the first set of layers that experience an increase all have the same feature map dimensionality of (64, 64, 16). All other layers experience a decrease of under 10 dimensions. Despite the increase in ID the double hunchback pattern is still well defined. Like with average pooling, applying 4x max pooling results in a large decrease in ID across most layers. The location of the local peaks at layer 4 and layer 7 continues to persist. 8x max pooling continues the trend of decreased ID in general with no other substantial changes. Like with CIFAR10, for a constant pooling window size, max pooling results in a higher ID at any given layer when compared to average pooling. Finally, when global max pooling is applied the ID curve experiences a large reduction in observed ID. Unlike with average pooling, the observed ID for ImageNet64x64-50 in the first two ResNet stages form discrete steps. The ID still peaks at layer 8, but now at a larger ID, then monotonically decreases until the final layer of the CNN.

6.4 MS-RRA and NNCS

Above it was shown that pooling the feature maps before the kNN is calculated can have a range of consequences on the observed ID. We now investigate the effects various types of pooling can have on the NNCS throughout a CNN. We expect that a layer's NNCS will be heavily influenced by the types of discriminating features a CNN learns to separate the classes in the dataset. In this context, three important characteristics of a dataset are 1) if the classes have similar features, 2) what the distribution of the features are, and 3) what



(a) CIFAR10 - Average Pooling

(b) CIFAR10 - Max Pooling

Figure 6.3: The NNCS curves for a ResNet20 trained on CIFAR10. The left and right figures show the NNCS curves when pooling is applied to the features maps at different window sizes for average pooling and max pooling, respectively.

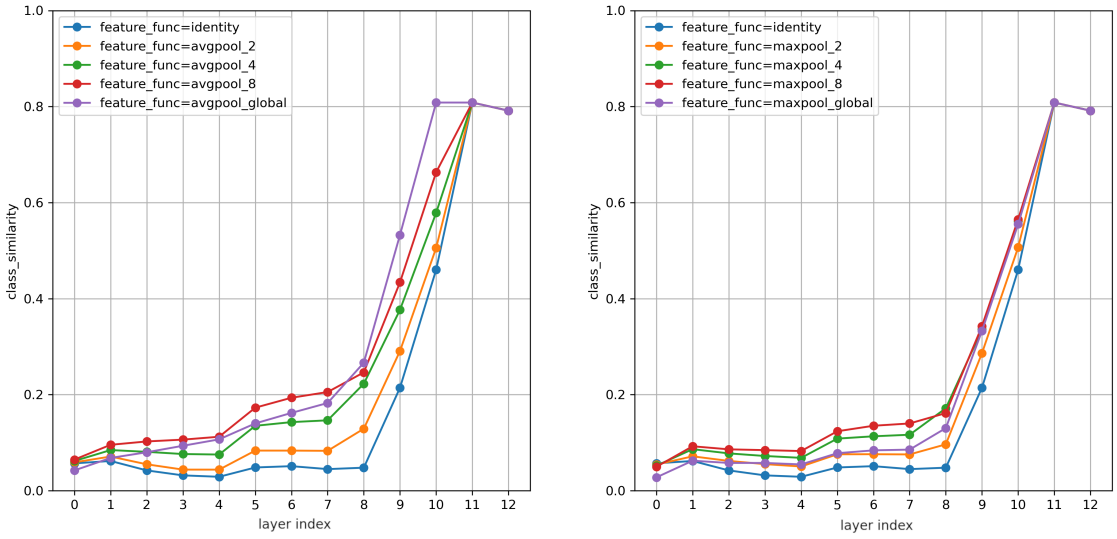
level of hierarchical features are used to distinguish the various classes. The similarity of the features will determine how hard two given classes are to distinguish from one another. The distribution of the features (both spatially and rate of occurrence) will determine how the types of pooling interact with NNCS. Finally, the hierarchical nature of the features will determine where in the CNN specific classes are more likely to become separable from one another.

The results for applying average pooling on CIFAR10 are shown in Figure 6.3a. The colour coding with respect to the size of the pooling is the same as the ID experiments. We now briefly review the NNCS curve when pooling is not applied (i.e., the blue curve). Starting at layer 0 (i.e., the raw dataset) one can see that the NNCS starts at 0.2, goes through an initial NNCS contraction until layer 3, from which the NNCS gradually increases until layer 8, at which point the NNCS spikes, and then NNCS at and beyond layer 10 approaches or equals 1.0. When 2x average pooling is applied the NNCS curve sees a small increase for the first 5 layers. With 2x average pooling there is now only a negligible decrease in NNCS during the layers that originally experienced the initial NNCS contraction. During the second ResNet stage (i.e., layers 5, 6, and 7) the NNCS curve jumps between 0.12 to 0.16 NNCS. The subsequent transition between feature map sizes (i.e., between layer 7 and layer 8) also experiences a non-negligible increase in NNCS. The largest increase between layers still happens between layer 8 and layer 9. When the average pooling window is increased to 4x the NNCS for every layer continues to increase. In addition, the subtleties in the NNCS

curve are diminished as the NNCS curve between layer 0 and layer 8 begins to resemble a linearly increasing line. However, the first 5 layers still have a smaller rate of increase, and a jump in NNCS still happens between layer 4 and layer 5. At 8x average pooling there is no longer a general increase in NNCS values, but now some layers experience a decrease. Despite this, the curve between layer 0 and layer 7 now more closely approximate a line. At layer 9 the NNCS now equals 0.92 indicating a higher level of class clustering in the latent structure of the dataset then what can be detected solely from the base (unpooled) domain. Once all layers are pooled to only their channels using global average pooling the NNCS between layer 0 and layer 7 (i.e., layers that still had non-channel dimensions) all see a 0.04 to 0.07 decrease in NNCS relative to the 8x pooling curve. For layer 0, we see the only instance of the NNCS decreasing below the identity curve. This is the first instance where pooling window increases but NNCS decreases.

As with previous experiments, max pooling shares many common trends with average pooling but with some key differences. In 2x max pooling layer 0 to layer 4 experience approximately the same change in ID as 2x average pooling did; however, the jump in NNCS between layer 4 and layer 5 is not as large. With 4x max pooling, layers between layer 1 to layer 9 all increase between 0.03 to 0.06 NNCS when compared to 2x average pooling. Layer 0 to layer 1 now experiences a more noticeable NNCS jump of 0.06 when compared to 4x average pooling. However, now the NNCS between layers 1 to 4 does not show a significant upward trend. The increasing trend does not emerge until after the layer 4 to layer 5 jump in NNCS. When 8x pooling is applied the curve between layer 1 and layer 7 finally begins to approximate a linearly increasing curve. Between layers 5 to 8 there is a significant drop in NNCS. However, the most substantial drop in NNCS occurs once max global pooling is applied. Most layers which experienced an increase in ID because of max pooling see their NNCS value for global max pooling fall back to approximately the NNCS value of the unpooled NNCS curve. The two notable exceptions are layer 0 where the NNCS is noticeably lower, and layer 8 which remains elevated; layers beyond 8 never experience significant changes for any pooling window size.

The results for applying average pooling to ImageNet64x64-50 are shown in Figure 6.4a. Like with CIFAR10, we see that the identity NNCS curve contains both the initial NNCS contraction, and the NNCS cluster spike. The main differences compared to CIFAR10 are the reduced magnitude of the NNCS curve, that the NNCS cluster spike happens over a larger range of layers, and that the spike doesn't approach 1.0. With 2x average pooling layers 0 to 4 experience minor change, layers 5 to 7 see a more substantial jump, and layer 8 sees the largest increase in NNCS out of all the layers. The increase in NNCS for subsequent layers diminishes until only negligible change is seen in layers 11 and 12. The 4x curve sees all NNCS values increase relative to the 2x NNCS curve, with the trend continuing



(a) Img64-50 - Average Pooling

(b) Img64-50 - Max Pooling

Figure 6.4: The NNCS curves for a ResNet20 trained on ImageNet64x64-50. The left and right figures show the NNCS curves when pooling is applied to the features maps at different window sizes for average pooling and max pooling, respectively.

between 4x and 8x. Looking at the global average pooled NNCS curve, one can see that the NNCS decreases relative to the 8x pooled curve for layers 0 to 7, but then increases between layers 8 to 10. In general, average pooling affects ImageNet64x64-50 in a similar manner to CIFAR10, but where the changes induced by average pooling are delayed by one window size. That is, the shape of the NNCS curves tend to be more correlated with the feature map size of the pooled feature map than with the magnitude of the pooling window. In fact, when comparing pooled CIFAR10 curves and ImageNet64x64-50 curves the average correlation coefficient between curves with equal average window size is 0.952, and with equal feature map size is 0.969. This trend is consistent with the ID experiments where the ID for ImageNet64x64-50 is largely unaffected by 2x pooling.

Moving to max pooling in ImageNet64x64-50, shown in Figure 6.4b, the pooled NNCS curves are depressed relative to their average pooled counter parts for the same pooling window size. As the pooling window size increases up to 8x the corresponding increase in NNCS is more consistent at preserving the general trend of the identity NNCS curve. As with CIFAR10 max pooling, the ImageNet64x64-50 max pooled curve magnitude between layer 0 to layer 8 experiences a moderate drop in NNCS, while seeing minor changes in latter layers. The decrease in NNCS during the early layers causes it to align with the 2x max pooled curves. Interestingly, the global max pooled curves for both CIFAR10 and ImageNet64x64-50 align with the NNCS curves generated using feature maps of the same dimensionality (since 2x max pooled ImageNet64x64-50 has the same dimensionality as the

identity CIFAR10).

6.5 MS-RRA and Intra-NNLS

From our ID analysis we know that the complexity of the embeddings reduces as the pooling window increases, and from our NNCS analysis we know that using increasing global information (from larger pooling window sizes) produces greater intra-class clustering at most points throughout a model. In this experiment we investigate how the structure of information changes throughout a model using intra-NNLS analysis. The intra-NNLS matrices for the CIFAR10 and ImageNet64x64-50 datasets for each pooling scale are shown in Figure 6.5. The intra-NNLS matrix when no pooling is applied is also shown for both models.

Looking at the identity intra-NNLS matrix for CIFAR10 we see the same similarity block structure as discussed in Chapter 4.4. There is a non-negligible similarity between the layers within the first and second stages, and between layers in the third stage and beyond. The similarity between these layers is greater if they are closer to one another. All pooling scales produce at least some change to the observed NNLS. When 2x average pooling is applied to the features of the CIFAR10 dataset we observe a noticeable decrease in the similarity between layers in the first two stages of the ResNet20 model. The further the layers are from one another the greater the decrease in similarity. The decrease in similarity is to be expected since high frequency spatial features are averaged away. As the pooling window is increased there continues to be a decrease in overall similarity between all layers. However, the similarity does not completely diminish and many of the trends partially remain. Once global average pooling is applied the block like pattern disappears in the first two stages of the ResNet20 model. The stark difference between the base intra-NNLS matrix and the global average pool matrix indicates that it is the local spatial features responsible for the broader similarity between the first two stages in the model.

The partial invariance to feature map size produced by average pooling in the intra-NNLS matrices provides two potential benefits. First, that value can be extracted from NNLS matrices without requiring the calculation of the RR kNNs using the full resolution dataset, and second, that there is coherent structure in the manifold formed by the pooled features. Whether the pooled manifold structure is the same structure as the base manifolds is to be seen. We hypothesize that the structure of the manifolds between the different scales gradually reduces the larger the pooling window. Based on the results from the ID curves within this chapter we suspect that the structure in the early stages of the model is most impacted.

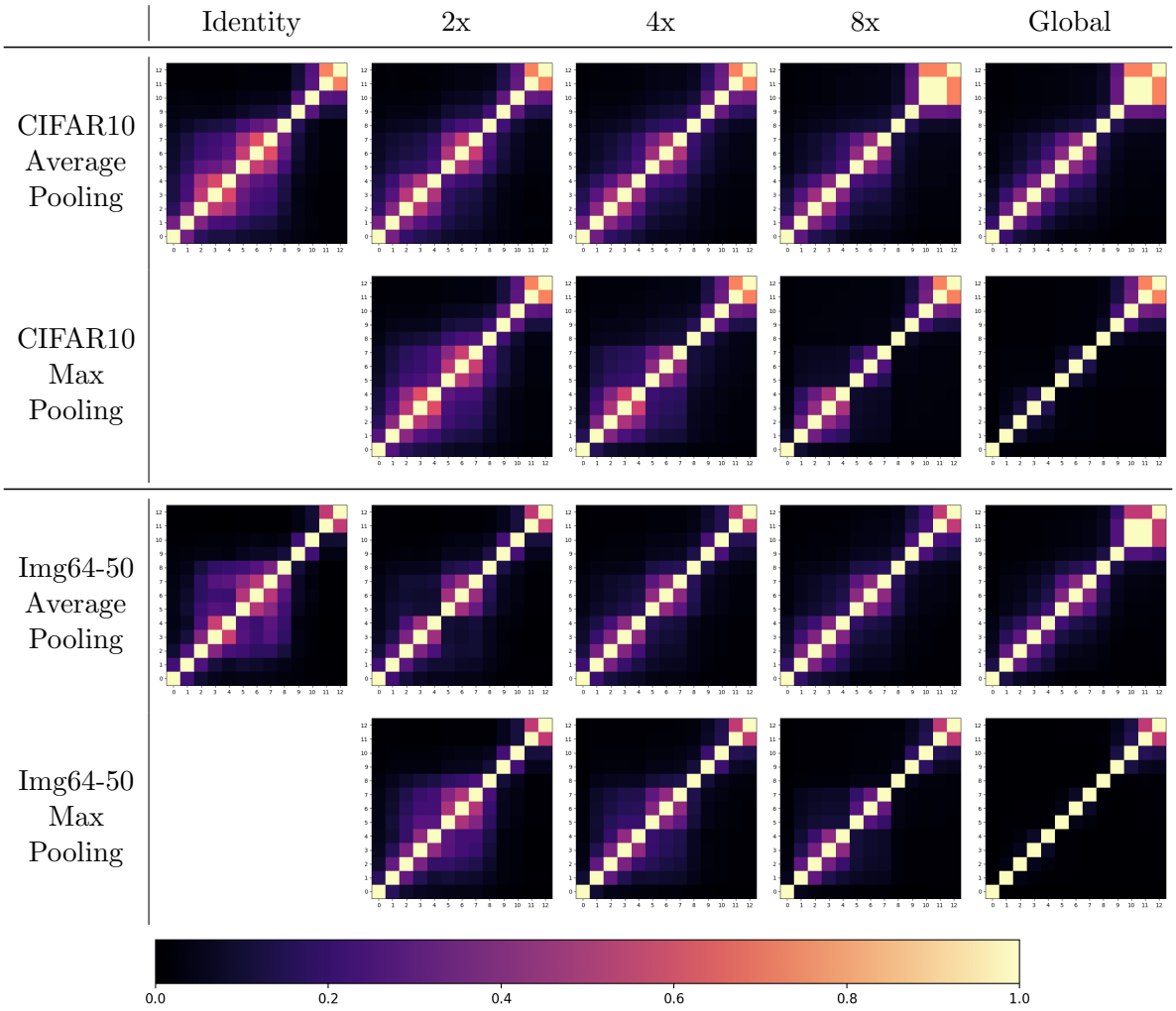


Figure 6.5: Intra-NNLS matrices computed from the latent embeddings of pooled features. The top half of the figure shows intra-NNLS matrices for the CIFAR10 dataset, and the lower half of the figure shows intra-NNLS matrices for the ImageNet64x64-50 dataset. For each dataset, there is a base intra-NNLS matrix without pooling applied, and a matrix for 2x, 4x, 8x, and global pooling windows for both average pooling and max pooling.

Unlike average pooling, max pooling better preserves the blocks structure between the early layers. For 2x max pooling on the CIFAR10 dataset we see that the inter-stage similarity between stages 1 and 2 is largely preserved; for average pooling this inter-stage similarity was significantly degraded when as little as a 2x pooling window size is applied. The deeper layer similarities are unaffected. Surprisingly, the inter-stage similarity remains when 4x max pooling is applied, but at 8x max pooling only the intra-stage block structures remain. Once global max pooling is applied all inter-layer similarity becomes negligible except for the layers with no spatial resolution to begin with. The lack of similarity for global max pooling tells us that a single dominate feature does not carry forward between contiguous layers. Max pooling is different from average pooling in that max pooling perseveres important local information, whereas average pooling removes them. Since average pooling caused the inter-stage similarity to quickly diminish while max pooling allows it to persist for several additional pooling sizes, we have evidence that it is the local features that are responsible for the block structure within the base intra-NNLS matrix.

The average similarity between the layers in stage 1 and the layers in stage 2 is 0.32 for the base CIFAR10 intra-NNLS matrix. This average similarity between stages tells us that regardless of the operations in stage 2, 32% of the structure in the latent manifolds between the first layer of stage 1 (i.e., layer 2) to the last layer of stage 2 (i.e., layer 7) does not experience substantial change; while the numerical values may change, the relations between latent embeddings do not. From the max pooling experiment, we know that this inter-stage similarity is caused by persistent local features.

The intra-NNLS matrices for the ImageNet64x64-50 dataset exhibits many of the trends shown in the CIFAR10 analysis, but with some superficial differences. The main general difference between the two datasets is that the average similarity in all ImageNet64x64-50 NNLS matrices have a decreased similarity; this is caused by the kNN parameterization. By increasing the NDR it would be possible to match the similarities. When 2x average pooling is applied the average inter-stage similarity between stages 1 and 2 is almost completely removed while the block intra-stage similarity remains. With CIFAR10, the inter-stage similarities were not diminished to this degree. Further experimentation is required to determine if this difference is a result of the kNN parameterization or is an intrinsic characteristic to the ImageNet64x64 dataset. Like with CIFAR10, as 4x to 8x average pooling is applied the overall intra-NNLS structure is consistent. The NNLS consistency means that the ImageNet64x64-50 latent embeddings experience the same spatial invariance. It is not until global average pooling where the inter-layer similarities see a large decrease. The max pooling experiments also demonstrate similar trends to the CIFAR10 max pooling, including the preservation of elevated inter-stage similarity for 2x to 8x window sizes, and the removal of almost all inter-layer similarity for global max pooling.

6.6 MS-RRA and Inter-NNLS

In the last section we applied pooling operations to the latent features and measured the RR's characteristics using intra-NNLS matrices. From this analysis we observed three main things, first that pooled latent features have structure, second that the structure is qualitatively similar to the unpooled intra-NNLS matrices but with the similarly diminishing the greater the pooling window, and finally that the type of pooling effects the properties of the similarity. We now examine whether the structure between the base latent embeddings and the pooled latent embeddings are either the same (or similar), or if the structures are substantially different from one another. This distinction is important as it will inform us to what degree the latent structure is imposed by the model architecture, and if the latent structure is inherently preserved across different spatial resolutions.

The following equation calculates the similarity between the base latent embeddings of a layer and the pooled latent embeddings of another layer within the same model

$$Q(H_a^1, H_b^c) = \frac{1}{n} \sum_{i}^{n} Q_s(K_{ai}^1, K_{bi}^c) \tag{6.1}$$

where H_a^1 is kNN graph for layer a for some model generated from layer a's features maps at $1x$ scale, and H_b^c is a kNN graph for layer b in the same model generated from layer b's features maps at a pooled window size of c using some pooling function. Let $\mathbf{y}_i^c$ refer to a given set of feature embeddings at layer i and pooled with window size c, and let $\mathbf{s}_i^c$ refer to a given set of layers in stage i of a model and who's layer features have each been pooled with window size c.

The results of this experiment for both the CIAFR10 and ImageNet64x64-50 using average and max pooling across different pooling window sizes are shown in Figure 6.6. Similar to the figure in the last section, the base intra-NNLS matrix is shown on the left for both datasets in Figure 6.6. For all other inter-NNLS matrices in Figure 6.6 the y-axis represents the embeddings of the base unpooled features, and the x-axis represents the embeddings for the pooled features. The window sizes used for each of the pooled layers are displayed along the top of each column.

Looking at the inter-NNLS matrix for CIFAR10 2x average pooling we see a stark decrease in similarity displayed in the CIFAR10 self-layer diagonal. The diagonal ones found in the base matrix have been replaced by a convex like curve. Starting at layer 0 we see an immediate decrease to 0.86 similarity. The self-layer similarity continues to decrease hitting a global minimum (along the diagonal) of 0.33 at layer 5 from which it begins to

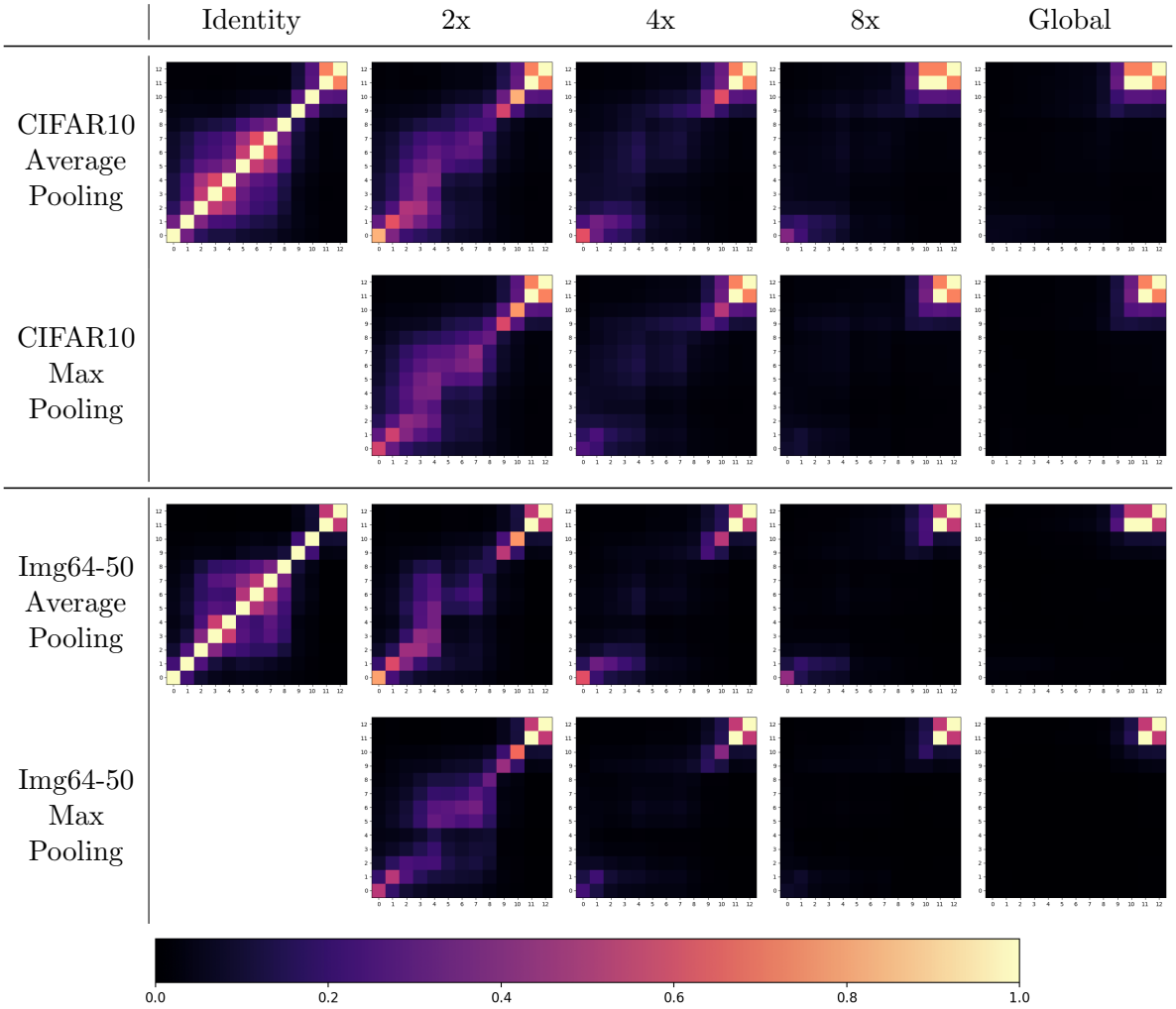


Figure 6.6: Inter-NNLS matrices computed from the latent embeddings of pooled features. In each figure the x-axis is for a model at the indicated pooling window size, and the y-axis is for the unpooled model. The top half of the figure shows inter-NNLS matrices for the CIFAR10 dataset, and the lower half of the figure shows inter-NNLS matrices for the ImageNet64x64-50 dataset. For each dataset there is a base inter-NNLS matrix without pooling applied, and a matrix for 2x, 4x, 8x, and global pooling windows for both average pooling and max pooling.

increase back towards 1.0. At layer 11 (i.e., a layer with no spatial dimension) the similarity once again equals 1.0.

The similarity for layers 9 and 10 experience a larger preserved similarity compared to layer 8. Throughout this work we have noticed that while layers 8, 9, and 10 have the same feature dimension they have behaved differently across a host of RRA experiments. MS-RRA has shown that layer 9 is where the model begins to remove spatial aspects from the feature vectors. Looking at 4x average pooling for layers 8 to 10 (i.e., stage 3) we see $\mathbf{y}_8^4$ has negligible similarity compared to $\mathbf{y}_8^1$, but $\mathbf{y}_9^4$ and $\mathbf{y}_{10}^4$ still maintain meaningful similarity with their respective unpooled features. Once 8x pooling is applied only $\mathbf{y}_{10}^8$ has a non-negligible similarity with its unpooled features. From this set of observations we hypothesize that layer 9 and layer 10 have each learned a form of 2x pooling (but not necessarily average pooling). Remember that the layers in stage 3 have a spatial feature size of 8 by 8. To make the transition between 8 by 8 the ResNet architecture includes a global average pooling operation (i.e., layer 11). But MS-RRA shows direct evidence that the model does not solely rely on the forced mapping to remove spatial information but instead preemptively removes spatial information.

Moving back to 2x average pooling, comparing the off-diagonal regions between stage 1 and stage 2, we see that the similarities between s_1^2 and s_2^1 experience a small decrease whereas the similarities between s_1^1 and s_2^2 drop to near zero. The large decrease between s_1^1 and s_2^2 is expected as these two groups of features have a 4x spatial difference in spatial resolution. What is surprising is the negligible change between s_1^2 and s_2^1. While these sets of features now share the same spatial resolution, the features in s_2^1 have gone through several additional layers in the model. We hypothesize that the increase in similarity indicates that there are spatial features relevant to the model that are either not detectable (or not as detectable) or not operated on by the model until the features are at the appropriate scale. Keep in mind that the convolution kernels in each layer are 3 by 3 and that each layer contains 2 convolution operations. Thus, the model is fully capable of learning to detect and operate on these larger spatial features. But, instead of changing the feature's latent position it is preserving the larger spatial features until stage 2. This aligns with the model primarily filtering out irrelevant information within stage 1 of the model, as observed in Chapter 4. The same increase is also experienced between the pooled features of stage 2 and unpooled features in the early layers of stage 3.

Interestingly, the features $\mathbf{y}_1^1$ are more similar to the features in s_1^2 than to s_1^1 by an average of 0.05. These increases in similarity are the only instance of an increase in similarity in the lower diagonal of the inter-NNLS matrix. The increase in similarity gives direct evidence that the model is primarily acting on small local features throughout stage 1. If this was not the case then average pooling would not provide a boost to similarity.

We suspect that future unpooled layers do not experience an increase in similarity with downstream pooled layers because the complexity of the feature detectors changes from basic edge and line detectors to more complex object detectors.

As the pooling window scale increases to 4x most similarity between layers approaches 0. The exception is the initial layers and the final layers without any spatial resolution. At 8x this trend continues and with global average pooling all similarity between all but the final layers are gone. We know from the intra-NNLS matrices in Figure 6.5 that each of the spatial pooled sets of features still has inherent structure. But importantly, we now know that the structure in each of the pooled representations is generally different from the representation of the unpooled features, where the larger the pooling window is the greater the difference between representations. Looking at max pooling for CIFAR10 there is only a minor difference in the change in similarity between the unpooled features and the max pooled features compared to average pooling.

Overall, the inter-NNLS matrices for ImageNet64x64-50 follow similar trends as the pooling window is increased for both average pooling and max pooling, including: 1) the asymmetry in similarity between stages 1 and 2, 2) the increased similarly of the unpooled layer 1 with the averaged pooled stage 2 layers for 2x average pooling, 3) the lingering similarity in layers 9 and 10, and 4) the complete fall off of similarity in the spatial layers for 4x, 8x, and global pooling. Looking at the 2x pooling window size for ImageNet64x64-50 inter-NNLS matrices we observe a distinct difference between average pooling and max pooling. For average pooling there is a distinct drop off in similarity between the pooled and unpooled stage 2 layers, but for max pooling this drop off in similarity is instead experienced between the pooled and unpooled stage 1 layers. We expect the root cause of this difference is related to stage 1 focusing on the removal of localized features (e.g., noise) for which average pooling would naturally ignore, and stage 2 focusing on detecting more discriminate features for which max pooling would naturally preserve but average pooling would remove.

6.7 Discussion

The experiments above demonstrate that reducing the feature map size via pooling has an effect on a models' RR. In general, ID, NNCS, NNLS, and the feature map size are relatively correlated with one another, where ID decreases, NNCS increases, and the similarities in the various NNLS matrices decrease as the dimensionality of the feature maps are reduced through pooling. For smaller reductions in feature map size the RRA metrics do not make major deviations from the trends of their respective identity curves.

But as the pooling magnitude increases many trends of these identity curves gradually or dramatically disappear, potentially being replaced by new trends that are specific to these lower granularity feature maps. The one exception is with the inter-NNLS matrices where the similarity between pooled and unpooled feature map latent structure is largely affected by any pooling. The relative invariance to low levels of pooling indicates that one does not need to use the full feature map size to gain actionable information from the RRA metrics, and that this holds for both CIFAR10 and ImageNet64x64-50.

Even though RRA may be performed reliably at a lower spatial resolution there are still many insights one can gain from performing MS-RRA. For ID we were able to plot the model's transition from focusing on local features to focusing on global features. Specifically, by comparing the unpooled ID curve to a globally pooled curve we are able to separate the effects of the number of features (or channels) that a layer may have, and to what degree the model is processing local features verses global features. While a model's focus on gradually more abstract features is known, the ID curves in MS-RRA directly show where the focus on these types of features changes. For NNCS we showed the local separability changes as one considers different spatial resolutions at each layer in a model. Importantly, we showed that as one considers larger spatial regions (through larger pooling windows) the local separability between classes increases, and that there is an optimal pooling window size of maximum separability depending on the layer's depth in the model. Using such information has the potential to improve a model's performance by introducing a larger spatial content earlier in a model's layer progression (we explore this in Chapter 7).

When we performed intra-NNLS analyses we saw that each pooling scale (with diminishing returns) preserved a similar block like structure for each of the ResNet stages. By considering both average pooling and max pooling we were able qualitatively detect behavioral differences between several of the stages. From multi-scale average pooling we know that the average structure between consecutive layers is the same regardless of spatial resolution. From multi-scale max pooling we know that the structure of the most dominate features is stable at higher spatial resolutions (smaller pooling window), but with the structure changing drastically at lower spatial resolutions. By comparing pooled and unpooled representations of the same model through inter-NNLS we were able to show that the structure of the latent embeddings is not the same. The larger the pooling window applied the greater the difference with the base latent representation. The fact that the structure is not the same tells us that a model may benefit from using this information through explicitly adding additional pooling operations to the ResNet model. When we combine this observation with the increased NNCS curves when pooling is applied, we have strong evidence that using pooled information in a classifier could improve a model's performance. One of the exceptions to this differing structure in scale was at layers 9 and

10. By using multi-scale inter-NNLS we showed that both layers 9 and 10 were partially invariant to pooling operations, indicating that they are directly learning some sort of implicit pooling operation that isn't explicitly average or max pooling.

Chapter 7

Spatial Transformed Attention Condensers

In the last chapter MS-RRA revealed to what degree different spatial resolutions of features result in more coherent class groupings as measured by NNCS. In general, the larger the pooling window used on a sample's features the closer the sample is to samples from the same class. This observation holds for any pooling window size across most CNN layers. However, peak class similarity for each layer often occurs with a pooling window that is smaller than a global pooling window size. In addition, we observed that the latent structure of pooled features are inherently different from unpooled features. In this chapter we propose spatial transformed attention condenser (STAC) modules, a specific instantiation of attention condensers [82], which is designed to take advantage of the heightened class similarity observed at specific spatial resolutions.

Designing targeted CNN architectures is a common method for achieving desired trade-offs between performance and efficiency requirements. Adding self-attention modules can be an efficient method for improving the performance-efficiency trade-offs of a deep neural network [82]. Knowing which self-attention module to use is often hard to determine and can require extensive hyperparameter testing. Using insights gained from MS-RRA we propose the Spatial Transformed Attention Condenser (STAC) module, a novel attention-condenser based self-attention module. We show that adding STAC modules to ResNet style architectures can result in up to a 1.6% increase in top-1 accuracy compared to vanilla ResNet models and up to a 0.5% increase in top-1 accuracy compared to SENet models on the ImageNet64x64 dataset, at the cost of up to 1.7% increase in FLOPS and 2x the number of parameters. In addition, we demonstrate that results from MS-RRA analysis can

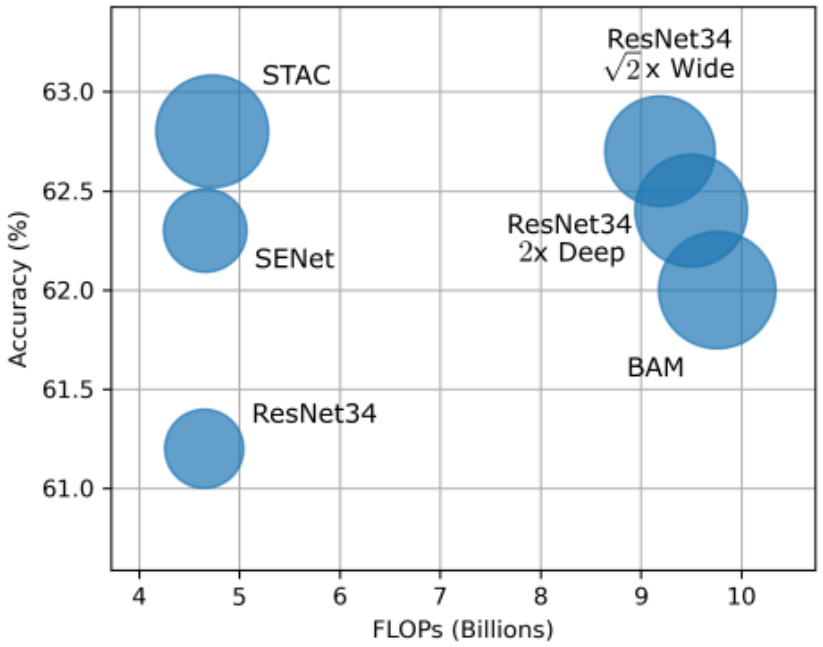


Figure 7.1: A comparison between six ResNet34 based architectures trained on the ImageNet64x64 dataset. The proposed STAC based model achieves the highest performance while requiring a fraction of FLOPS and a similar number of parameters (proportional to the associated circle area) compared to the next closest model. The precise numbers can be found in Table 7.1. The STAC, SENet, and BAM models in this figure use standard module placement.

be used to select an effective parameterization of the STAC module resulting in competitive performance compared to an extensive parameter search.

7.1 Spatial Transformed Attention Condenser (STAC) Modules

The proposed STAC module is constructed as follows. Given an input feature map Y^i, a condenser operation $Q^i = P(Y^i)$ spatially reduces the size of Y^i. Then a set of attention operations A identifies the important regions of Q^i and produces an attention map $K^i = A(Q^i)$. An expansion layer E spatially increases the attention map K^i to a full-focus

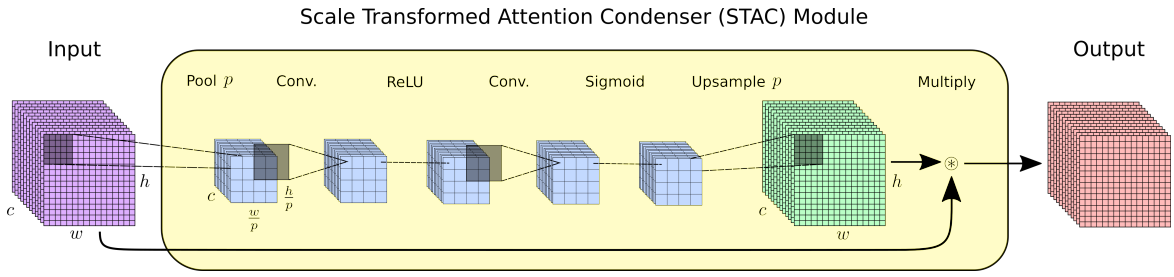


Figure 7.2: An illustration of a Scaled Transformed Attention Condenser (STAC) Module.

self-attention map $T^i = E(K^i)$ which matches Y^i's spatial resolution. Finally, the full-focus self-attention map T^i is element-wise multiplied with Y^i to output the spatially transformed attention feature map $Y^j = Y^i \times T^i$. The STAC module is depicted in Figure 7.2.

In this work P is an average pooling operation where the stride size and pooling window are equal, A is multi-layer sub-network with convolution, ReLU, convolution, sigmoid operations sequentially applied, and E is a nearest neighbour upsampler. Conceptually the proposed STAC module follows a similar construction to a sequence and excitation network (SENet) but with the condenser operation having a limited window size and with, the addition of an expansion operation to match the input feature map size. We hypothesize that the structural bias designed into the STAC module will allow a CNN to better make use of the intrinsic spatial features found at specific spatial resolutions within a dataset, and will result in improved model performance with negligible increase in computation.

We explore two different placement locations of the STAC module, following the notation used in (Hu et al.) [43], 1) *standard* placement where the STAC module is placed after the residual block but before the skip connection, and 2) *post* placement where the STAC module is placed after the skip connection.

7.2 Results

We now compare CNNs using the proposed STAC module to models using other self-attention modules as well as standard ResNet models adjusted to have a similar number of FLOPS or parameters. The other self-attention modules include SENet [43] modules and BAM [61] modules (note that we use the module design but not the macro-architecture placement of these modules as presented in (Park et al.) [61]). Each of the three self-attention modules are tested in both the *standard* and *post* placement stategies described in (Hu et al.) [43] (see Figure 7.3). For the STAC module we use a pooling window of 8

Table 7.1: Model Comparison - Top-1 Accuracy, FLOPS, Number of Parameters

Model	ResNet20 - CIFAR10			ResNet34 - ImageNet64x64-50			ResNet34 - ImageNet64x64		
	Top-1 Acc.	**FLOPS**	**Param.**	**Top-1 Acc.**	**FLOPS**	**Param.**	**Top-1 Acc.**	**FLOPS**	**Param.**
Base [39]	89.9 ± 0.2	41.5M	272K	76.8 ± 0.5	4.65B	21.3M	61.2 ± 0.1	4.65B	21.8M
Base - 2x-Deep	90.7 ± 0.5	85.7M	564K	76.5 ± 0.5	9.50B	44.0M	62.4 ± 0.3	9.50B	44.4M
Base - $\sqrt{2}$x-Wide	**91.2** ± 0.2	78.5M	522K	**77.4** ± 0.6	9.07B	41.6M	**62.7** ± 0.1	9.19B	42.6M
SENet [43] - Standard	90.5 ± 0.2	41.7M	305K	77.6 ± 0.5	4.66B	23.8M	62.3 ± 0.2	4.66B	24.3M
BAM [61] - Standard	90.5 ± 0.2	86.6M	614K	75.2 ± 0.7	9.76B	47.7M	62.0 ± 0.1	9.76B	48.2M
STAC - Standard (Ours)	**90.6** ± 0.2	42.3M	563K	**77.7** ± 0.4	4.73B	43.9M	**62.8** ± 0.1	4.73B	44.4M
SENet [43] - Post	90.4 ± 0.2	41.7M	305K	75.7 ± 0.5	4.66B	23.8M	61.7 ± 0.2	4.66B	24.3M
BAM [61] - Post	**90.7** ± 0.3	86.6M	614K	*	9.76B	47.7M	*	9.76B	48.2M
STAC - Post (Ours)	90.4 ± 0.2	42.3M	563K	**76.6** ± 0.4	4.73B	43.9M	**62.2** ± 0.2	4.73B	44.4M

*In these configurations the BAM module based models were unstable with the majority of the runs remaining near random performance.

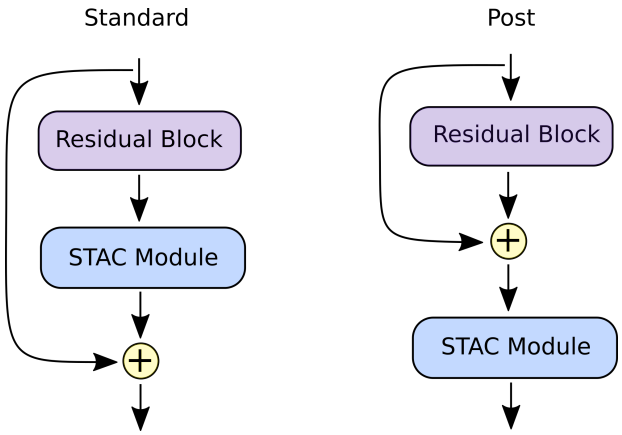


Figure 7.3: Examples of STAC module placement.

(since it provides the highest overall CS scores as observed in Figure 6.3a). For the BAM module we use a dilation factor of 4 (for a similar receptive field as the STAC module) and a compression factor of 4. The standard ResNet models include a *vanilla* ResNet model, a *2x-deep* ResNet model, and a $\sqrt{2}$-*wide* ResNet model.

We compare the performance of these nine different models on three model-dataset pairs, including: ResNet20-CIFAR10, ResNet34-ImageNet64x64-50, and ResNet34-ImageNet64x64, where the ResNet34 model is the normal size model from (He et al.) [39]. Note that the final model-dataset pair is trained on the entire 1000 class ImageNet64x64 dataset [12]. The first model is trained for 100 epochs, a base learning rate of 0.1, and batch size of 128. The second model is trained for 100 epochs, a base learning rate of 0.01, and a batch size of 128. The third model is trained for 40 epochs, a base learning rate of 0.01, and a batch size of 256. The learning rates were selected to maximize the performance of the base ResNet model configuration for each dataset. All model configurations use the same set of crops, flip, rotation, colour jitter, grayscale, and blur augmentations as shown in Table 4.2. Results comparing the accuracy, FLOPS, and number of parameters of these models is shown in Figure 7.1. The accuracy results for each model are an average of 8 training runs.

In all cases the use of the STAC-Standard module resulted in improved performance over the base model by 0.7%, 0.9%, and 1.6% on the CIFAR10, ImageNet64x64-50, and ImageNet64x64 datasets, respectively, at a small increase to FLOPS but over double the number of parameters. For CIFAR10 both the 2x-deep and $\sqrt{2}$x-wide models outperformed the STAC-standard module, but both models require at least 1.85x more FLOPS with a similar number of parameters. However, for ImageNet64x64-50 and ImageNet64x64 the STAC-Standard module outperformed both variants of the base model with similar

differences in FLOPS and parameter count. Amongst the self-attention modules using standard placement the STAC module outperformed the other two designs on all three datasets. On CIFAR10 and ImageNet64x64-50 the STAC module narrowly outperformed SENet, while on ImageNet64x64 the STAC module outperformed SENet by 0.5% at the cost of 1.5% additional FLOPS and almost 2x the number of parameters.

In all but one case all module variants using post placement underperformed standard placement, with the one exception being the BAM-post module trained on CIFAR10. In this instance the BAM-post module outperformed all other self-attention configurations but still underperformed the $\sqrt{2}$x-wide base model variant. All other configurations using the BAM-post module were unstable during training using the selected training parameters. As a reminder, the training parameters used in this experiment were selected to optimize the respective base ResNet models. Overall, the highest performing model for CIFAR10 was ResNet20 $\sqrt{2}$x-wide, and for ImageNet64x64-50 and ImageNet64x64 was the ResNet34 models using the STAC module in standard placement.

7.3 Ablation Results

Above it was shown that the STAC module can increase the performance of a variety of ResNet model sizes trained on datasets with different properties at the cost of a small increase in FLOPS and doubling of the number of parameters compared to standard ResNet models. In this section we perform a series of ablation tests on STAC based models. Each experiment is performed on a ResNet20 model trained on either CIFAR10 of ImageNet64x64-50 using the same training parameters from Chapter 4.1 with augmentations. The tests in this section are explored using only the *standard* location of the STAC module. Unless stated otherwise, the default configuration for all ResNet20 models in this section use STAC modules, standard placement, and with a condenser window size of 8. All results in this section are from 8 independently trained models.

7.3.1 Self-Attention Receptive Field

The condenser operation (average pooling) used to reduce the spatial feature size for the downstream self-attention operations has the consequence of increasing the self-attention operation's receptive fields; namely the two convolutional operations in each STAC module. The 3x3 spatial shape of these kernels further expands the STAC module's receptive field such that each feature in the output attention map is dependent on a larger portion of the

original input feature map. Table 7.2 shows the results of changing the spatial feature size of both the first convolution kernel $C1$ and the second convolution kernel $C2$ from a 3x3 kernel to a 1x1 kernel for all modules in the ResNet20 model.

Table 7.2: STAC Kernel Size - ResNet20

Dataset	C1	C2	Top-1 Acc.	FLOPS	Params.
CIFAR10	1	1	90.3 ± 0.2	41.7M	305K
	1	3	90.4 ± 0.1	42.0M	434K
	3	1	90.6 ± 0.3	42.0M	434K
	3	3	90.6 ± 0.2	42.3M	563K
Img64-50	1	1	73.3 ± 0.4	167M	308K
	1	3	73.4 ± 0.5	168M	437K
	3	1	73.6 ± 0.6	168M	437K
	3	3	73.6 ± 0.5	169M	566K

A kernel configuration of $(1, 1)$ has marginally more FLOPS but the same number of parameters as a model using SENet modules and performs within ±0.3% of the respective SENet modules (see the respective GAP rows in Table 7.4). For both datasets the kernel configuration $(3, 1)$ showed marginal performance benefits over $(1, 3)$, with $(3, 1)$ matching the average performance of configuration $(3, 3)$ but with marginally higher standard variation. For both datasets the configurations $(1, 3)$ and $(3, 1)$ are attractive options to use in a model given the limited performance implications while significantly reducing the number of parameters.

7.3.2 Location of STAC Module

As shown in Chapter 6 the stages of a ResNet model have different effects on the increase (or decrease) of the observed NNCS. We now explore to what degree (if any) each stage of a ResNet model benefits from using the STAC module, as well as the associated computational costs of each configuration. Table 7.3 shows the results for a normal ResNet20 model, when the layers in only a single stage of a ResNet20 use the STAC module, and when all layers in a ResNet20 model use the STAC module.

For CIFAR10 the STAC module provides a performance benefit when applied to all layers individually and has the most impact when applied to stage 2, falling 0.2% short of the complete configuration. Stages 1 and 3 provide accuracy gains even through the configuration

Table 7.3: STAC Stage Location - ResNet20

Dataset	S1	S2	S3	Top-1 Acc.	FLOPS	Params.
CIFAR10				89.9 ± 0.2	41.5M	272K
	×			90.1 ± 0.2	41.8M	286K
		×		90.4 ± 0.3	41.8M	328K
			×	90.1 ± 0.2	41.8M	494K
	×	×	×	90.6 ± 0.2	42.3M	563K
Img64-50				71.6 ± 0.5	166M	275K
	×			71.7 ± 0.6	167M	289K
		×		72.2 ± 0.7	167M	330K
			×	73.0 ± 1.0	167M	496K
	×	×	×	73.6 ± 0.5	169M	566K

requires significantly more parameters. Using STAC modules with ImageNet64x64-50 shows a different pattern where the deeper the stage the more impact the STAC modules provides. Unlike with CIFAR10, when trained on ImageNet64x64-50 no single stage configuration approaches the performance of using the STAC module throughout the model.

7.3.3 Condenser Size

The size of the condenser window used in Section 7.2 (i.e., $P = 8$) was derived through observations made from the NNCS curves in Section 6.2. We now explore a wider range of condenser windows sizes to see how this choice affects the computational tradeoffs of using the STAC module. Table 7.4 shows the result of changing the condenser window size from 1 (i.e., no pooling) to using global average pooling on the entire input feature map (i.e., resulting in SENet).

Most STAC module windows sizes for both datasets perform near peak performance. For CIFAR10 all window sizes achieve within 0.2% of peak performance, even though the number of FLOPS is significantly increased for smaller condenser window sizes (i.e., the feature maps in the STAC module are larger). The highest performing window size is 1, which requires over double the number of FLOPS as using GAP pooling. However, for ImageNet64x64-50 the top performing window sizes fall between 2 to 16, with the larger window sizes requiring less FLOPS but with greater variation in performance between runs. Both the smallest and largest window sizes under perform by at least 0.5%.

Table 7.4: STAC Condenser Size - ResNet20

Dataset	P	Top-1 Acc.	FLOPS	Params.
CIFAR10	1	90.7 ± 0.3	84.2M	563K
	2	90.6 ± 0.2	52.3M	563K
	4	90.5 ± 0.2	44.3M	563K
	8	90.6 ± 0.2	42.3M	563K
	GAP	90.5 ± 0.2	41.7M	305K
Img64-50	1	73.1 ± 0.5	337M	566K
	2	73.7 ± 0.2	209M	566K
	4	73.6 ± 0.2	177M	566K
	8	73.6 ± 0.5	169M	566K
	16	73.6 ± 0.6	167M	566K
	GAP	73.0 ± 0.5	167M	308K

7.3.4 Intervention Strategies

It was shown in Section 7.3.2 that the STAC module location is an important consideration in performance tradeoffs, and in Section 7.3.3 that various condenser sizes provide comparable accuracy. We now consider three different strategies for selecting the condenser window size used for any given stage in a ResNet model. The first approach uses a *greedy* strategy where we repeat the stage location experiment from Section 7.3.2 but for every viable window size (Table 7.5), then each stage is assigned the highest performing window size for each respective stage. The second strategy assigns the window size corresponding to the *maximum CS* observed in Figure 6.3a for each respective dataset. The third strategy uses the largest possible uniform window size (i.e., the feature map spatial length in the final ResNet stage). The window size, accuracy, and FLOPS for these three strategies are shown in Table 7.6.

Overall, each of the placement strategies results in similar performance. For CIFAR10 the greedy strategy produced the best accuracy, costing an additional 13M FLOPS, while the three strategies for ImageNet64x64-50 demonstrated comparable performance. For both datasets the greedy strategy followed a similar progression of window sizes, with the first stage using the largest window size and getting progressively smaller as the stages progress. The maximum NNCS strategy favoured similar window sizes across all three stages for both datasets, with the last stage using the equivalent of global pooling.

Table 7.5: STAC Optimal Intervention - ResNet20

Dataset	S1	S2	S3	1	2	4	8	16	32	64
CIFAR10	×			90.1	90.0	90.1	90.1	**90.2**	**90.2**	
		×		90.0	90.1	90.1	**90.4**	90.2		
			×	**90.4**	90.2	90.2	90.1			
Img64-50	×			71.6	71.9	72.0	71.7	72.0	**72.3**	71.9
		×		71.8	71.8	**72.4**	72.2	72.3	72.1	
			×	72.4	**73.0**	72.7	**73.0**	72.6		

Table 7.6: STAC Parameterization Selection Strategies - ResNet20

Dataset	Strategy	S1	S2	S3	Top-1 Acc.	FLOPS
CIFAR10	Greedy	16	8	1	90.8 ± 0.2	56.0M
CIFAR10	Max CS	8	4	8	90.5 ± 0.2	43.0M
CIFAR10	Max Uni.	8	8	8	90.6 ± 0.2	42.3M
Img64-50	Greedy	32	4	2	73.5 ± 0.6	184M
Img64-50	Max CS	8	8	16	73.4 ± 0.4	168M
Img64-50	Max Uni.	16	16	16	73.5 ± 0.5	167M

7.4 Discussion

In this chapter using the observations gained through MS-RRA we proposed STAC modules (Section 7.1), a self-attention model designed to improve a CNN's ability to learn class-based feature detectors. We demonstrated that this evidence based self-attention design results in improved performance in general over base models, outperforms other self-attention modules in most cases, and who's performance gains required less additional FLOPS compared to larger variants of base models for similar performance gains (Section 7.2).

We investigated a large assortment of STAC module parameterizations. Section 7.3.1 showed that using a more limited receptive field within a self-attention model can achieve performance on par with a SENet style module that uses GAP sized attention windows. In addition, it was shown that connecting the local fields with $3x3$ convolutions, to form *medium* size receptive fields, is likely to be beneficial for model performance. In Section 7.3.2 the results were inconclusive as to which stage in a ResNet model would most benefit from the STAC module, but regardless, the use of the STAC module improved model performance over a vanilla ResNet model. In addition, it was demonstrated that using

STAC modules throughout a model consistently results in the overall best performance compared to targeted placement. Section 7.3.3 showed that the condenser size used for the STAC modules is forgiving with regards to performance, and that the computational cost drops off quadratically with the window size. As such choosing a middle of the road window size is likely a reasonable strategy for an initial parameter selection. Section 7.3.4 further supported this result as three unique selection strategies were explored, and all resulted in performance within 0.3% of each other. However, this finding only holds if the STAC module is used throughout a model. If limited placement is used, then specific window sizes demonstrate superior performance depending on the stage.

Overall, we found that the performance gains from using the STAC module to be relatively robust in the selection of hyperparameters when the STAC module is used throughout a model. In general, to see desirable performance increases we recommend that STAC modules should be used throughout a model with condenser window sizes chosen with the aid of MS-RRA. For CIFAR10 and ImageNet64x64-50, this range is approximately between 4 to 16. Under these conditions our experiments demonstrate that STAC module is a viable building block for ResNet based classification CNNs.

Chapter 8

Conclusions and Future Directions

This chapter summarizes the contributions within this thesis, discusses the limitations of the proposed methods, and presents future areas of investigation.

8.1 Summary of Contributions

8.1.1 Representational Response Analysis Framework

In Chapter 3 we presented a novel analytic framework, called representational response analysis (RRA), for analyzing the latent structure within a CNN. To the best of our knowledge RRA is the first multifaceted framework designed to jointly study the latent structure of a CNN in a computationally efficient manner. The proposed RRA framework models the relationships between the latent embeddings of the data samples at each layer in a CNN through the use of a kNN graph. The proposed RRA framework uses three core metrics to study the complexity and performance of a model at each layer, and a similarity metric to compare the structure between layers. For the complexity metric we use the TwoNN intrinsic dimensionality estimator, for the performance metric we proposed NNCS to measure the average class similarity of each sample to its neighbours, and for the similarity metric we proposed NNLS to measure the average neighbour similarity between layers. Each of these metrics was designed to be computationally efficient to compute from the underlying set of kNN graphs.

In Chapter 5 we study aspects of the kNN hyper-parameterization on the three RRA metrics. We discovered that one can significantly reduce the size of the kNN by limiting both

the number of samples used to 1) construct the kNN and 2) the number of neighbours per sample in the kNN, while still being able to detect most of the relevant latent information provided ratio between these two values is kept constant. These insights will allow researchers to apply the RRA framework on larger model-dataset pairs, more quickly gain insights into their model designs, and to more quickly iterate on these CNN model designs.

8.1.2 RRA and Applying Augmentation During Training

In Chapter 4 we demonstrate the utility of the RRA framework by studying the effects of applying augmentations during training to a CNN's latent representations. We choose this problem as applying augmentations to improvement a CNN's performance is one of the most basic and common place design choices a researcher can make. However, studying the effects of augmentations is typically limited to studying the effect on CNN classification performance, or external behavioral traits (e.g., characterizing the types of mistakes a model will make). By using the RRA framework we demonstrated that studying the interactions between CNN design choices and the internal latent characteristics of a CNN can be a powerful tool for better understanding the impacts of specific design choices. For using augmentations during training our primary contributions were showing strong evidence that CNNs trained with augmentations will remove more irrelevant features early on in the shallower layers and allows for the deeper layers to focus more on class discrimination.

8.1.3 MS-RRA

In Chapter 6 we introduced multi-scale representational response analysis (MS-RRA) and studied the impacts of spatial pooling the feature maps for each layer prior to kNN calculation. By doing so we were able to study the impacts of using a coarser spatial resolution on our ability to detect changes in a CNNs latent representations. Four novel observations from our MS-RRA include: 1) some spatial pooling can be applied to the latent feature maps without a major loss of precision in the RRA metric (with the exception of inter-NNLS), 2) there exists an optimal scale at which samples are most class-wise similar to one another, 3) that the peak scale at which samples are most class-wise similar changes throughout a model, and 4) that the final layers in a ResNet20 model naturally learn to pool features before the CNN enforces it through a global average pooling operation. The implications of these observations are that (1) RRA can be performed with less computations, (2 and 3) it is beneficial to build modules into a model to make use of the elevated NNCS, and (4) it may be better to manually enforce pooling instead of having the model learn pooling.

8.1.4 Spatial Transformed Attention Condensers

In Chapter 7 we designed STAC modules to take advantage of our findings from Chapter 6 that showed that pooled features maps have improved class clustering throughout most a of CNN, and that the pooled features have a different latent structure compared to unpooled feature maps. We hypothesized that building a network architecture to take advantage of these finds from MS-RRA would result in improved model performance. To that effect, the proposed STAC modules provide a computationally efficient attention mechanism to each layer through the use of pooling operations. We show that adding STAC modules to ResNet style architectures can result in up to a 1.6% increase in top-1 accuracy compared to vanilla ResNet models and up to a 0.5% increase in top-1 accuracy compared to SENet models on the ImageNet64x64 dataset, at the cost of up to 1.7% increase in FLOPS and 2x the number of parameters. In addition, we demonstrate that results from MS-RRA analysis can be used to select an effective hyper parameterization of the STAC module resulting in competitive performance compared to an extensive hyper parameter search. A key takeaway from this chapter is that RRA can be an effective tool to guide the improvement of a CNN.

8.2 Limitations

One of the limitations in this work is the sole focus of applying RRA to image classification CNNs. In general, RRA is not inherently restricted to just classification models. For most fixed feature length model types (e.g., object detection, tabular data) the calculation of the kNNs, ID, and NNLS would be unaffected. Just NNCS would have to be modified for non-classification only models. For variable length models (e.g., sequence-based tasks) RRA would need to be extended to be able to handle comparing feature maps of different sizes.

Despite the computational benefits of using a common kNN model on downstream metrics, the upfront calculation of the kNNs is still computationally expensive on two fronts. The calculation of each data sample's feature maps for every layer of interest requires a forward inference pass of a CNN and the storage of the features. Within this work we used a maximum image size (for ImageNet64x64-50) of $(64, 64, 3) = 12288$ features, and the largest ResNet20 feature map size of $(64, 64, 32) = 131072$. If we were to perform RRA on the full ImageNet dataset at full resolution on a standard ResNet50 the storage required for all feature maps for a single layer would reach over 100 terabytes of disk space. Even on enterprise servers this is not a trivial number of resources to occupy. The other challenging factor is the distance comparisons required to calculate the kNNs. In this work we used a brute force approach which was only feasible given the relatively small number of

samples (as many as 50000). Applying a brute force approach on over one million samples in ImageNet at full resolution would require a massive amount of expensive computers to complete in any reasonable amount of time (considering that the brute force is $\mathcal{O}(n^2)$).

In this work our application of RRA has been limited to primarily investigating the latent structure of the output of ResNet blocks. Each of these blocks contains several internal operations, including: convolutions, batchnorms, the addition of two feature maps, and ReLU activation functions. The output block of each ResNet block is a ReLU activation function. While convenient, the choice of investigating only the output of ResNet blocks is arbitrary and was chosen in this work as it is the main point of focus for most other other latent embedding analysis methods. The focus on the output of the blocks and specifically on ReLU activation functions has the potential to bias the analysis and obfuscated other internal aspects of the ResNet blocks. Specifically, it was noted in [23] that the internal layer operations can behave in unique manners despite the global consistency between layers. Applying RRA to the internals of a layer could potentially provide valuable insight.

8.3 Future Work

The proposed RRA framework is highly versatile. As such, we were only able to explore a limited subset of potential uses in this work. Below we present a small list of potential areas of future research for which we believe RRA would provide useful insights if applied. We also present potential improvements and extensions to the proposed RRA framework.

8.3.1 Applications of RRA

For this work we used RRA in two primary ways, first to investigate the effects of applying augmentation during training, and second to investigate the similarity of a CNN's feature maps at different spatial resolutions. However, the proposed RRA framework has the capability to be applied is a wide range of A/B testing scenarios with regards to CNN model construction. Such possibilities include a more in-depth analysis of specific augmentations, when during a model's training process the model learns specific discriminative abilities, and differences between model architectures beyond ResNets such as with transformer models [19, 78].

For data augmentation we only explored the effects of augmentation as an all or nothing approach, but RRA allows one to perform a more detailed investigation of the effects of each individual augmentation, and the consequence of specific augmentation parameter

selection. In addition, one would be able to investigate the specific locations within a CNN where a model becomes invariant to each augmentation. Doing so would provide a model designer with the required insight to adjust a model's architecture or training procedure to have specific behaviour with respect to data invariances.

Of particular interest is the recent advances in training CNNs using self-supervised learning. Such approaches include pretext learning [16, 24], contrastive methods [9, 37], non-contrastive methods [4, 10, 86], and semi-supervised methods [48]. Some research has been done to better understand how the latent representations a self-supervised CNN differ those of a supervised CNN [33, 71, 79]. However, such efforts are limited in scope in similar ways that supervised models have been studied in isolation. Future work will include applying RRA to better understand the differences between the latent embeddings learned from supervised and self-supervised training methods.

A largely ignored line of research for classification models is the study of the dynamics of within a model's latent space during the training process. Some approaches choose to study the flatness or sharpness of the loss landscape by interpolating between two sets of weights at different points during training [30, 53]. With the RRA framework one can instead perform RRA at different points during training and measure the change observed within any given RRA metric. By doing so one could design a tailored learning algorithm for the problem at hand.

8.3.2 Improvements to RRA

Future work for improvements to the core RRA framework will revolve around making RRA more tractable for larger CNN model-dataset pairs. The first set of improvements will include making the computation of the underlying kNN more efficient. In this work we employed a brute force approach while calculating the kNN. The brute force approaches is $\mathcal{O}(n^2)$ with respect to the number of samples in the dataset. One approach for approximating a kNN is called nearest neighbour descent proposed by Dong *et al.* [18], and only costs approximately $\mathcal{O}(n^{1.14})$. The nearest neighbour descent algorithm achieves between 0.89 to 0.99 percent recall on the datasets they investigate. We would begin our analysis by measuring how accurate the nearest neighbour descent algorithm is on CNN feature maps, then continue on to measure the impacts of the approximations on the RRA metrics. If successful then there are additional tricks that can be employed to further reduce the kNN calculation time, such as providing the nearest neighbour descent algorithm a good starting point by making use of previously calculated kNNs from other layers of the CNN.

The second improvement to the core RRA framework will be to investigate potential

alternatives to the complexity estimator. Throughout this work we used the TwoNN estimator to approximate a layer's intrinsic dimensionality. However, we showed in Chapter 5 that the TwoNN estimator requires enough samples to be used for an accurate ID curve to be calculated across all layers in a CNN. The minimum required number of samples is higher than the other two RRA metrics. As such, a complexity measure that required less samples to be accurately calculated would allow RRA to be more rapidly used.

8.3.3 Extensions to RRA

Reducing the similarity between two layers (i.e., NNLS) to a single value provides a useful measure for high level model design decisions (e.g., layer removal). On the other hand, such reduction also removes most of the inter-sample relationship information, thereby reducing one's ability to study the complex interactions between layers throughout a network. While one cannot perform classical topological data analysis on the kNNs (due to limited number of neighbour distances being tracked), the kNN graphs used in RRA still allow for higher order relationships between samples to be studied. For example, it would be possible to study when pairs of samples become neighbours in a CNN, properties of the pairs while they are neighbours, and when pairs of samples are no longer neighbours. When considering the entire dataset using this approach one can see the interactions between layers. For example, aspects of a network like connection-cancellation would become evident (i.e., if one layer moves a lot of samples near each other and a down stream layer moves those samples apart). By studying how layers interact with each other on a more granular level (when compared to scalar layer similarity measures) one can tailor a CNN's design.

Another way to make use of the set of kNN graphs generated through RRA is for data visualization techniques such as TSNE [76] or UMAP [58]. Both of these techniques effectively project kNN graphs from high dimensional feature space to a lower dimensional space (typically 2D or 3D for visualization). Often these projections are used by researchers to gain an intuitive sense of what a model is doing and how data is clustering. Normally TSNE and UMAP operate on a single set of features, but with RRA we have a set of features (and corresponding kNNs) for each layer in a CNN. With this set of kNNs one could apply the data visualization methods at each layer in a model. We also hypothesize that one could create a meta-kNN of sorts, from which one could then apply TSNE or UMAP too. By doing so one could visualize a more representative relational image of how a model processes a set of data and not just how these approaches see the data in a single layer.

References

[1] Forest Agostinelli, Matthew Hoffman, Peter Sadowski, and Pierre Baldi. Learning activation functions to improve deep neural networks. *arXiv preprint arXiv:1412.6830*, 2014. 14

[2] Guillaume Alain and Yoshua Bengio. Understanding intermediate layers using linear classifier probes. *arXiv preprint arXiv:1610.01644*, 2016. 2, 18, 24, 29, 38, 53, 66

[3] Alessio Ansuini, Alessandro Laio, Jakob H Macke, and Davide Zoccolan. Intrinsic dimension of data representations in deep neural networks. In *Advances in Neural Information Processing Systems (NIPS)*, 2019. 2, 19, 23, 26, 37, 49, 52, 53

[4] Adrien Bardes, Jean Ponce, and Yann LeCun. Vicreg: Variance-invariance-covariance regularization for self-supervised learning. *arXiv preprint arXiv:2105.04906*, 2021. 100

[5] Yoshua Bengio, Jérôme Louradour, Ronan Collobert, and Jason Weston. Curriculum learning. In *Proceedings of the 26th Annual International Conference on Machine Learning (ICML)*, pages 41–48. ACM, 2009. 16

[6] David Berthelot, Nicholas Carlini, Ian Goodfellow, Nicolas Papernot, Avital Oliver, and Colin A Raffel. Mixmatch: A holistic approach to semi-supervised learning. In *Advances in Neural Information Processing Systems*, 2019. 16

[7] Gunnar Carlsson. Topology and data. *Bulletin of the American Mathematical Society*, 2009. 2, 19, 24

[8] Gunnar Carlsson and Rickard Brüel Gabrielsson. Topological approaches to deep learning. *arXiv preprint arXiv:1811.01122*, 2018. 19

[9] Ting Chen, Simon Kornblith, Mohammad Norouzi, and Geoffrey Hinton. A simple framework for contrastive learning of visual representations. In *International conference on machine learning*, pages 1597–1607. PMLR, 2020. 100

[10] Xinlei Chen and Kaiming He. Exploring simple siamese representation learning. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 15750–15758, 2021. 100

[11] François Chollet. Xception: Deep learning with depthwise separable convolutions. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2017. 12

[12] Patryk Chrabaszcz, Ilya Loshchilov, and Frank Hutter. A downsampled variant of imagenet as an alternative to the cifar datasets. *arXiv preprint arXiv:1707.08819*, 2017. 48, 89

[13] Ekin D Cubuk, Barret Zoph, Dandelion Mane, Vijay Vasudevan, and Quoc V Le. Autoaugment: Learning augmentation strategies from data. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2019. 15

[14] Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. Imagenet: A large-scale hierarchical image database. In *2009 IEEE conference on computer vision and pattern recognition*, pages 248–255. IEEE, 2009. 1, 14

[15] Terrance DeVries and Graham W Taylor. Improved regularization of convolutional neural networks with cutout. *arXiv preprint arXiv:1708.04552*, 2017. 15

[16] Carl Doersch, Abhinav Gupta, and Alexei A Efros. Unsupervised visual representation learning by context prediction. In *Proceedings of the IEEE international conference on computer vision*, pages 1422–1430, 2015. 100

[17] Diego Doimo, Aldo Glielmo, Alessio Ansuini, and Alessandro Laio. Hierarchical nucleation in deep neural networks. *Advances in Neural Information Processing Systems*, 2020. 19

[18] Wei Dong, Charikar Moses, and Kai Li. Efficient k-nearest neighbor graph construction for generic similarity measures. In *Proceedings of the 20th international conference on World wide web*, pages 577–586, 2011. 100

[19] Alexey Dosovitskiy, Lucas Beyer, Alexander Kolesnikov, Dirk Weissenborn, Xiaohua Zhai, Thomas Unterthiner, Mostafa Dehghani, Matthias Minderer, Georg Heigold, Sylvain Gelly, et al. An image is worth 16x16 words: Transformers for image recognition at scale. *arXiv preprint arXiv:2010.11929*, 2020. 99

[20] Elena Facco, Maria d'Errico, Alex Rodriguez, and Alessandro Laio. Estimating the intrinsic dimension of datasets by a minimal neighborhood information. *Scientific Reports*, 2017. 19, 23, 26

[21] Karl Pearson F.R.S. Liii. on lines and planes of closest fit to systems of points in space. *The London, Edinburgh, and Dublin Philosophical Magazine and Journal of Science*, 1901. 67

[22] Keinosuke Fukunaga and David R Olsen. An algorithm for finding intrinsic dimensionality of data. *IEEE Transactions on Computers*, 1971. 19

[23] Robert Geirhos, Kantharaju Narayanappa, Benjamin Mitzkus, Matthias Bethge, Felix A Wichmann, and Wieland Brendel. On the surprising similarities between supervised and self-supervised models. *arXiv preprint arXiv:2010.08377*, 2020. 99

[24] Spyros Gidaris, Praveer Singh, and Nikos Komodakis. Unsupervised representation learning by predicting image rotations. *arXiv preprint arXiv:1803.07728*, 2018. 100

[25] Sixue Gong, Vishnu Naresh Boddeti, and Anil K Jain. On the intrinsic dimensionality of image representations. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2019. 19

[26] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT press, 2016. 17

[27] Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial nets. In *Advances in neural information processing systems*, 2014. 15

[28] Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial networks. *Communications of the ACM*, 2020. 1

[29] Ian J Goodfellow, Jonathon Shlens, and Christian Szegedy. Explaining and harnessing adversarial examples. *arXiv preprint arXiv:1412.6572*, 2014. 1

[30] Ian J Goodfellow, Oriol Vinyals, and Andrew M Saxe. Qualitatively characterizing neural network optimization problems. *arXiv preprint arXiv:1412.6544*, 2014. 18, 100

[31] Ariel Gordon and et al. Morphnet: Fast & simple resource-constrained structure learning of deep networks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2018. 14

[32] Arthur Gretton, Olivier Bousquet, Alex Smola, and Bernhard Schölkopf. Measuring statistical dependence with hilbert-schmidt norms. In *International Conference on Algorithmic Learning Theory*. Springer, 2005. 2, 20, 23

[33] Tom George Grigg, Dan Busbridge, Jason Ramapuram, and Russ Webb. Do self-supervised and supervised methods learn similar visual representations? *arXiv preprint arXiv:2110.00528*, 2021. 100

[34] David R Hardoon, Sandor Szedmak, and John Shawe-Taylor. Canonical correlation analysis: An overview with application to learning methods. *Neural Computation*, 2004. 2, 20, 23

[35] Mark Harmon and Diego Klabjan. Activation ensembles for deep neural networks. *arXiv preprint arXiv:1702.07790*, 2017. 14

[36] Fengxiang He, Tongliang Liu, and Dacheng Tao. Control batch size and learning rate to generalize well: Theoretical and empirical evidence. In H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett, editors, *Advances in Neural Information Processing Systems (NIPS)*. 2019. 18

[37] Kaiming He, Haoqi Fan, Yuxin Wu, Saining Xie, and Ross Girshick. Momentum contrast for unsupervised visual representation learning. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 9729–9738, 2020. 100

[38] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. In *Proceedings of the IEEE International Conference on Computer Vision (ICCV)*, 2015. 14

[39] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016. 11, 12, 33, 88, 89

[40] Geoffrey Hinton, Oriol Vinyals, and Jeff Dean. Distilling the knowledge in a neural network. *arXiv preprint arXiv:1503.02531*, 2015. 15

[41] Sepp Hochreiter and Jürgen Schmidhuber. Flat minima. *Neural Computation*, 1997. 17

[42] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural Computation*, 1997. 10, 13

[43] Jie Hu, Li Shen, and Gang Sun. Squeeze-and-excitation network. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2018. 14, 87, 88

[44] Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. *Proceedings of The 32nd International Conference on Machine Learning (ICML)*, 2015. 11

[45] Angela H Jiang, Daniel L-K Wong, Giulio Zhou, David G Andersen, Jeffrey Dean, Gregory R Ganger, Gauri Joshi, Michael Kaminksy, Michael Kozuch, Zachary C Lipton, et al. Accelerating deep learning by focusing on the biggest losers. *arXiv preprint arXiv:1910.00762*, 2019. 16

[46] Xiaojie Jin, Chunyan Xu, Jiashi Feng, Yunchao Wei, Junjun Xiong, and Shuicheng Yan. Deep learning with s-shaped rectified linear activation units. In *13th AAAI Conference on Artificial Intelligence*, 2016. 14

[47] Nitish Shirish Keskar, Dheevatsa Mudigere, Jorge Nocedal, Mikhail Smelyanskiy, and Ping Tak Peter Tang. On large-batch training for deep learning: Generalization gap and sharp minima. *arXiv preprint arXiv:1609.04836*, 2016. 17

[48] Prannay Khosla, Piotr Teterwak, Chen Wang, Aaron Sarna, Yonglong Tian, Phillip Isola, Aaron Maschinot, Ce Liu, and Dilip Krishnan. Supervised contrastive learning. *Advances in neural information processing systems*, 33:18661–18673, 2020. 100

[49] Simon Kornblith, Mohammad Norouzi, Honglak Lee, and Geoffrey Hinton. Similarity of neural network representations revisited. *arXiv preprint arXiv:1905.00414*, 2019. 1, 2, 21, 23, 25, 28

[50] Yann LeCun. The mnist database of handwritten digits. *http://yann. lecun. com/exdb/mnist/*, 1998. 1

[51] Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. Deep learning. *Nature*, 2015. 10, 37

[52] Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 1998. 10, 12

[53] Hao Li, Zheng Xu, Gavin Taylor, Christoph Studer, and Tom Goldstein. Visualizing the loss landscape of neural nets. In *Advances in Neural Information Processing Systems (NIPS)*, 2018. 18, 100

[54] Zhong Qiu Lin, Mohammad Javad Shafiee, Stanislav Bochkarev, Michael St Jules, Xiao Yu Wang, and Alexander Wong. Do explanations reflect decisions? a machine-centric strategy to quantify the performance of explainability algorithms. *arXiv preprint arXiv:1910.07387*, 2019. 1

[55] Zhong Qiu Lin and Alexander Wong. Progressive label distillation: Learning input-efficient deep neural networks. *arXiv preprint arXiv:1901.09135*, 2019. 15

[56] Octavio Loyola-Gonzalez. Black-box vs. white-box: Understanding their advantages and weaknesses from a practical point of view. *IEEE access*, 2019. 1

[57] Scott M Lundberg and Su-In Lee. A unified approach to interpreting model predictions. *Advances in neural information processing systems*, 30, 2017. 1

[58] Leland McInnes, John Healy, and James Melville. Umap: Uniform manifold approximation and projection for dimension reduction. *arXiv preprint arXiv:1802.03426*, 2018. 1, 67, 101

[59] Preetum Nakkiran, Gal Kaplun, Yamini Bansal, Tristan Yang, Boaz Barak, and Ilya Sutskever. Deep double descent: Where bigger models and more data hurt. *arXiv preprint arXiv:1912.02292*, 2019. 17

[60] Nicolas Papernot and Patrick McDaniel. Deep k-nearest neighbors: Towards confident, interpretable and robust deep learning. *arXiv preprint arXiv:1803.04765*, 2018. 1, 2, 19

[61] Jongchan Park, Sanghyun Woo, Joon-Young Lee, and In So Kweon. Bam: Bottleneck attention module. *arXiv preprint arXiv:1807.06514*, 2018. 14, 87, 88

[62] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Kopf, Edward Yang, Zachary DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. Pytorch: An imperative style, high-performance deep learning library. In *Advances in Neural Information Processing Systems 32*, pages 8024–8035. Curran Associates, Inc., 2019. 34

[63] Karl W Pettis, Thomas A Bailey, Anil K Jain, and Richard C Dubes. An intrinsic dimensionality estimator from near-neighbor information. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 1979. 19

[64] Milos Radovanovic, Alexandros Nanopoulos, and Mirjana Ivanovic. Hubs in space: Popular nearest neighbors in high-dimensional data. *Journal of Machine Learning Research*, 2010. 65

[65] Maithra Raghu, Justin Gilmer, Jason Yosinski, and Jascha Sohl-Dickstein. Svcca: Singular vector canonical correlation analysis for deep learning dynamics and interpretability. In *Advances in Neural Information Processing Systems (NIPS)*, 2017. 1, 20, 23

[66] Prajit Ramachandran, Barret Zoph, and Quoc V. Le. Searching for activation functions. In *6th International Conference on Learning Representations (ICLR), Workshop Track Proceedings*, 2018. 14

[67] Joseph Redmon, Santosh Divvala, Ross Girshick, and Ali Farhadi. You only look once: Unified, real-time object detection. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 779–788, 2016. 1

[68] Marco Tulio Ribeiro, Sameer Singh, and Carlos Guestrin. "why should i trust you?" explaining the predictions of any classifier. In *Proceedings of the 22nd ACM SIGKDD international conference on knowledge discovery and data mining*, pages 1135–1144, 2016. 1

[69] Simone Scardapane, Michele Scarpiniti, Danilo Comminiello, and Aurelio Uncini. Learning activation functions from data using cubic spline interpolation. In *Italian Workshop on Neural Nets*. Springer, 2017. 14

[70] Mohammad Javad Shafiee, Akshaya Mishra, and Alexander Wong. Deep learning with darwin: Evolutionary synthesis of deep neural networks. *Neural Processing Letters*, 2018. 14

[71] Shashank Shekhar, Florian Bordes, Pascal Vincent, and Ari Morcos. Objectives matter: Understanding the impact of self-supervised objectives on vision transformer representations. *arXiv preprint arXiv:2304.13089*, 2023. 100

[72] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: A simple way to prevent neural networks from overfitting. *The Journal of Machine Learning Research*, 2014. 12

[73] Kenneth O. Stanley and Risto Miikkulainen. Evolving neural networks through augmenting topologies. *Evolutionary Computation*, 2002. 13

[74] Christian Szegedy, Vincent Vanhoucke, Sergey Ioffe, Jonathon Shlens, and Zbigniew Wojna. Rethinking the inception architecture for computer vision. In *Proceedings of IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016. 12

[75] Mingxing Tan and Quoc Le. Efficientnet: Rethinking model scaling for convolutional neural networks. In *International conference on machine learning*, pages 6105–6114. PMLR, 2019. 1

[76] Laurens Van der Maaten and Geoffrey Hinton. Visualizing data using t-sne. *Journal of machine learning research*, 9(11), 2008. 1, 67, 101

[77] Vladimir N Vapnik and A Ya Chervonenkis. On the uniform convergence of relative frequencies of events to their probabilities. In *Measures of complexity: festschrift for alexey chervonenkis*, pages 11–30. Springer, 2015. 26

[78] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. *Advances in neural information processing systems*, 30, 2017. 1, 99

[79] Kirill Vishniakov, Zhiqiang Shen, and Zhuang Liu. Convnet vs transformer, supervised vs clip: Beyond imagenet accuracy. *arXiv preprint arXiv:2311.09215*, 2023. 100

[80] Wei Wen and et al. Learning structured sparsity in deep neural networks. 2016. 14

[81] Alexander Wong. Netscore: Towards universal metrics for large-scale performance analysis of deep neural networks for practical on-device edge usage. In *International Conference on Image Analysis and Recognition (ICIAR)*. Springer, 2019. 17

[82] Alexander Wong, Mahmoud Famouri, Maya Pavlova, and Siddharth Surana. Tinyspeech: Attention condensers for deep speech recognition neural networks on edge devices. *arXiv preprint arXiv:2008.04245*, 2020. 14, 85

[83] Alexander Wong, Mahmoud Famuori, Mohammad Javad Shafiee, Francis Li, Brendan Chwyl, and Jonathan Chung. Yolo nano: A highly compact you only look once convolutional neural network for object detection. *arXiv preprint arXiv:1910.01271*, 2019. 14

[84] Alexander Wong, Mohammad Javad Shafiee, Brendan Chwyl, and Francis Li. Ferminets: Learning generative machines to generate efficient neural networks via generative synthesis. *arXiv preprint arXiv:1809.05989*, 2018. 14

[85] Sergey Zagoruyko and Nikos Komodakis. Wide residual networks. *arXiv preprint arXiv:1605.07146*, 2016. 1

[86] Jure Zbontar, Li Jing, Ishan Misra, Yann LeCun, and Stéphane Deny. Barlow twins: Self-supervised learning via redundancy reduction. In *International conference on machine learning*, pages 12310–12320. PMLR, 2021. 100

[87] Barret Zoph and Quoc V. Le. Neural architecture search with reinforcement learning. In *5th International Conference on Learning Representations (ICLR), Conference Track Proceedings*, 2017. 13