Interpreting deep neural networks with sample sets

AEW Venter

orcid.org/0000-0001-7014-8711

Dissertation accepted in fulfilment of the requirements for the degree *Master of Engineering in Computer and Electronic Engineering* at the North-West University

Supervisor:	Prof MH Davel
Co-supervisor:	Dr MW Theunissen

Graduation:	July 2022
Student number:	27383962

Declaration

I, Arthur Edgar William Venter, hereby declare that the dissertation entitled "Interpreting deep neural networks with sample sets" is my own original work and has not already been submitted to any other university or institution for examination.

Arthur Edgar William Venter

Student number: 27383962

Signed on the 17th day of February 2022 at Potchefstroom.

Acknowledgements

This study was conducted under the Multilingual Speech Technologies (MuST) research group of the North-West University, which is also a member of the Centre for Artificial Intelligence Research (CAIR) of the Department of Science and Innovation. It was supervised by Professor Marelie Davel and Doctor Tian Theunissen.

The effort required to produce a master's thesis is by no means a small feat and mine was no exception. Although I submit this thesis under my own name, none of this would have been possible if were not for the contributions and assistance of the following people:

Ulrike Janke, whose work behind the scenes did not go unnoticed. I sincerely thank you for going through all the trouble to ensure a healthy work environment. Your friendly demeanour made me always feel welcome within the group.

Prof. Marelie and **Dr. Tian**, whose guidance and understanding nature played an instrumental part during the course of this thesis. Both of you taught me more than just how to write a good dissertation. Thank you for always going the extra mile and supporting me in my quest for knowledge.

All my **friends** and **fellow colleagues**, whose camaraderie has been greatly appreciated. Thank you all for your helpful conversations and deep insights. I am privileged to have such good friends and to be a part of such a skilled group of researchers.

My partner, **Chara Grant**, whom was there when I needed her the most. Your charismatic personality never ceased to bring a smile to my face. Thank you, for your timely distractions, uplifting spirit and always bringing me the necessary caffeine when it was needed.

My **family**, in particular my parents **Arthurita Venter** and **Willie Venter**. Without you, I would not be the man I am today. Thank you from the bottom of my heart for the years of encouragement, guidance and support throughout my life. I love you both very much.

My **Heavenly Father**, without whom, I am sure I would not have been provided the opportunity to pursue this master's degree in the first place. Thank you for all the strength and patience You have provided me with over the years. I feel truly blessed by the people who surround me and the opportunities I have been given.

Galatians 2:19-20 (ESV): For through the law I died to the law, so that I might live to God. I have been crucified with Christ. It is no longer I who live, but Christ who lives in me. And the life I now live in the flesh I live by faith in the Son of God, who loved me and gave himself for me.

Soli Deo Gloria

Abstract

Despite their impressive performances on a range of widespread tasks, deep neural networks (DNNs) are generally considered 'black box' models due to the lack of transparency behind their decision-making processes. Researchers address this issue through the use of interpretability techniques which, in the context of this study, uses some set of rules to map the output of the network back onto its inputs.

In recent works, *sample set analysis* has been proposed as a novel methodology to better study the generalisation capabilities of DNNs through analysing the natural sample clusters formed by the network itself. By being able to directly identify the nodes that process the largest number of class samples, this methodology does offer some potential as a possible means for improving DNN interpretations.

In this exploratory study, we investigate the applicability of sample set analysis as a tool for DNN interpretability purposes. We do this by analysing the inner workings of networks trained on the MNIST data set through using sample set analysis in conjunction with the Layer-wise Relevance Propagation (LRP) interpretability technique, while verifying the results using a custom generated synthetic data set.

Our analysis led to the introduction of *encoding sample sets*, an additional sample set category that groups class samples according to their binary node activation patterns in a given layer. Through encoding sample sets, we further introduce the concepts of *core* and *variation* nodes, which refer to the nodes that activates for all encoding sample sets within a layer or only a subset of them, respectively.

When used in conjunction with LRP, encoding sample sets are capable of generating interpretations which represent groups of samples rather then representing them individually. We coined this approach *set interpretations* and found that it provides interpretations highly similar to its individual counterparts while simplifying the interpretation process.

Keywords: *deep neural networks, interpretability, sample sets, core nodes, variation nodes, encoding sample sets, node sample sets, Layer-wise Relevance Propagation*

Contents

List of Figures

List of Tables

List of Abbreviations

ANN Artificial Neural Network

DNN Deep Neural Network

CNN Convolutional Neural Network

RNN Recurrent Neural Network

MLP Multilayer Perceptron

MNIST Modified National Institute of Standards and Technology

SGD Stochastic Gradient Descent

ReLU Rectified Linear Unit

MSE Mean Squared Error

DTD Deep Taylor Decomposition

IG Integrated Gradients

CAM Class Activated Mapping

Grad-CAM Gradient-weighted Class Activation Mapping

SHAP SHapley Additive exPlanations

DeepLIFT Deep Learning Important FeaTures

LIME Local Interpretable Model-agnostic Explanation

LRP Layer-wise Relevance Propagation

PASCAL Pattern Analysis, Statistical Modelling and Computational Learning

VOC Visual Object Classes

GAMI-Net Generalised Additive Models with structured Interactions

IDE Integrated Development Environment

CIFAR Canadian Institute For Advanced Research

Chapter 1

Introduction

Overview of the study and its importance. We provide the scope and objectives of this study and describe the methodology followed.

1.1 Background

For many machine learning tasks, it is often necessary to build complex models in order to achieve good performance. However, it is common for such complex models to be notoriously difficult to understand in terms of how they arrive at particular predictions when presented with previously unseen data samples. Therefore we use *interpretability* methods to extract rationales for classification decisions from the model. Interpretability is a crucial aspect of machine learning, especially in domains such as self-driving vehicles [1] and healthcare [2], [3], where understanding the reasoning behind the decisions that models make is paramount.

Deep Neural Networks (DNNs) are powerful machine learning methods that are capable of producing impressive results in numerous complex tasks such as image classification

[4], [5], natural language processing [6], or speech recognition, due to their ability to handle large amounts of data [7]. Despite there being a current general understanding of DNN behaviour, there is no direct way to observe the reasoning behind their internal operations as data is processed. For this reason, DNNs are generally regarded as 'black box' methods; in other words, it is difficult to innately comprehend their decision-making processes. From the different approaches proposed to interpret various types of DNNs, such as visual explanation and explanations by example [8], we focus primarily on feature attribution methods [9]–[11] for reasons explained later in Section 2.3.

The specific definition of interpretability varies depending on the context in which it is used. For our study, we use the definition provided by *Montavon et al.*, who state that interpretability refers to how well you can determine the rule by which a model maps the input features to output values, as well as how subsequent changes to the inputs will impact these output values [10]. This definition of interpretability allows us to assign a 'relevance' score to individual input features, quantifying their influence on the final classification [4].

In recent related work [12] suggested that simple Multilayer Perceptron (MLP) networks are able to delegate their decision-making processes to specific subcomponents, such as particular nodes in the network. The sets of samples that interact with these subcomponents were simultaneously introduced by Davel et al. in [13]–[15] as *node-specific sample sets* or just *sample sets*. It has also been shown that estimators trained on these different subcomponents can be used to correctly classify previously unseen data [12]. In other words, these subcomponents can be utilised to correctly classify data set samples. Sample set analysis, however, has mostly been studied as a tool to probe the generalisation ability of deep neural networks. Despite this, the information provided by sample set analysis suggests that it could also be used as a means to study the interpretability of DNNs.

Our goal of this study is to determine whether sample set analysis can contribute to interpretability in any useful way. We study this by investigating the inner workings of trained DNNs using sample sets alongside a selected interpretability method. Sample sets are still a relatively new concept, making this mainly an exploratory study. We hope that

our results will lay the foundation for future studies that will benefit researchers when interpreting DNNs.

1.2 Problem statement

DNNs have achieved outstanding predictive performance in a variety of domains [16], but are notoriously difficult to interpret due to their internal operations being too complex to be inherently interpretable. Despite the numerous interpretability techniques available today, the research field is still very active [11]. Furthermore, the recently proposed concept of *sample sets* analysis exists, which is normally used to probe the generalisation capabilities of DNNs [12].

Currently, feature attribution techniques typically produce an individual interpretation per sample. Our goal during the span of this study is to discover whether sample set analysis is capable of enhancing the interpretations generated by existing interpretability methods, by determining whether interpretations can be grouped using information from sample set analysis. Such a process whereby sample set analysis can be used alongside other interpretability methods to interpret the predictive behaviour in neural networks has not yet been defined. Specifically, we do not yet know whether using sample sets to analyse the subcomponents of a network is useful when interpreting DNNs.

1.3 Project scope

Taking the problem statement above into consideration, we restrict the initial research scope to a limited set of architectures and data sets:

- **Data sets:** Due to the exploratory nature of this study, our networks will be trained on the publicly available Modified National Institute of Standards and Technology (MNIST) image classification data set [17]. The simplicity of the MNIST data set

allows us to better investigate the inner workings of our networks without needing to consider the impact of some external factors that could occur due to data set complexity. We also generate several synthetic data sets, which are used alongside MNIST throughout the study to demonstrate how certain interpretability methods can be applied and evaluated.

- **Architectures:** The main focus will be on a classification problem with a standard, fully-connected, feedforward, Rectified Linear Unit (ReLU)-activated [18] neural network that has a large number of trainable parameters, which will allow us to study the network in detail. We acknowledge that it is more prominent within literature to use interpretability techniques in conjunction with convolutional architectures. However, due to the exploratory nature of this study, and the novelty of sample set analysis, we limit our focus to only the above-mentioned architectures.

Our goal with this study does not entail creating a state-of-the-art interpretability technique, but instead explores the possibility of enhancing modern post hoc interpretability methods (see Section 2.3.3). This is done by exploring whether sample set analysis can be used as a tool in conjunction with specific interpretability methods to improve their interpretations. The scope of this project does not include a full diagnostic of the applicability of sample set analysis with regard to interpreting DNNs.

1.4 Research questions

Taking the project scope into consideration, we prepare the following questions:

- Which currently used interpretability techniques are best suited to determine the applicability of sample set analysis as an interpretability technique enhancer?
- How can one refine sample set analysis to be applicable as a tool alongside interpretability methods?

- To what degree will this refined sample set methodology enhance the capabilities of modern interpretability techniques?
- Will interpretability methods continue to similarly attribute importance to the same inputs as they did before the application of sample set analysis?

1.5 Objectives

In order to answer the research questions above, this study will have the following objectives:

- Select an appropriate interpretability technique. Obtain or implement an interpreter that is capable of generating interpretations by attributing relevance to the input features and integrate it with the in-house analysis codebase.
- Establish baseline MLP models for sample set analysis using MNIST and synthetic data to utilise as analysis environment.
- Investigate the sample sets of the chosen networks. Explore the relationship between sample sets across the entire network and develop an intuition regarding their potential use to supplement existing interpretability techniques.
- Explore the applicability of sample sets as a tool to enhance the capabilities of a provided interpretability tool. Formalise the methodology for analysing the baseline networks using sample set analysis.
- Determine whether sample set analysis makes any noteworthy contributions when used alongside the chosen interpretability method to generate interpretations.

1.6 Research methodology

This study is an investigation into the applicability of sample sets as a usable tool for deep neural network interpretability. The following will form part of the study:

- **Literature review:** Gain a fundamental understanding of the background:
  - The basic principles of deep neural networks.
  - The methodology behind sample set analysis.
  - The implementation and evaluation of relevant interpretation techniques.
- **Experimental framework selection:** Select the most appropriate interpretability technique as a basis for initial experimentation. Identify prominent indicators for evaluating the interpretability techniques.
- **Development environment:** The following codebases will be required for our experimental analysis (discussed in further detail in Section 3.2):
  - Mustnet, our research group's PyTorch-based, in-house codebase for DNN training, optimisation and analysis.
  - A publicly available in-house interpretability codebase will be added as an extension to Mustnet, for interpreting and analysing MLPs. If we are unable to obtain an existing codebase, the selected interpretability method will be implemented from first principles and verified against existing results.
- **Codebase development:** Extend our research group's in-house codebase for analysing neural networks with the necessary components required for this investigation.
- **Experimental development:** We will start with the simplest architecture and data set alongside the most appropriate interpretability technique, adding complexity incrementally as the project progresses:
  - Determine a baseline model for the various tasks.

  - Investigate the inner workings of the model using sample sets and a suitable interpretability technique.
  - Compare and evaluate the validity and meaningfulness of the results of each investigation.

  During experimental development, we will use a synthetic data set where a ground truth is required (to develop an intuition or test new methods) and the MNIST data set for a practical application.

- **Sample set analysis framework:** Formalise the current methodology with regard to analysing sample sets for the purposes of interpreting DNNs.
- **Assess and discuss the experimental findings:** The feasibility of using sample set analysis as a means to enhance modern interpretability methods will be analysed and discussed. The meaningfulness of the final results will also be assessed.

1.7 Dissertation overview

This dissertation has the following structure:

- Chapter 2 contains an overview of deep neural networks followed by additional background on deep neural network interpretability and sample set analysis.
- Chapter 3 describes the models trained and optimised for this study, as well as the implementation and verification of our in-house Layer-wise Relevance Propagation (LRP) interpretability technique, which is selected in Section 2.5, to analyse the efficacy of sample sets as an interpretability tool.
- Chapter 4 explores node sample sets and their ability to group samples into meaningful clusters when interpreting network predictions. Here, node sample sets refers to the original definition of sample sets.

- Chapter 5 introduces a new concept dubbed *encoding sample sets*, given the limitations of node sample sets observed and described in Section 4.4. Unlike node sample sets, encoding sample sets are capable of generating single interpretations from groups of samples, which is demonstrated in this chapter.
- Chapter 6 concludes the dissertation with a summary of the key findings of the investigation.

Chapter 2

Literature review

Background study of deep neural networks, network interpretability, layer-wise relevance propagation and sample set analysis.

2.1 Introduction

In this chapter, we discuss the topics relevant to this study as presented in literature. We provide an overview of the structure and optimisation procedures of DNNs in Section 2.2. Section 2.3 discusses the role of DNN interpretability, along with related techniques. A background to sample set analysis and its methodology is provided in Section 2.4. Reasons as to why we chose LRP as our primary interpretability technique of interest are provided in Section 2.5. We explain how LRP functions in Section 2.6.

2.2 Deep neural networks

Section 2.2.1 serves as a short introduction to DNNs, while Section 2.2.2 describes some of the training and optimisation methods of DNNs.

2.2.1 An introduction to DNNs

DNNs are powerful machine learning methods that are capable of producing impressive results on numerous complex tasks such as image classification, natural language processing [19], speech recognition [20], [21] or human action recognition [22], [23]. This is due to their ability to harvest and process large amounts of data. Deep learning models can be optimised effectively, are scalable to high-dimensional data, and produce excellent performance on previously unseen data [16]. For these reasons, DNNs have become an important instrument in solving large-scale, real-world problems.

At the time of writing, the term DNN has come to refer to a wide variety of techniques within the Artificial Neural Network (ANN) domain. These techniques are characterised by using layers of nodes connected via trainable weights to generate an output function $y = f(\mathbf{x}, \boldsymbol{\theta})$, which best approximates the task's true function f'. Here, $\mathbf{x}$ and $\boldsymbol{\theta}$ refer to the network's inputs and trainable parameters, respectively. Throughout this dissertation, we will also be using the following terms with these descriptions:

- ***width:*** The number of nodes per hidden layer.
- ***depth:*** The number of hidden layers.
- ***shallow:*** The hidden layers closest to the model's input features.
- ***deep:*** The hidden layers closest to the output values.

When given only the width and depth of a model, the reader can assume that all hidden layers will have the same number of nodes.

In the forthcoming chapters, we will be focusing primarily on MLPs. These are the simplest forms of ANNs and are used in both regression and classification problems. MLPs approximate functions by defining every node a_k in the succeeding layer $(l+1)$ as the weighted sum, $\sum_j w_{jk}^{(l,l+1)}$, of all nodes a_j in the current layer (l), which is then passed through an activation function σ, as shown in Eq. 2.1 [16].

$$a_k^{(l+1)} = \sigma(\sum_j a_j^{(l)} w_{jk}^{(l,l+1)} + b^{(l+1)}) \tag{2.1}$$

Here $w_{jk}^{(l,l+1)}$ denotes the individual weight from node j from layer (l) to node k in layer $(l+1)$. The term $a_k^{(l+1)}$ refers to the activation value of node k in layer $(l+1)$ and similarly $a_j^{(l)}$ refers to the activation value of node j in layer (l). The multiplication of the appropriate weights and activation values/input features ($\mathbf{a}^{(l)}/\mathbf{x}$), at a single layer, results in what is essentially a linear regression model ($\mathbf{a}^{(l)}\mathbf{w}_{(l,l+1)}$).

The element-wise non-linear activation function (σ) is then used to perform non-linear transformations on the inputs to make the model capable of learning more complex tasks. From the variety of available activation functions (tanh, sigmoid [18], leaky ReLU [24], etc), we chose standard ReLU, as defined in Eq. 2.2, due to its popularity and performance in multiple applications [24], [25].

$$\sigma(\mathbf{x}) = max(0, \mathbf{x}) \tag{2.2}$$

The bias term provides the model with more flexibility and assumes the same role as a constant in a linear function. A bias term can be placed in any of the hidden layers of a DNN [26]. Adding a bias on the first layer only (rather than all layers) produces a network that is easier to analyse analytically and equates to an additional weight connected to an input feature with a constant value of one. As long as there is a bias in the first layer, an MLP is able to automatically propagate this bias to deeper layers, if needed - the weight linking the bias node to a node in a subsequent layer is emphasised, while all other input features are suppressed. This is not only theoretically possible, but observed in practice

[14].

As an example of how these mathematical functions work in practice, consider the simple MLP shown in Figure 2.1. It has an output of $\mathbf{y}^{(l+1)} = \sigma(\mathbf{a}^{(l)}\mathbf{w}^{(l,l+1)})$ where σ is a non-linear activation function, and $\mathbf{a}^{(l)} = \sigma(\mathbf{a}^{(l-1)}\mathbf{w}^{(l-1,l)})$ and $\mathbf{a}^{(l-1)} = \sigma(\mathbf{x}^{(l-2)}\mathbf{w}^{(l-2,l-1)} + b^{(l-1)})$.

It is important to understand that MLPs, as discussed here, are one of the simplest forms of DNNs. Other, more complex, architectures also exist. One such example is a Convolutional Neural Network (CNN) [27], [28] which specialises in processing data such as images in a grid-like pattern. CNNs utilise convolutional and pooling layers to reduce the amount of redundant data through a feature extraction process. Typically, the network then performs classification using these extracted features via a feedforward network.

Recurrent Neural Networks (RNNs) [29], [30] are another complex architecture, which

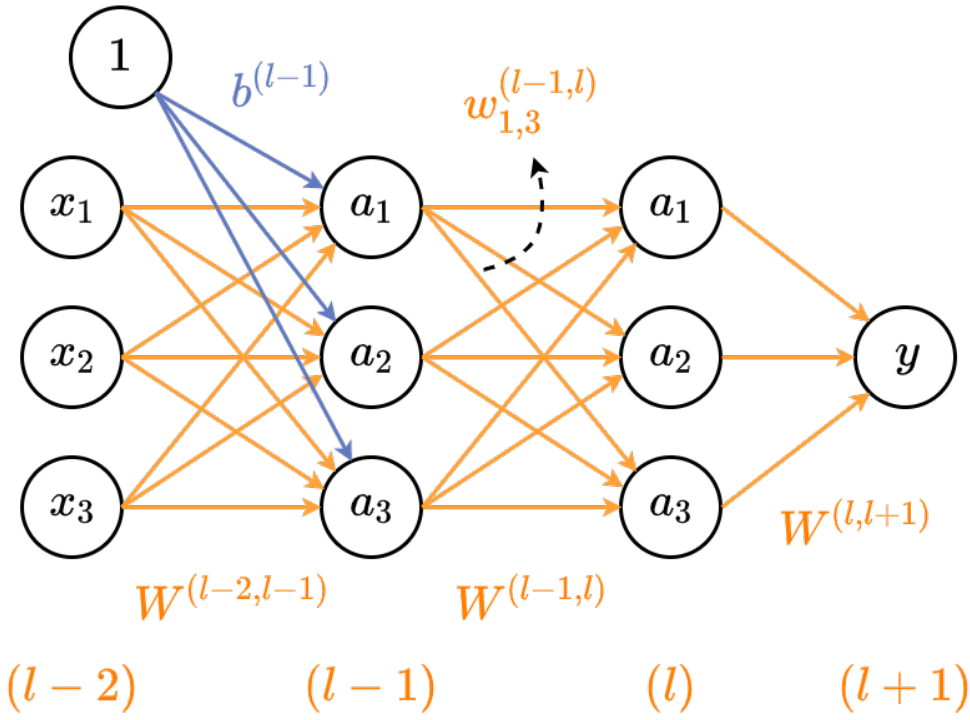


Figure 2.1: A simple MLP with a depth of two layers, width of three nodes, three inputs, a bias and one output. The different layers are shown at the bottom.

specialise in processing sequential data such as time-series data. Whereas standard MLP networks are only capable of using the inputs from the previous layer to generate outputs for the current layer, RNNs possess an internal memory, which theoretically allows them to 'remember' all previous inputs when generating outputs. Over the years, many variations have been proposed to the standard RNN architecture, such as time delay neural networks [31], and the influential long short-term memory networks [32].

While acknowledging the existence of these more complex DNNs, we still limit ourselves to the traditional MLP architecture. Its simplicity remains ideal when performing an exploratory study such as ours, as it prevents the added complexities of other DNN structures from complicating the analysis process. That is not to say that elements of our study will not be applicable to other neural networks, as we aim to study elements that are also present in multiple DNN archetypes.

2.2.2 Training and optimisation

Optimisation of an ANN, and therefore an MLP, occurs by tuning the network's set of parameters ($\boldsymbol{\theta}$) so as to minimise the loss of the network [16]. The loss is the measurable error between the prediction made by the network and the true target value as measured on a train set. Two popular loss functions used today are the Mean Squared Error (MSE) [33] and Cross-Entropy [16] loss functions. We chose Cross-Entropy (otherwise known as Log Loss) for our models, as it is commonly used in classification tasks, and is suitable for classification in the presence of more than two target classes. MSE, on the other hand, is best suited to regression-type problems due to its emphasis on numerical output values.

The parameters of a network are trained using *gradient descent*, which updates each parameter θ within the set of parameters $\boldsymbol{\theta}$ proportionally to the negative of the gradient of the loss with respect to θ:

$$\Delta\theta = -\eta\frac{\partial L}{\partial\theta} \tag{2.3}$$

where L represents the loss, η the learning rate and $\Delta\theta$ the change in that parameter's value. Thus, the updated parameter, θ' is equal to the current parameter value θ plus its change in value $\Delta\theta$ as shown in Eq. 2.4.

$$\theta' = \theta + \Delta\theta \tag{2.4}$$

Updating every parameter for all training samples can be highly computationally inefficient. To combat this issue, we use the Stochastic Gradient Descent (SGD) [16] optimisation technique, which updates the parameters using an approximated average gradient from a batch [1] of randomly chosen samples that were passed through the network [26]. The computational efficiency gained from using SGD allows networks to use larger data sets. Another optimisation technique that functions similarly to SGD is Adam, which is commonly used for its adaptive estimates of lower-order moments [34].

Just as we optimise our trainable parameters, so must we optimise the model's hyperparameters which are used by the model when calculating the training parameters. The term 'hyperparameters' refers to the set of parameters chosen beforehand that affects the training process, but which are not updated during the training process [16], [26].

A basic, but popular way to optimise hyperparameters is by performing a parameter sweep (also known as a grid search). This involves an exhaustive search through a manually specified subset of hyperparameter values, where some network measurements, for instance validation error rate, may be used to determine the optimal set of hyperparameter values. Alternative optimisation methods also exist, such as stochastic searches [35], which randomly search for hyperparameter values, while using what they learn to narrow down their future searches [36]. Bayesian hyperparameter optimisation [37] can also be used, which chooses potential hyperparameter values based on previous observations.

Before training, weights should be initialised to encourage convergence, good performance, and stability [16]. Additionally, through proper weight initialisation, problems such as

[1] Some authors use 'SGD' when processing samples individually and 'minibatch SGD' when processing samples in batches [26]. We use 'SGD' to refer to processing samples in batches.

exploding or vanishing gradients can be reduced or avoided [38]. Weights can be initialised through methods such as sampling all weights from a uniform distribution, but typically, more sophisticated initialisation schemes, such the Xavier [39] and Kaiming [40] schemes, are preferred. For this study we chose the Kaiming initialisation scheme as it is a good option for ReLU-activated networks [40]. The weights are randomly initialised, but differ in range depending on the width of the previous layer.

We will be using the Kaiming-based initialisation scheme as implemented by Pytorch [41] (described in Section 3.2, which draws samples from a uniform distribution $\mathcal{U}(-\phi, \phi)$ where:

$$\phi = g \times \sqrt{\frac{3}{fan\_in}} \tag{2.5}$$

Here g refers to some optional scaling factor and *fan_in* to the number of inputs from the previous layer to the current layer [40]. Bias weights, on the other hand, are initialised from a uniform distribution $\mathcal{U}(-\sigma, \sigma)$ where:

$$\sigma = \frac{1}{\sqrt{fan\_in}} \tag{2.6}$$

Regularising techniques such as Batch Normalisation [42], dropout [43], [44] and weight decay [26], are useful tools when aiming to reduce generalisation error. In this context, generalisation is defined as the ability of a DNN to perform well on data that is different from the training data [45], where the generalisation gap is the measured difference between the DNN training and evaluation error rates [16]. The larger the gap, the worse the network's generalisation. The implementation of regularisation techniques in some sense simplifies the model by constraining it to a simpler set of possible solutions [16], [26]. However, we aim to investigate the inner workings of our models in later chapters, which can be altered or obscured by the use of these regularisation techniques. Thus, for the purposes of this study, we will not be using any technique that might have a regularising effect.

2.3 Interpreting deep neural networks

In Section 2.3.1 we briefly discuss model interpretability and its importance. Section 2.3.2 discusses the taxonomy we will be using to classify the interpretability techniques, while Section 2.3.3 provides an overview of some of the prominent approaches taken regarding interpretability. Lastly, we discuss validating the efficacy of interpretability techniques in Section 2.3.4.

2.3.1 Importance of interpretability techniques

Despite their efficiency in solving complex tasks, DNNs are not without their drawbacks. Due to their multilayer structure and non-linear nature, we still struggle to obtain insight into their internal operation and behaviour [10], [46], [47]. This causes DNNs to be generally regarded as 'black box' methods; meaning that it is difficult to innately comprehend how neural networks arrive at certain outputs when given previously unseen data samples [9]. Note that although we mainly focus on classification type problems within this study, interpretability methods are not limited to these types of tasks.

The uncertainty associated with the inner workings of DNNs may impede the development of better models and hinder human experts in verifying classification decisions. This causes DNNs to be considered less trustworthy, especially when used in domains where explanation and reliability are crucial, such as autonomous driving [1] and medical applications [2], [3].

The following anecdote from Samek et al. [48] motivates why the concept of interpretability is important. In the early 1900s there lived a horse nicknamed *Clever Hans* that could supposedly count. This scientific sensation was later on investigated by psychologist Oskar Pfungst, who discovered that Hans did not derive the correct answer through arithmetic, but rather through subtle cues from the questioner's reactions as he approached the correct answer. This phenomenon manifests itself in modern DNNs as so-called *Clever Hans predictors*, where neural networks show bias towards superfluous components in the data

set it is being trained on [48].

To provide another example, the Pattern Analysis, Statistical Modelling and Computational Learning (PASCAL) Visual Object Classes (VOC) competition was a series of annual classification challenges from $2005-2012$ [49]. During each challenge, the methods built by participants were evaluated on a provided annotated data set. It was later shown by Lapuschkin et al. [50], [51] that one of the methods used to win the PASCAL VOC competition as described in Everingham et al. [49] did not perform as initially perceived. Rather than finding the elements in the data which best described the object of interest, it made its classifications based on correlations within the data sets. The method identified boats and trains via the presence of water and rails, respectively. Horses were recognised not through the features of the horse itself, but rather by the presence of a copyright watermark on the images that were presented to the model.

In summary: deep neural networks often lack a clear *interpretability* of the classifier's predictions. In this context, interpretability refers to how well one can determine the rule by which a model maps input features to output values and how subsequent changes to the inputs will impact the output [10]. As a result, a selected interpretability technique will describe how well a machine learning model captures the essence of the data it is trained on.

Thus, when given an appropriately trained model, interpretability techniques should verify the model's integrity by providing easily explainable rationales for the classifier's decisions [9]. These explanations should be presented by showcasing how the output of the model maps to the provided input features. This will not only help practitioners in other fields to understand the strengths and weaknesses of a model, but also how to compensate for it [52]. In turn, this will also help professionals to expose biases within data sets or to gain new insights [50], [51].

2.3.2 DNNs interpretation taxonomy

Differentiating between interpretability techniques can be very tricky as there does not appear to be a generally agreed-upon taxonomy among authors. To mitigate this issue, we rely on the guidelines provided by Molnar [53] to classify different techniques. This includes categorising techniques based on the results they generate:

- **Feature summary statistics**: Some methods generate feature relevance, feature interaction values, or some other statistic for each feature. These are known as feature summary statistics.
- **Feature summary visualisation**: The statistics given for each input feature by *feature summary statistics* may be visualised. This is sometimes necessary for a summary statistic to be fully useful.
- **Model internals**: The results provided by the inherently interpretable components of intrinsic interpretation techniques (the definition of this is given later in this section). This would include decision thresholds and feature splits in decision trees and linear model weights, among others.
- **Data point**: These are methods that generate and return new data points to help interpret a model. The example given by Molnar is counterfactual explanations: input features are changed until the output changes in a relevant way (for example, the classification would completely flip to another class), but the new data point that is produced is still similar to the original data point. The differences between these two similar data points can be used to identify the features that were relevant to the particular output.
- **Intrinsically interpretable model**: This is done by approximating a target model with an another, inherently interpretable model. The second model is then interpreted through *model internals* or *feature summary statistics*.

We additionally use the following criteria, also described by Molnar [53] and Adadi et al. [11], to classify the different approaches for interpreting DNNs:

- Intrinsic or extrinsic
- Model-specific or model agnostic
- Local or global

We further define these different criteria as follows:

Intrinsic or extrinsic

The two main branches under which interpretability techniques are typically classified are *intrinsic* or *extrinsic* techniques [53]. A relatively simple way to ensure model interpretability is to structure the model so it is *intrinsically interpretable* (which is to inherently interpretable) to the end user. Examples of such models are decision trees [54] and Bayesian Rule Lists [55].

Unfortunately, the complexity required of models to achieve good results on certain tasks leads to the choice of models that are not intrinsically interpretable, such as DNNs. Thus, to be intrinsically interpretable, models need to be simple or simplified by modifying their internal structure. Both of these actions come at the cost of the model's accuracy [11], [56].

Alternatively, highly complex, highly accurate but inscrutable models can be constructed and trained, as well as interpreted, using separately developed techniques that do not form part of the model's internal operations. Such techniques are referred to as *extrinsically interpretable* methods and include techniques such as text explanations, visual explanations, explanations by example, and explanations by simplification [8]. These methods, however, can be more complex and time-consuming than intrinsically interpretable techniques.

Model-specific or model-agnostic

A simple taxonomy for further classifying different interpretability techniques, is by determining whether the technique is *model-specific* or *model-agnostic* [8], [57]. Model-specific

techniques are designed for a specific model type, while *model-agnostic* techniques are designed to be applicable to any or most particular models.

Intrinsically interpretable techniques will always be model-specific as the technique can only be applied to the singular model it was intrinsically designed for. On the other hand, several extrinsically interpretable methods exist which can be classified as either model-specific or model-agnostic.

Local or global

An even further classification of interpretability techniques is between *global* and *local* explanations [8], [58]. Global explanations refer to understanding the model predictions over the entire data set, i.e. the features that are on average most important across all samples. Local explanations, on the other hand, typically refer to understanding predictions based on a single sample.

2.3.3 Approaches to interpreting DNNs

As one would expect, the method by which a DNN is interpreted depends on several factors, including the network's architecture, whether the output is regression- or classification-based, and the nature of the provided input features. Thus, several approaches have been developed in an attempt to interpret the different types of DNNs.

We mentioned previously that DNNs are typically not intrinsically interpretable. However, some examples of intrinsically interpretable DNNs do exist, such as Generalised Additive Models with structured Interactions (GAMI-Net), proposed by Yang et al. [59], which attempts to balance model accuracy and model interpretability. GAMI-Net is a DNN model that is split into multiple sub-networks, where each sub-network is intended to either capture the main effect of a single feature, or the effect of an interaction between two particular features. Regardless of the existence of such models, we focus primarily on extrinsic interpretation techniques.

From the different extrinsic interpretation techniques, we highlight a popular category that is still actively under investigation, namely, *feature attribution methods* [7], [9], [60]–[62]. These methods utilise different techniques to attribute importance scores to the input features of a model. Although the different techniques have overlapping characteristics, they can each be classified into a specific approach according to the most prominent of these characteristics.

Perturbation-based approach

One such approach is *perturbation-based methods*, which typically refers to techniques where individual inputs or nodes are perturbed to observe the effect on the output and activation values of later layers [47], [52], [63]. Although techniques following this approach can show that models may be sensitive to changes in their structure, it has been noted that these methods are computationally inefficient. Models need to be trained once in normal conditions, with each subsequent perturbation requiring a separate forward pass to monitor the changes in the model's outputs [62]. This can become computationally expensive as the number of features to be perturbed increases.

Gradient-based approach

Another approach is *gradient-based methods*, utilised by techniques such as Integrated Gradients (IG) [52], [62], Class Activated Mapping (CAM) [64] and Gradient-weighted Class Activation Mapping (Grad-CAM) [65]. These approaches calculate and use the derivative or partial derivative of a model's output with respect to the input features to attribute importance to each of them.

For instance, IG calculates the average gradient of the model's output with respect to each input feature as they are interpolated from a baseline value x' (typically zero), to their original value x. The resulting gradients inform which inputs have the strongest effect on the model's output. Despite being able to attribute importance scores to the input features of individual examples, IG is not able to explain the interactions between

features nor does it provide global feature importance [52].

CAM is a model-specific technique that highlights the inputs a network primarily uses for its classification decisions. Grad-CAM is used in CNN architectures, and may be viewed as a more flexible and generalised version of CAM. Unlike IG, when using Grad-CAM the gradients of the predicted class's logits, rather than the gradients of the features themselves, are used to produce feature ranking scores [65]. This process, however, can only be utilised within a CNN architecture [62]. Thus, this method is not applicable to our study and will not be considered further.

Additive feature attribution approach

Another approach is *additive feature attribution*, which creates an approximate surrogate linear explanation model f' for the prediction of the true model f, where the contributions of each feature, plus some baseline output of f, add up to the prediction of f for a given input sample [60]. Some of the techniques that follow this approach are: Local Interpretable Model-agnostic Explanation (LIME) [66], Deep Learning Important FeaTures (DeepLIFT) [67] and SHapley Additive exPlanations (SHAP) [60].

Backward propagation approach

Lastly, a set of techniques exists that shares many similarities with the additive feature attribution approach, but has enough dissimilarity that there is some debate among authors as to whether or not these techniques should be included within the same category. Lundberg et al. [60] would argue that these approaches should be unified under the definition of additive feature attribution techniques, since the final attributions add up to the model output value as is typical of the additive attribution approach. However, in the research conducted by Ancona et al. [62], Montavon et al. [10] and Samek et al. [9] we see these techniques are functionally unique enough to be classified separately. For clarity, we refer to the work of Ancona et al. [62] and Montavon et al. [10], which categorise these techniques as *backward propagation techniques*. The authors define this category as

those techniques that use the information gathered from the forward pass of the network to calculate some explanatory values during the backward pass, which may then be used to generate interpretations.

Although the techniques from both approaches share many similarities, the emphasis lies on the mechanisms of the interpretation process. Additive feature attribution focuses on generating a simplified linear version of the trained model for its interpretations, where the relevance of each feature forms the linear weights [60]. The definition of this approach does not focus on the generation of the relevance values themselves in the first place, and as a consequence, many techniques from other approaches can be categorised as an additive attribution approach. The defining characteristic of backward propagation techniques are their use of functions, akin to mathematical rules, that particularly rely on the backpropagation algorithm to iteratively redistribute relevance from the model's output throughout the network, back onto its input features. In this context, *relevance* refers to node or input contributions towards the predicted classification, which can be either positive (in line with the current prediction) or negative (against the current prediction). In other words, backward propagation techniques focus on the redistribution of relevance from the output, throughout the network, to the network's input features in order to arrive at their final feature attributions, rather than through the generation of an accurate surrogate model. However, depending on whether or not the feature relevance values generated can be arranged or represented according to the definition of additive attribution techniques, backward propagation techniques, similarly to any other technique, may fall under the additive attribution technique banner, but they need not necessarily.

It is to be noted that not all backward propagation techniques fit the definition of additive attribution approaches with regard to how their final feature relevance values are acquired, and not all additive attribution techniques make use of the backward propagation approach. It can also be noted that gradient-based approaches may also fall under the definition of backward propagation techniques, since the gradient is utilised, but not all backward propagation techniques make use of a gradient-based approach, so the classification does not go both ways.

An early example of a backward propagation technique is deconvolution, which does not use backpropagation in the strict sense of the word. Simply put, it can be thought of as using a convolutional neural network model in reverse [10]. The technique uses max-pooling layers to correctly attribute importance to the inputs, along with other methods such unpooling, rectification, and filtering to reconstruct the variable states in previous layers [47].

Another example is guided-backpropagation, which can be thought of as an extension or variant of the deconvolution technique [10], [47], [68]. Rather than relying on max-pooling layers, guided-backpropagation uses ReLU activations to correctly attribute importance. This technique was used by Springenberg et al. [68] to visualise relevant parts of an image through combining deconvolution and backpropagation.

Both guided-backpropagation and deconvolution share some flaws in our context, such as that they are only applicable to very specific CNN models [47], [68]. Also, neither method upholds the conservation principle, which refers to when the relevance of the output layer for a given class is fully captured by the contributions of the input features for that same class. This is to say, the relevance redistributed from upper-level nodes to lower-level nodes during the backward pass should be the same as the contributions of the lower-level nodes to upper-level nodes during the forward pass [10]. Both deconvolution and guided-backpropagation focus only on positive relevancies while discarding negative ones, which does not adhere to the conservation principle as relevance is removed from the system. Although this is acceptable for some applications, for our study we would prefer to show the interpretation in its entirety as far as we can because, as will be shown in later chapters, even negative relevancies can hold important information regarding a model's predictions.

The final backpropagating technique we looked into is LRP, which is based on Deep Taylor Decomposition (DTD) [4]. LRP [69] is a conserving (adheres to the conservation principle), backward propagation technique that redistributes the relevance of a model's output back unto its input features through the use of several *redistribution* equations. We further explore the concept of LRP in Section 2.6.

All approaches mentioned in this section generate *local* interpretations with *feature summary visualisation* type results and are *model-specific* [2] [8], [57], with the exception of additive feature attribution techniques that are able to generate both *local* and *global* interpretations, that can be both *model-agnostic* and *model-specific*, and may thus be seen as producing *intrinsically interpretable model*-type results. Whether the addition of sample set analysis is able to change the granularity of an interpretation technique will be determined in the upcoming chapters, specifically Chapter 5.

2.3.4 Evaluating interpretability techniques

The validity of any attribution based interpretability technique should be tested to ensure the correctness and accuracy of the explanations it generates [4], [10]. Visually, for image classification tasks, the importance attributed to each input feature can be organised into visual *feature relevance attribution maps* (henceforth referred to as attribution maps) to compare the efficacy of different techniques [46]. In our study, attribution maps represent the degree to which a feature contributes toward a model's prediction (shown as a shade of blue) or contradicts it (shown as a shade of red). Neutral features that neither contribute nor contradict a model's prediction are represented by the colour green. It is important to note that attribution maps only take the importance values attributed to each input into account when visualising interpretations and not the features themselves. Attribution maps are seen as one of the most common mediums to visualise the interpretations of a network trained on images, such as the MNIST data set and our synthetic data set, as illustrated by Bach et al. [4], and Montavon et al.[10].

Although simple, this method relies on human judgement and is prone to inaccurate assessments. It is typically better to analytically evaluate an interpretability technique, which can be done by assessing the measure to which it upholds the following criteria [70]:

[2]Some authors disagree on the exact definitions of these terms and would rather classify the interpretability approaches provided in Section 2.3.3 as model-agnostic [11].

- ***Accuracy:*** The correctness of an interpretation when generated on data different from the model's training sets.
- ***Fidelity:*** The extent to which an interpretability technique reflects the true behaviour of the model.
- ***Consistency:*** The extent to which the variance of an interpretability technique's results is limited when providing explanations for multiple similar models trained on similar tasks.
- ***Stability:*** Interpretations generated for a specific model on similar predictions for similar inputs should also be similar and not vary significantly between interpretations.
- ***Novelty:*** The extent to which an interpretability technique shows the differences in novel models when compared to already-established models.

Additionally, the practical applicability of an interpretability technique to models using real-world data should be taken into consideration, along with how understandable the explanations generated by the technique are to the end user. Although other criteria, such as comprehensibility, certainty and representativeness exist, we limit ourselves to only the first four criteria as defined by Robnik [70], which we believe to be the most important for this study. Some of the common ways to determine how well an attribution-based interpretability technique fulfills these requirements, which are applicable to our study, are [10], [71], [72]:

- ***Perturbation evaluations:*** This involves masking the features that were deemed the most important by the interpretability technique in some order (typically in descending order), through either removing or changing them. If the technique functions correctly, then the removal of these features should correspond to a noticeable decrease in the model's accuracy [10], [46], [70].
- ***Ground-truth comparisons:*** This refers to creating a controlled testing environment where the model's explanations are already known and can be compared

to the explanations generated by a given interpretability technique. Such environments include generating a synthetic data set as a known baseline for the target model [10], [73], arithmetic toy-problems [10] and user-annotated benchmarks [71]. Despite being a useful tool for testing techniques, it is not always possible to create such an environment that also reflects the complexity of real-world data tasks [9].

- ***Model parameter randomisation test:*** Compares the interpretations of a trained model to the same type of interpretations on an untrained, randomly initialised network with the same architecture. If both the interpretations are similar, then we can deduce that the interpretations do not depend on the model's parameters and would not be suited for tasks that are dependant [72].

- ***Data randomisation test:*** Similar to the previous requirement, this compares the interpretations of a model trained on labelled data to the same type of interpretations on the same model, but with the labels randomly shuffled between samples. If the interpretations are similar, then we can deduce that the interpretations do not depend on the link between the samples and their respective labels [72].

Recent work [74] has shown that it is relatively easy to fool some perturbation-based interpretability techniques into assigning false attributions to features, while the true feature attribution remains unchanged. This vulnerability is especially true when considering proprietary models, where the architecture of the model in question is unknown. In research done by Slack et al. [74], a classifier was built to determine whether or not an input sample had been perturbed by an interpretability technique. If false, the input sample was redirected to a malicious model that was trained to make predictions based exclusively on a protected class, such as gender or race. However, if any input perturbations were detected in this way, the input samples were redirected to a model that was trained fairly, and the resulting explanations reflected the feature importances of the fairly trained model, while the true feature importances of the malicious model in use were mostly able to go unnoticed.

2.4 Sample sets analysis

In some recent work [12], [13], [15], [75], it was shown that fully connected, ReLU activated, feedforward networks have the capacity to delegate their class-specific decision-making to specific nodes and groups of nodes. This delegation tended to become more selective in deeper network layers. Through additional sample set analysis, it was also discovered that some nodes are specific in the sense that they only activate (have a non-zero value) for a limited set of samples [14]. This effect is so strong that unseen samples can even be correctly classified through estimators trained on such nodes.

Any node in the network activates for a specific set of samples. Note that a node is considered active if the activation value is non-zero, irrespective of how small that activation value is. We refer to this set as the node-specific *sample set* [12].

Sample sets typically contain samples of different classes, each individual class forming a *class-specific sample set* at that node. When a sample activates a node, the node provides a non-zero activation value for that sample's set of input features. Using sample set analysis, we can analyse the distribution of samples among the nodes of a network, and also pinpoint what nodes were activated the most for any given class. We also study the classification ability of individual nodes and nodes in collaboration.

Sample sets are created during the forward pass of gradient descent when the network's weights are used to determine which nodes are active. The subsequent backward pass and update phase fine-tunes these weights towards specific classifications [14]. The existence of sample sets, and how they came to be, prompted investigations regarding the effect of gradient-based optimisations on these networks. Such networks are typically seen as single entities, but the above work shows that the different nodes of the network can be viewed to act as separate functions with both local and global elements. *Locally*, each node utilises specific, selected information from the input space, while *globally*, nodes cooperate to solve the overall layer-wide task [12], [13].

Nodes can also be linked together via an *activation path*, which can be seen as the path

leading from the input features to the output layer that activates for a specific sample. From these paths, it is possible to determine the paths and subsets of paths that *maximally activate* for a given class or set of samples.

Networks can be analysed through sample set analysis in conjunction with other explanatory methods such as activation distributions, sample distances, weight and activation gaps, or perplexities with entropy [12], [13], [75]. This was mostly done to study networks in terms of their generalisation ability, but these techniques also offer a modicum of interpretability and could potentially be used alongside other, more common interpretability techniques to obtain useful information.

2.5 Choosing an interpretability technique

Sample set analysis, as discussed in Section 2.4, is still very much a novel concept that has yet to be fully investigated. Our goal with this study is to determine if there is any meaningful information that can be extracted by using sample set analysis as a tool to supplement common interpretability methods. We hypothesise that sample sets generated from a fully trained model can be used to enhance DNN interpretations. We limit the choice of interpretability techniques solely to the approaches discussed in Section 2.3.3.

Perturbation-based approaches are generally very simple, easily implemented techniques, but may become extremely computationally inefficient as the size of the network and the number of required perturbations increase [62]. Similarly to perturbation-based approaches, gradient-based approaches are typically easily implemented and effective, due to their using the gradients naturally calculated by DNN models [58], [65]. However, this means that these techniques are completely dependent on the gradients provided by the model, which could cause future problems as in the case of flat gradients - where gradients change very little, which hampers the generation of quality interpretations [52].

Additive feature attribution approaches, such as SHAP, are generally model-agnostic, utilise locally generated surrogate models to fully explain entire trained models [60], and

are also sometimes able to provide global interpretations.

Backward propagation techniques provide a way to project the relevance from a node in the deeper layers of a network onto the input features [9], [10], [68]. Although this node is typically the output node, the structure of these methods should allow the interpretation of any node in any layer of the network, which subsequently allows us to investigate individual nodes as opposed to the model as a whole. This means that by using a backward propagation technique along with sample set analysis, we are able to analyse and interpret different nodes individually, and potentially find the correlation between a node's sample set size [12] and its impact on the model's classification prediction.

For the purposes of this study, various of the aforementioned approaches could be used to investigate the applicability of sample analysis as a tool for interpretability technique enhancement. However, we choose the backward propagation technique LRP due to its current popularity, broad applicability and excellent performance on various neural network-related tasks.

2.6 Layer-wise Relevance Propagation

LRP is a widely used backward propagation technique that is specifically designed to generate local explanations for neural network structured models with different types of inputs and is thus model-specific. The technique was originally introduced by Bach et al. [4] and was later expanded by Montavon et al. [61], [69] with additional redistribution rules (see Section 2.6.1), which provides greater flexibility when interpreting neural networks.

As discussed in Section 2.3.3, LRP is categorised under the backward propagation approach, as an extrinsic interpretation technique, due to its use of functions that do not form part of the original DNN model it is interpreting. Also as displayed in Figure 2.2, LRP adheres to the conservation principle. The technique uses the prediction of a model to redistribute relevance unto the shallow layers of the network until it reaches the in-

put features [4], [10], [61]. Every node from the output layer back to the input features individually redistributes its relevancies to all of its connected input nodes.

In Section 2.6.1 we discuss the various different rules used by LRP to redistribute relevance, whilst in Section 2.6.2 we delve deeper into where in the network these rules are best applied.

2.6.1 Local redistribution rules

Relevance redistribution occurs according to the application of rules (henceforth called LRP rules) [4], [61], which are comprised of a set of equations that dictate precisely how relevance is redistributed from the redistributing node to the connected input nodes. These rules are primarily based on Eq. 2.1 (repeated in Eq. 2.7), which is used in calculating the activation values of nodes during forward propagation.

$$a_k^{(l+1)} = \sigma(\sum_j z_{jk} + b^{(l+1)}), \quad \text{where} \quad z_{jk} = a_j^{(l)} w_{jk}^{(l,l+1)} \tag{2.7}$$

The z_{jk} term is added to simplify the equation and refers to the contribution of node j in layer l to the activation value of node k in layer $l+1$ [46]. Using Eq. 2.7, we can now define LRP-0 in Eq. 2.8, the most basic LRP rule:

$$R_j^{(l)} = \sum_k \frac{z_{jk}}{\sum_{j'} z_{j'k}} R_k^{(l+1)} \tag{2.8}$$

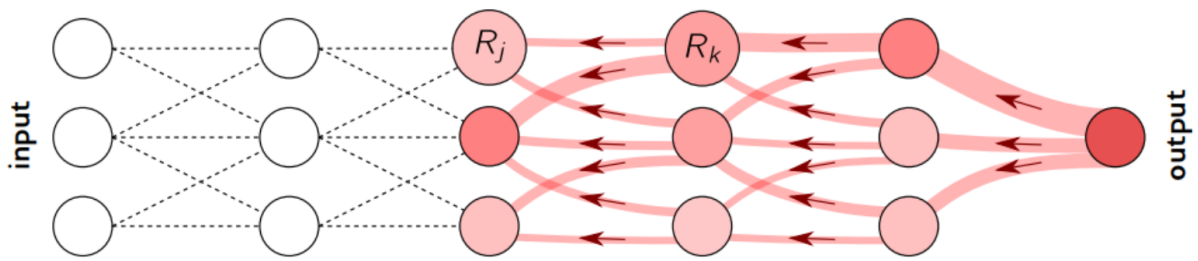


Figure 2.2: LRP example showing the redistribution of relevance unto the lower layers [10], [61].

Here, $R_j^{(l)}$ refers to the relevance value of node j in layer l and subsequently $R_k^{(l+1)}$ refers to the relevance value of node k in layer $l+1$. The rule determines the relevance of node j by dividing the contribution of node z_{jk} by the sum of all node $j^{'}$ contributions in the same layer $\sum_{j'} z_{j'k}$. This value is then multiplied by the relevance of node k in the succeeding layer, which determines the relevance of node j in layer l to node k in layer $l+1$. This process is repeated and summed for every node k in layer $l+1$ node j is connected to, which determines the relevance of node j to the total relevance of layer $l+1$. Through LRP's conservation property, as mentioned in Section 2.3.3, the relevance of node j to layer $l+1$ is then equal to the relevance of node j to the output node.

LRP-0 shows mathematically how the relevance of a node can be redistributed from an upper layer to a lower layer in the network, while also forming the basis from which all other rules in Table 2.1 are derived. The table was originally presented by Montavon et al. [61]. For this study, we will focus only on the primary set of rules described in Section 3.4.2. The rest of the rules were added for the sake of completion.

It should also be noted that these local redistribution rules are multiplicative by nature. All equations are multiplied by the relevance of the previous layer in order to obtain the relevance of the current layer. This ensures that the total relevance of each layer remains the same.

2.6.2 Layer Groups

During training, the layers of a network contain an abstract representation of the provided input features. These representations become more entangled and class-specific between nodes in the deeper layers of the model [12], [61]. The layers of the model may be categorised into the upper, middle or lower layer groups depending on how entangled the representations are, or how close the given layer is to the input features. However, due to LRP's flexibility, the precise point at which the transition between layer groups occurs depends on the network and its properties [61]. We refer the reader to Section 3.4.2 for information on how we divided our models' layers between the three groups.

Table 2.1: LRP rules used for this study. Table originally presented in [61]. Here $z_{jk} = a_j^{(l)} w_{jk}^{(l,l+1)}$ as in Equation 2.7, and the notation $(.)^+ = max(0,.)$ and $(.)^- = min(0,.)$. l_j and h_j refer to the lowest and highest admissible input values and All weights w are assumed to be between the layers $(l, l+1)$. Note that all rules are derived from DTD except for LRP-$\alpha\beta$, which is derived from DTD only for the case $\alpha = 1, \beta = 0$.)

Name	**Formula**	**Layers**
LRP-0 [4]	$R_j^{(l)} = \sum_k \frac{z_{jk}}{\sum_{j'} z_{j'k}} R_k^{(l+1)}$	Output
LRP-ϵ [4]	$R_j^{(l)} = \sum_k \frac{z_{jk}}{\epsilon + \sum_{j'} z_{j'k}} R_k^{(l+1)}$	Upper/Middle
LRP-γ [61]	$R_j^{(l)} = \sum_k \frac{a_j^{(l)}(w_{jk} + \gamma w_{jk}^+)}{\sum_{j'} a_{j'}^{(l)}(w_{j'k} + \gamma w_{j'k}^+)} R_k^{(l+1)}$	Lower
LRP-$\alpha\beta$ [4]	$R_j^{(l)} = \sum_k (\alpha . \frac{(z_{jk})^+}{\sum_{j'}(z_{j'k})^+} - \beta . \frac{(z_{jk})^-}{\sum_{j'}(z_{j'k})^-}) R_k^{(l+1)}$	Lower
w^2-rule [69]	$R_j^{(l)} = \sum_k \frac{w_{jk}^2}{\sum_{j'} w_{j'k}^2} R_k^{(l+1)}$	Input
z^β-rule [69]	$R_j^{(l)} = \sum_k \frac{x_j^{(l)} w_{jk} - l_j^{(l)} w_{jk}^+ - h_j^{(l)} w_{jk}^-}{\sum_{j'} x_{j'}^{(l)} w_{j'k} - l_j^{(l)} w_{j'k}^+ - h_j^{(l)} w_{j'k}^-} R_k^{(l+1)}$	Input

Although singular LRP rules may be applied to interpret entire networks, they are best used in combination with one another. The reason for this is that each rule is best suited to a particular layer group within the network. See Table 2.1 for reference. A big reason for having different redistribution rules is to highlight only the most important inputs by, in part, ignoring nodes with negligible information [61], [76].

Within the upper layers, the nodes' representation of the input features can be seen as a mathematical combination of the representations by nodes from previous layers [12]. These features are entangled within the nodes in the upper layers to the point that they contain both the relevant and non-relevant representations, according to the LRP

interpreter. A way to separate these representations is to continuously backpropagate them until the nodes containing relevant representations once again become disentangled from the non-relevant ones like in the shallower layers [61]. This can be done using the LRP-0 redistribution rule as it simply backpropagates the representations without modifying them like some of the other redistribution rules, making them insensitive to these overlapping concepts [4].

The layer at which the different representations separate from one another is generally where the network transitions from the upper layers to the middle layers, which form the bulk of the model's layers. Here, features are less entangled, but the different layers can have varying or unimportant classification concepts. Suitable redistribution rules, like LRP-ϵ, can be applied to eliminate or lessen the impact of nodes containing negligible information without fear of ignoring the important information; which would have otherwise been entangled within the upper layers [4], [61]. This occurs through the ϵ term increasing the size of the function's denominator, which subsequently decreases the size of the fraction due to the numerator remaining constant. Increasing the ϵ term will further decrease the size of the fraction, which lessens the impact of the specific node's contribution z_{jk}. LRP-ϵ works well as it also accounts for any numerical instabilities.

Despite LRP providing interpretations containing both positive and negative relevancies, positive relevancies are still seen as the important component of an interpretation. As the layers get closer to the inputs, specific asymmetric rules such as LRP-$\alpha\beta$ or LRP-γ can be used to highlight positive relevancies above negative relevancies without breaking the conservation property LRP adheres to. Lower layer rules are similar to the middle layer rules but are additionally used to make the explanation more coherent for humans. This is typically achieved by the rules highlighting the positive or negative relevancies depending on the rule and how it is applied [61].

2.7 Conclusion

In this chapter we briefly discussed the functionalities behind DNNs. We continued to describe the role of interpretability in DNNs, followed by an overview of some of the attribution-based interpretability techniques applicable to our study, while also introducing the concept of sample sets analysis. After considering the different interpretability options, we selected LRP and described the technique in detail.

Chapter 3

Tools and data sets

Description of experimental setup and initial interpretability codebase implementation and evaluations.

3.1 Introduction

In this chapter we discuss the study's experimental environment, which includes the different data sets, tools, and codebases to be used, as well as the configuration of the in-house LRP interpreter. We additionally verify the correctness of the interpreter.

Further information regarding our development environment is provided in Section 3.2. We describe the data sets and define the models used throughout the upcoming experiments in Section 3.3. In Section 3.4, we describe the implementation and verification of the in-house LRP interpreter. Afterwards, we visualise and analyse the interpretations of the defined networks via the use of generated attribution maps in Section 3.5 in order to demonstrate the use of LRP in practice.

3.2 Development environment

The Python [77] programming language was used to create and develop additional tools when they were needed during our study. All experiments were performed using PyTorch [41], an open source machine learning framework that integrates with Python, which allows the development and training of DNNs. Matplotlib [78], an open source plotting library, was also used to visualise the results of our different experiments. Additionally, mustnet [1], a PyTorch-based in-house codebase, was utilised for MLP optimisation and to generate a synthetic data set (Section 3.3.1). This codebase was extended as part of this study to include an independently developed LRP toolkit for generating interpretations and investigating the inner workings of MLPs. Captum [79], a model interpretability and understanding library for PyTorch, was used to compare the results generated by our LRP toolkit.

3.3 Models and data sets

In this section, we discuss the models and data sets used throughout this study.

We give a detailed description regarding the creation of our custom synthetic data set in Section 3.3.1 and provide reasons for the decisions made during the process. In Section 3.3.1 we describe the MNIST data set. Lastly, in Sections 3.3.2 and 3.3.3, we define the network architecture and hyperparameter values of several networks developed and trained on the MNIST and synthetic data sets respectively, along with their performance values.

3.3.1 Data sets

Here we describe the MNIST and synthetic data sets as used in our study. As is standard practice, the samples of each data set are further divided into one of three data set

[1] `https://bitbucket.org/must_research/mustnet3`

partitions, for the purposes of training our DNN models. Typically, the goals of the different data set partitions are the following [16]:

- ***Training set:*** Used to initially fit a model and train model parameters.
- ***Validation set:*** Used to tune hyperparameter values and determine the generalisability of a model.
- ***Evaluation set:*** A final test to determine a model's performance on previously unseen data.

Once the models have been trained, we conduct upcoming experiments on the training and validation set. However, in the interest of brevity we limit the results of this study mainly to those of the training set. We do, however, include results on all three data sets where we feel this is applicable, and specifically add the results of the evaluation set for the interpretation enhancement technique we have developed in Section 5.5 to give an indication of its performance on data it was not specifically trained on. Unless otherwise specified, the reader can assume any results provided throughout this study were generated using the *training set*.

Describing the synthetic data set

We generated a custom synthetic data set to demonstrate the capabilities of our LRP codebase. Through using a self-generated custom synthetic data set, we are able to control the sample features used for generating interpretations. This allows us to better determine the accuracy and fidelity of our interpreter before applying it to a network trained on the MNIST data set, which is closer to how a data set would present itself in practice.

When creating a synthetic data set, we want to know which features are most important for classification before performing any type of interpretation. This allows us to assess whether the interpretations generated by the in-house LRP interpreter are reliable by

comparing them to what we know they should be. Secondarily, investigating the interpreter's performance on a network trained on the synthetic data set will also help us develop an intuition of how our interpreter will eventually perform on the MNIST data set.

To do this, we created a data set consisting of several trigonometric functions that closely resemble MNIST while being less visually complex with fewer classes that are more distinct from one another. Each sample of the synthetic data set is created by independently generating *amplitude* values, using Eq. 3.1 below.

$$amplitude = trig(period + freq) * max\_amplitude \tag{3.1}$$

Where *trig* refers to one of the following six trigonometric functions: sine, cosine, tan, cosec, sec and cot. The *period* is equal to 2π which, under normal conditions, is a full cycle of any of the trigonometric functions. *freq* refers to the frequency of the function, which is sampled from a uniform distribution $\mathcal{U}(0, \frac{\pi}{2})$, while *max_amplitude* refers to the maximum amplitude value for the given function and is similarly sampled from another uniform distribution $\mathcal{U}(1, 3)$. By sampling the maximum amplitude and frequency values from the chosen uniform distributions, we ensure that there will be some variation between the different samples of the same class.

For every sample, we generate and add 100 amplitude values to a single list, which is then used along with Matplotlib's scatter function to generate a 30 by 30 pixel image plot of one of the different trigonometric functions. Each sample thus contains a total of 900 (30×30) features. These plots are then altered to represent their respective trigonometric functions in white on a black background, as shown in Figure 3.1, to mimic MNIST. Afterwards, each plot is converted from an image to a 2-dimensional list consisting of 30 rows and columns holding each pixel's base numerical value, which ranges from 0 - 255. Lastly, each list is then normalised between *[0, 1]* by dividing all pixel values by 255.

We repeat this process until we have generated a synthetic data set that consists of 60 000 training samples, and 10 000 validation and evaluation samples. In summary, the

synthetic data set can be seen as a series of 30 by 30 value lists, divided evenly between six trigonometric function classes. This means that for each function class, there are 6 000 training samples, and about 1 666 validation and evaluation samples.

When generating interpretations using this synthetic data set, the interpreter should, ideally, focus on the features that distinguish the classes from one another. This includes the general presence or absence of features in one class, but not in others. For instance, the interpretations of samples from the *sine* class should primarily highlight the lack of features in the bottom left and top right of each sample. Similarly, the interpretations of samples representing the *cosine* class should primarily highlight the lack of features in the middle of each sample, while also indicating either the lack of features near the bottom edges, or the general shape of the function. It is easier to first evaluate whether our interpreter truly isolates these distinguishing features using the synthetic data set than on the MNIST data set, which contains multiple classes with similar features such

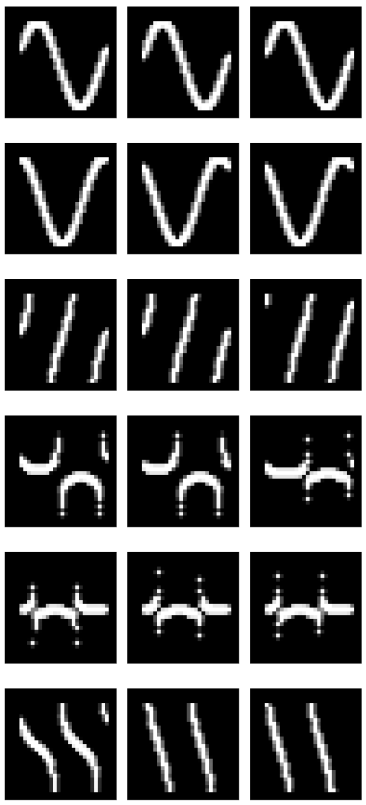


Figure 3.1: Synthetic data set sample examples. From top to bottom, each row shows examples for the following classes: sine, cosine, tan, cosec, sec and cot.

as two, three and eight.

Describing the MNIST data set

MNIST [17] is a simple image data set consisting of 70 000 samples of handwritten digits. Each sample is represented as a 28 by 28 image that displays a single digit from zero to nine, which forms the ten total classes of the data set, as white in the centre of a black background. Every sample thus contains a total of 784 (28×28) features. For this study, the data set is partitioned into 55 000 training samples, 5 000 validation samples and 10 000 evaluation samples. The data set is well known in the machine learning community and is frequently used to test new concepts and ideas because of its simplicity and ease of use. This makes the data set an ideal choice for our study. We refer the reader to Table 3.1 for information regarding the main differences between MNIST and our custom-generated synthetic data set.

3.3.2 Defining the synthetic model

We train a set of suitable, yet simple, MLP networks on the synthetic data set described in Section 3.3.1 as close to 100% accuracy on the training and validation sets as possible. The use of a ReLU activation function at each layer, the Cross-Entropy loss function, the Kaiming initialisation scheme, a learning rate of 0.01, and a batch size of 64 were kept consistent across all networks. We performed a grid search over the following hyperparameters: layer width, network depth, whether to use an SGD or Adam optimiser, and

Table 3.1: Training, validation and evaluation set partitioning and number of features per sample for the data sets used within this study.

Data set	Num. features per sample	Training set size	Validation set size	Evaluation set size
MNIST	784	55 000	5 000	10 000
Synthetic	900	60 000	10 000	10 000

whether or not to use a bias term in the first layer. All data set samples are normalised between the minimum and maximum values of *[0, 1]*.

Due to the simplicity of the synthetic data set, all networks were able to fully interpolate the data set within one epoch (achieve 100% training accuracy). Among those trained to interpolation, we select one network for further analysis according to the lowest loss achieved on the validation set.

The architecture of the selected network and its performance values are provided in Table 3.2. The performance values of this network trained over additional initialisation seeds are shown in Table 3.3 to demonstrate that it obtained consistent results.

The selected network is trained using an SGD optimiser, and its hidden layers are optimised for a layer width of 20 nodes on the first layer and 10 nodes on every subsequent hidden layer. A bias term is used in the first layer.

Since our long-term goal is to investigate the inner working of our chosen DNN, we refrain from using any form of explicit regularisation that might interfere with our investigations. This is done in a similar manner to the procedure used by Davel et al.[12].

Table 3.2: Architectural and performance values for the chosen synthetic data set network.

Hyperparameter	**Value**	
Input dimensions	900	
Output dimensions	6	
Layer dimensions	20 10 10 10	
Epochs trained	1	
Sets	**Accuracy**	**Loss**
Train	1.00000	0.00025
Valid	1.00000	0.00024
Eval	1.00000	0.00025

Table 3.3: Performance values for other seeds of the chosen synthetic data set network.

	Seed 1		**Seed 2**		**Seed 3**		**Seed 4**	
	Acc	**Loss**	**Acc**	**Loss**	**Acc**	**Loss**	**Acc**	**Loss**
Train	1.00000	0.00030	1.00000	0.00029	1.00000	0.00025	1.00000	0.00017
Valid	1.00000	0.00030	1.00000	0.00029	1.00000	0.00025	1.00000	0.00016
Eval	1.00000	0.00030	1.00000	0.00029	1.00000	0.00025	1.00000	0.00016

In contrast to the MNIST model (as defined in Section 3.3.3), we opted to use a smaller architecture for our synthetic model. Despite having a larger number of input features per sample when compared to the MNIST data set (900 vs. 784), the synthetic set contains more training samples (60 000 vs 55 000) with fewer variations between samples, along with fewer, more distinct classes. This results in a much simpler data set than MNIST.

As briefly discussed in Section 3.3.1, the purpose of the synthetic network is to be used alongside the MNIST model, as will be described in Section 3.3.3, demonstrating how the LRP codebase can be applied and evaluated.

3.3.3 Defining the MNIST model

Similar to the synthetic networks described in Section 3.3.2, we train a set of simple MLP networks on the MNIST data set as close to 100% accuracy on the training and validation sets as possible. The use of a ReLU activation function at each layer, the Cross-Entropy loss function, the Kaiming initialisation scheme, a learning rate of 0.01, a batch size of 64, and a layer depth of 10 were kept consistent for all networks. We perform a grid search of the following hyperparameters: layer width, optimiser, and whether or not a bias should be included in the first layer or not.

Each sample of the MNIST data set is normalised between the minimum and maximum values of *[-1, 1]*. This differs from the samples of the synthetic data set in Section 3.3.2, which are normalised between the values of *[0, 1]*. The reasoning for this deviation has

to do with the effect different normalisation values have on the interpreter, as will be discussed in more detail in Section 3.5.2.

The networks are purposefully given an overly sufficient number of 10 hidden layers for analysis purposes and to better resemble the network used by Davel et al. [12], which will be helpful when replicating their results in Section 4.2. A final network is chosen for further exploratory analysis based on the lowest validation loss achieved.

The selected network's architecture and performance values are provided in Table 3.4. The performance values of this network over different seeds are shown in Table 3.5 to demonstrate that the network is consistent.

Similar to the synthetic network, the chosen MNIST network is trained to interpolation with a bias term in the first layer, using SGD without using any regularisation methods during training. The hidden layers are optimised for a layer width of 100 nodes per layer.

It has been noted that training an overly sufficient network, such as the one defined in this section, on a simple task such as MNIST, encourages overfitting [16], [26]. Due to the large amount of trainable parameters, the network is able to fit itself 'too' well on the

Table 3.4: Architecture and performance values for the chosen MNIST data set network.

Hyperparameter	**Value**	
Input dimensions	784	
Output dimensions	10	
Layer dimensions	100 100 100 100 100 100 100 100 100 100	
Epochs trained	81	
Sets	**Accuracy**	**Loss**
Train	1.00000	0.00017
Valid	0.98400	0.11806
Eval	0.98160	0.21803

Table 3.5: Performance values for other seeds of the chosen MNIST data set network.

	Seed 1		**Seed 2**		**Seed 3**		**Seed 4**	
	Acc	**Loss**	**Acc**	**Loss**	**Acc**	**Loss**	**Acc**	**Loss**
Train	1.00000	0.00006	1.00000	0.00004	1.00000	0.00009	1.00000	0.00011
Valid	0.98200	0.12944	0.98540	0.11301	0.98540	0.13077	0.98140	0.14342
Eval	0.97970	0.30338	0.98030	0.28607	0.98250	0.30731	0.98030	0.24537

training data, which subsequently makes it harder for the network to generalise to new data. This explains the larger losses seen on validation and evaluation sets in Tables 3.4 and 3.5. We acknowledge this fact and will continue to use these networks for further analysis as it should not interfere with our goal of investigating the inner workings of the network nor obscure its results when used in the forthcoming chapters.

3.4 The LRP interpreter codebase

In Section 3.4.1 we provide reasons for implementing our own in-house LRP interpreter codebase, rather than just using an already-existing, publicly available LRP codebase.

Furthermore, in Section 3.4.2, we provide the configurations used when generating interpretations using our interpreter codebase. Despite not using an external LRP codebase, in Section 3.4.3 we verify our interpretations by comparing our results to a publicly available, pre-established LRP codebase.

3.4.1 Reasons behind implementing our interpreter

We required a functional LRP interpreter codebase to investigate the inner workings of sample sets and their function within DNNs. This meant that the chosen codebase needed to be easily understandable, but also allow alterations to its functionality to better suit some of our later experiments.

We investigated two publicly available codebases, namely, iNNvestigate [80], and the LRP toolkit implemented in the Captum.ai (henceforth referred to as Captum) Python package [79]. Regrettably, both codebases were limited in terms of their functionality. INNvestigate proved very difficult to work with, as little information is given regarding its internal operations. The amount of time required to fully understand the codebase and integrate it with our own would have been too great an investment.

Captum, on the other hand, is much more streamlined and integrated well with our existing codebase. Unfortunately, Captum's LRP toolkit was only recently added to the rest of the codebase, and consequently contained only a few of the LRP rules needed for our study. In addition, although the codebase integrated well with our own, it suffered from the same problem as iNNvestigate: trying to change some of its internal operations proved difficult, and documentation is again limited. Later, as we will show in Section 3.4.3, it was determined that one of the LRP rules was incorrectly implemented.

After failing to find a suitable codebase, we determined that it would be prudent to rather create our own in-house LRP interpreter. By independently developing our own interpreter, we not only ensure that it will operate as is required for this study, but that it will also help us gain some intuition of how LRP functions as a feature attribution tool.

Although we cannot directly use the aforementioned external codebases for the experiments presented later on in this dissertation, we can still use them to verify the results of our own interpreter and ensure it is up to par with state-of-the-art LRP implementations. We do this by comparing the results of our implemented interpreter to that of Captum, as will be discussed in Section 3.4.3. We chose Captum because it was easier to work with than iNNvestigate.

3.4.2 Interpreter configurations

As described in Section 2.6.1, LRP interpretations become more accurate when multiple LRP rules are appropriately used in conjunction with one another. However, some rules, specifically rules developed for the input layers of different models, are only used for

certain DNN applications. This restricts which model archetypes the interpreter can be used with [61]. Thus, we limit ourselves to only the most commonly used rules first listed in [4], at their most appropriate layer groups as advised by Table 2.1, which was originally presented in work done by Montavon et al. [61]:

- The *LRP*-0 at the *output layer*.
- The *LRP*-ϵ at the *upper* and *middle layers*.
- The *LRP*-$\alpha\beta$ at the *lower* and *input layers*.

The upper, middle and lower layers are defined as in Section 2.6.2. Seeing as there is no standardised method to determine where one layer group transitions to another, we use one of the experiments performed in Davel et al. [12], and referred to in Section 4.3, to empirically analyse where the model transitions from one layer group to another. This is done by monitoring the change in network node behaviour from layer to layer. The different layer groups of our models as selected using this heuristic technique are shown in Table 3.6 (a more detailed analysis was not performed).

Following the works of other authors [10], [60], we apply the commonly used values of 2, 1 and 1^{-16} to the α, β and ϵ constants, respectively. We select these values as they were used in similar applications and can be chosen independently from the network's architecture.

Table 3.6: Network layers and LRP rules assigned to each layer grouping. From first to last row, the network layers are given in order of: input layer, first hidden layer to last hidden layer and output layer.

	MNIST Network	**Synthetic Network**	**LRP rule**
Input Layer	784	900	LRP-$\alpha\beta$
Lower Layers	100-100	20	LRP-$\alpha\beta$
Middle Layers	100-100-100-100-100	10-10	LRP-ϵ
Upper Layers	100-100-100	10	LRP-ϵ
Output Layer	10	6	LRP-0/LRP-base

3.4.3 Comparison with Captum

In order to verify the implementation of our interpreter, we compare its interpretations of the MNIST network to those generated by the LRP implementation in Captum. See Section 3.5 for a more detailed discussion of the interpretations generated by our interpreter.

Protocol

We first perform a side-by-side comparison of the attribution maps generated from a single interpretation of both codebases on the training set of the following MNIST classes: zero, three, six and nine. Secondly, we measure the similarity of interpretations from the two systems for additional samples from the training set on the same classes. This is accomplished by calculating the average cosine similarity scores between each pair of interpretations from both codebases.

The cosine similarity score measures the cosine of the angle between two vectors in a multi-dimensional space [81], where the cosine similarity between the two vectors $\mathbf{x}$ and $\mathbf{y}$ can be defined as:

$$sim(\mathbf{x}, \mathbf{y}) = \frac{\mathbf{x} \cdot \mathbf{y}}{\|\mathbf{x}\|\|\mathbf{y}\|} \quad \text{where} \quad \|\mathbf{x}\| = \sqrt{x_1^2 + x_2^2 + ... + x_n^2} \tag{3.2}$$

where $\|\mathbf{x}\|$ is the Euclidean norm of vector $\mathbf{x}$, and n its total number of elements. Similarly, $\|\mathbf{y}\|$ is the Euclidean norm of vector $\mathbf{y}$.

In its current state, Captum's LRP implementation is limited to the LRP-$\alpha\beta$, LRP-γ and LRP-ϵ rules shown in Table 2.1. Therefore, we compare the two codebases through interpretations generated by using each rule alone, as well as in combination with one another as specified in Table 3.7.

Table 3.7: Network layers and LRP rules assigned to each layer grouping of the MNIST network. This configuration is used for the sole purpose of comparing the interpretations generated on this network by our interpreter to those of Captum. From the first to the last row, the network layers are given in order of: input layer, first hidden layer to last hidden layer and output layer.

	MNIST Network	**LRP rule**
Input Layer	784	LRP-γ
Lower Layers	100-100	LRP-$\alpha\beta$
Middle Layers	100-100-100-100-100	LRP-ϵ
Upper Layers	100-100-100	LRP-ϵ
Output Layer	10	LRP-ϵ

Analysis using positive inputs

We begin with a comparison for the case where the data set samples are normalised between *[0, 1]*. Following the protocol set out earlier in this section, we provide the attribution maps in Figure 3.2, and the average cosine similarity between interpretation pairs in Table 3.8.

The attribution maps for each set of interpretations appear to be identical, which is supported by an above 0.996 average cosine similarity for each class.

Table 3.8: Average cosine similarity scores between our and Captum's interpretations when using a normalisation range of *[0, 1]*.

Sample:	**LRP-ϵ only:**	**LRP-γ only:**	**LRP-$\alpha\beta$ only:**	**Combination:**
0	0.99958	0.99999	1.00000	1.00000
3	0.99904	0.99999	1.00000	0.99999
6	0.99597	0.99996	1.00000	0.99997
9	0.99920	0.99999	1.00000	1.00000

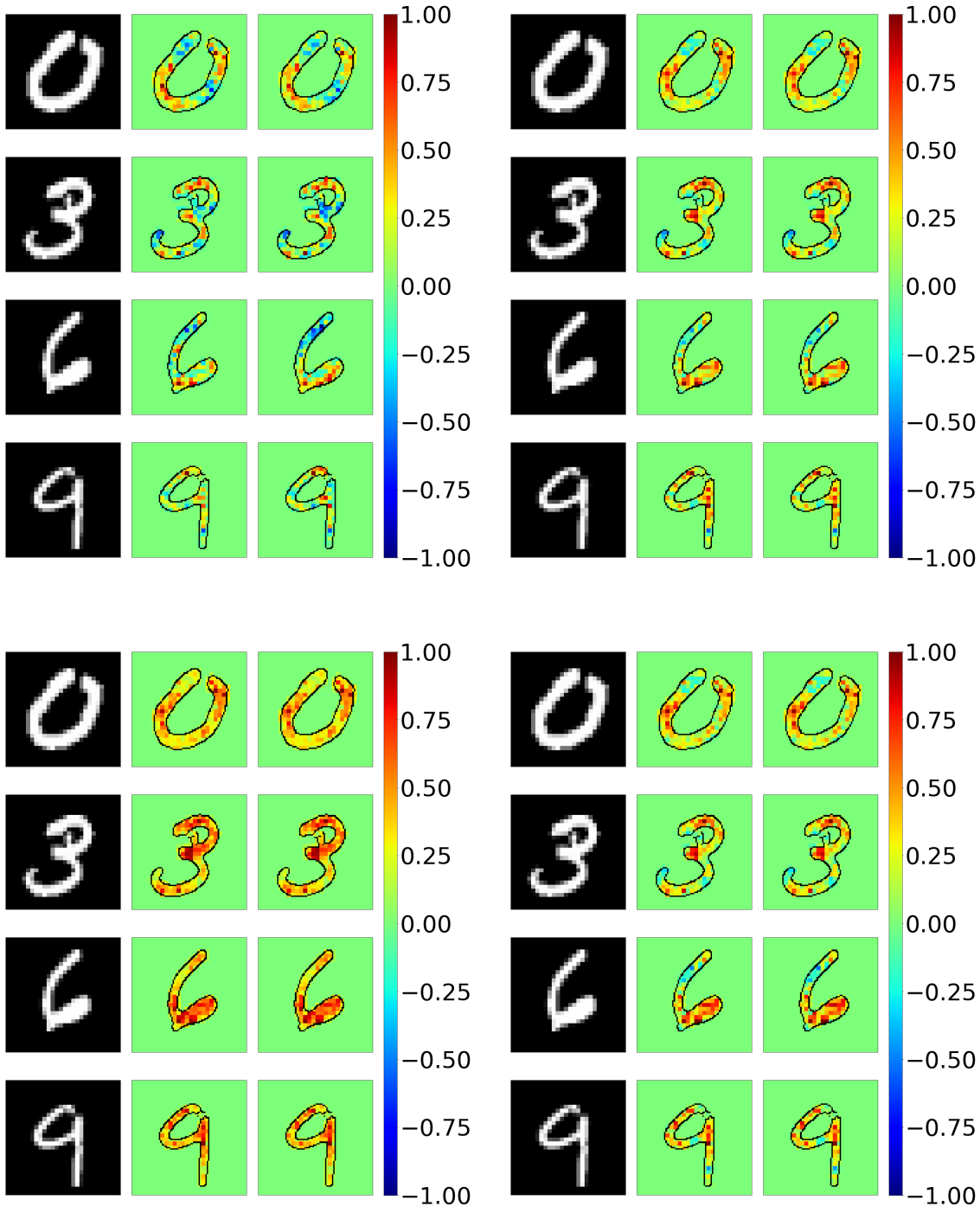


Figure 3.2: Interpretations generated by Captum (middle column) and our in-house LRP codebase (right column) for the training set of our MNIST model with the data set samples normalised between *[0, 1]*. Each pair of interpretations left to right, top to bottom uses the following rules: only the LRP-ϵ rule, only the LRP-γ rule, only the LRP-$\alpha\beta$ rule and a combination of each rule as specified in Table 3.7.

Analysis using positive and negative inputs

As will be discussed in Section 3.5.2, we will mostly be working with data sets normalised between *[-1, 1]* throughout the study, which will allow us to consider all features of each sample.

Therefore we also create comparisons for the case where the model's data sets are normalised between the values of *[-1, 1]*. The attribution maps for this comparison are given in Figure 3.3, and the average cosine similarity scores in Table 3.9.

The interpretations from both codebases are essentially identical except for when using the LRP-$\alpha\beta$ rule alone, which appears to be generating nonsensical interpretations with regard to Captum's codebase. The cosine similarity scores also show an averaged similarity of about 1.000 and above for all classes, except for when using LRP-$\alpha\beta$ rule alone, where this value decreases to 0.953 at its lowest. Despite the large visual differences between the different LRP-$\alpha\beta$ interpretations, their average cosine similarity still remains relatively high. This could be due to the similarities between the background features having a large impact and skewing the results. However, when the LRP-$\alpha\beta$ rule is used in conjunction with other rules, the attribution maps for the two codebases are once again almost identical. We investigate this discrepancy in the next section.

Table 3.9: Average cosine similarity scores between our and Captum's interpretations using a normalisation range of *[-1, 1]*.

Sample	**LRP-ϵ only**	**LRP-γ only**	**LRP-$\alpha\beta$ only**	**Combination:**
0	0.99996	0.99996	0.96437	0.99999
3	1.00000	1.00000	0.96371	0.99998
6	1.00000	0.99974	0.95260	0.99999
9	0.99990	0.99997	0.96766	0.99985

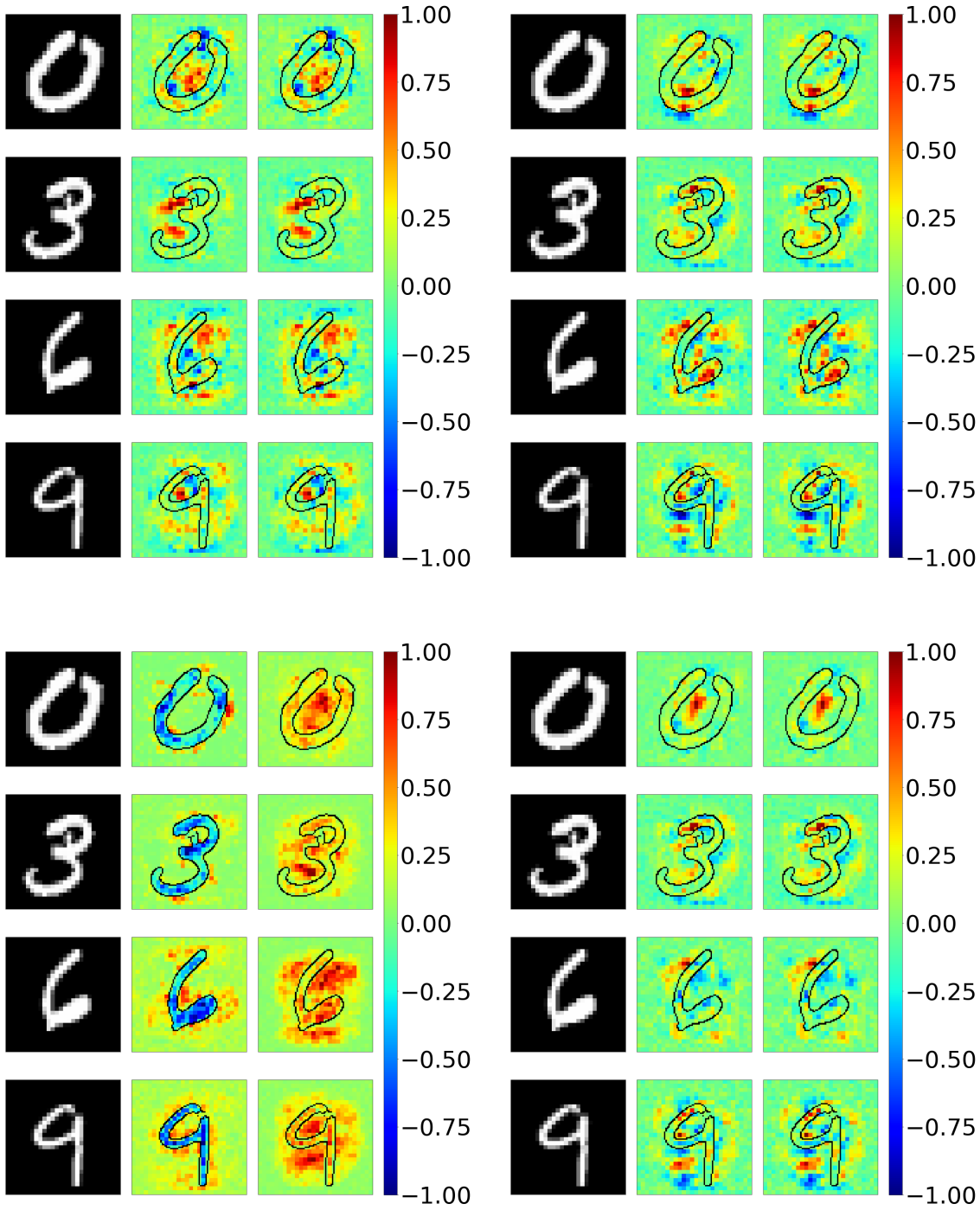


Figure 3.3: Interpretations generated by Captum (middle column) and our in-house LRP codebase (right column) for the training set of our MNIST model with the data set samples normalised between *[-1, 1]*. Each set of interpretations left to right, top to bottom uses the following rules: only the LRP-ϵ rule, only the LRP-γ rule, only the LRP-$\alpha\beta$ rule and a combination of each rule as specified in Table 3.7.

LRP-$\alpha\beta$ rule discrepancy

The LRP-$\alpha\beta$ rule is asymmetric, in the sense that it multiplies an α and β term with the positive and negative components of the equation respectively. In order to uphold to the conservation property, for which LRP is known (Backward propagation heading in Section 2.3.3), the α and β terms must adhere to the following relationship:

$$\beta = \alpha - 1 \tag{3.3}$$

This results in four distinct scenarios:

- $\alpha > 0.5$: The positive components are emphasised above the negative components.
- $\alpha < 0.5$: The negative components are emphasised above the positive components.
- $\alpha = 0.5$: Both the positive and negative components are multiplied by the same constant value, resulting in neither of them being emphasised above the other.
- $\alpha = 1$: The negative components are multiplied by a zero-value beta constant resulting in only the positive components being displayed.

When using a constant value of 0.5 for the α term, the LRP-rule does not fully utilise its asymmetric characteristics.

The LRP-$\alpha\beta$ rule in Captum is only implemented for the use case where $\alpha = 1$ and $\beta = 0$. Figure 3.2 clearly shows Captum's interpretation eliminating all negative relevancies while its interpretation in Figure 3.3 does not.

Through investigating the codebase, we discovered that Captum does not separate the positive and negative components strictly after calculating Eq. 2.7 within the LRP-$\alpha\beta$ rule as initially thought. Rather, Captum separates the positive and negative components for the weight term w only *before* performing the rest of the calculation, which appears to be incorrectly including negative relevancies in the interpretations when none should

be present. However, this only occurs when samples with negative values (normalised between *[-1, 1]*) are introduced into the data set. The addition of these negative relevancies indicates an incorrect implementation of the LRP-$\alpha\beta$ rule on the part of Captum.

To test this hypothesis, we alter the LRP-$\alpha\beta$ rule of our in-house interpreter to function in this manner. Figure 3.4 shows the attribution maps of the interpretations using only the LRP-$\alpha\beta$ rule, after we changed our own interpreter to divide the positive and negative components in the same way we assume Captum does. As before, the average cosine similarities between interpretations for the different classes are also included in Table 3.10.

Now the interpretations from both codebases are identical, indicating that the suspected implementation of the LRP-$\alpha\beta$ rule for the Captum codebase is indeed as described earlier. The results shown in this section verify our in-house implementation of an LRP interpreter, with the exception of the LRP-$\alpha\beta$ rule.

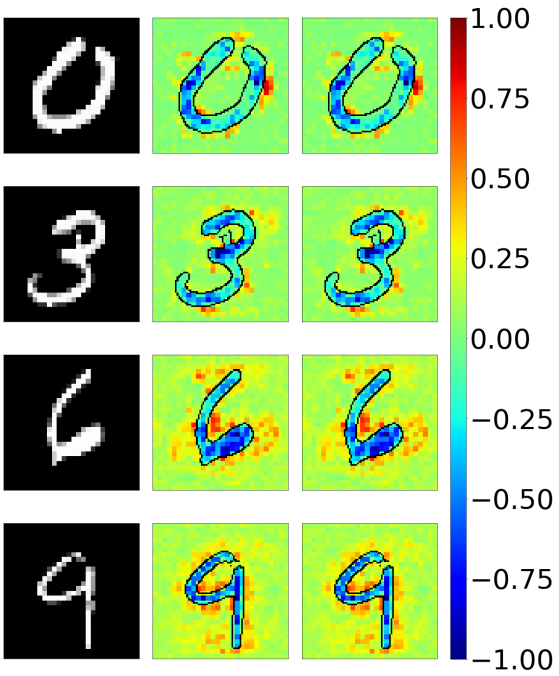


Figure 3.4: Interpretations using only the LRP-$\alpha\beta$ rule for Captum (middle column) and our altered in-house LRP codebase (right column) for the training set of the MNIST model with data set samples normalised between *[-1, 1]*.

Table 3.10: Average cosine similarity scores between our and Captum's interpretations when using only the LRP-$\alpha\beta$ rule along with a normalisation range of *[-1, 1]*, after the LRP-$\alpha\beta$ rule of our interpreter has been temporarily altered to match Captum's suspected behaviour.

Sample	**LRP-$\alpha\beta$ only**
0	1.00000
3	1.00000
6	1.00000
9	1.00000

Regarding the LRP-$\alpha\beta$ rule: our implementation correctly eliminates negative relevancies for both normalisation ranges, while only redistributing positive relevance for the case $\alpha = 1$ and $\beta = 0$. This is what we expected from the rule equation (Table 2.1), and is more in line with the results found in other literature [4]. Thus for all upcoming experiments, we will be relying on our implementation of the LRP-$\alpha\beta$ rule.

3.5 Model interpretation results

We already verified the implementation of the LRP codebase by comparing it against Captum in Section 3.4.3. Additionally, we also need to test the fidelity and accuracy of its interpretations on a trained neural network.

In this section, we discuss and evaluate our in-house LRP codebase by first applying it to the synthetic network in Section 3.5.1. Afterwards, we examine the effects of different normalisation values on the interpretations in Section 3.5.2 and analyse their correctness in Section 3.5.3. Lastly, after verifying the interpreter, we substitute the role of the synthetic data set with the MNIST data set by similarly applying the interpreter to the MNIST network in Section 3.5.4 and thereby determining the consistency and stability of the generated interpretations in Section 3.5.5.

3.5.1 Synthetic model attribution maps

Using the synthetic model along with our implemented interpreter, we start by generating a set of four local attribution maps (as described in Section 2.3.4), for random samples from each of the six classes. In doing so, we develop an intuition with regard to the general interpretation of each class. These attribution maps are shown in Figure 3.5.

The generated attribution maps seen in Figure 3.5, help us to visualise the distribution of the relevance attributed to input features by the interpreter. This gives us an impression of where the classifier is 'looking' during inference, as well as providing an idea of the effectiveness of the interpreter. The relevance, as redistributed by the interpreter, seems to be isolated to only the foreground features (white features). Although correct, this is a limited view of the classifier as it does not take all the features of the sample into consideration when allocating relevance. Currently, it is unclear whether the highlighted features

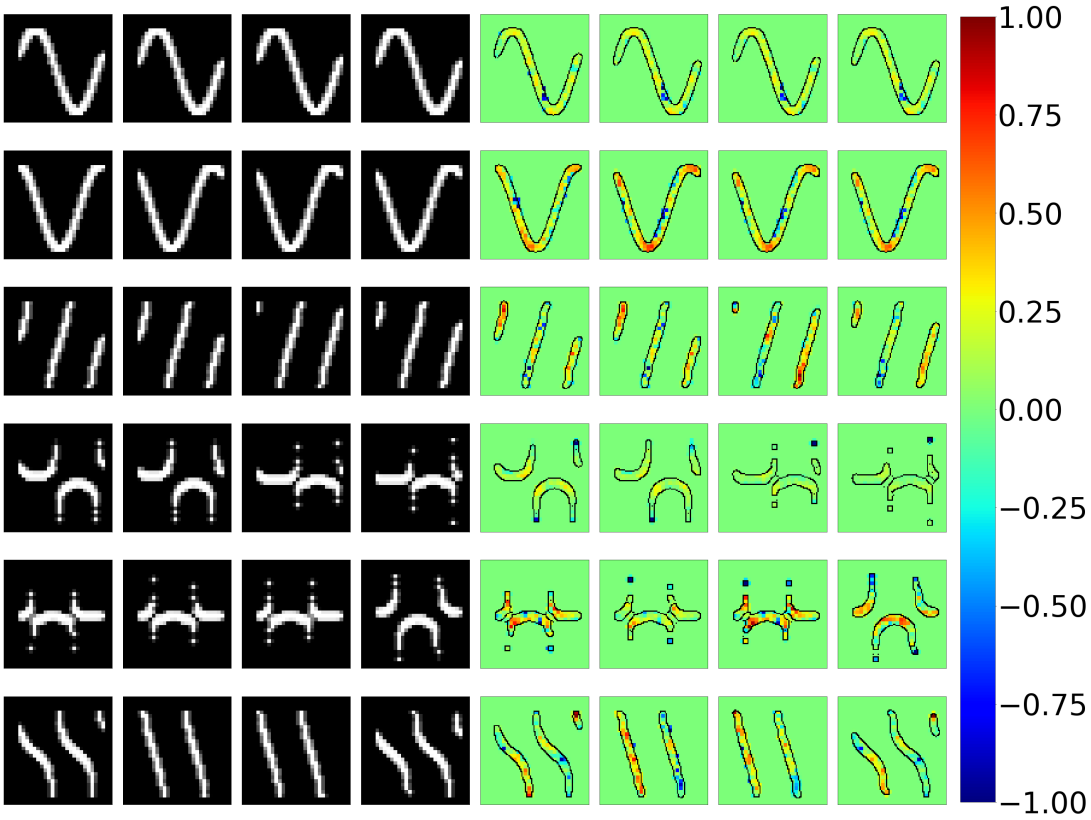


Figure 3.5: Initial attribution maps (right) of four different samples of each class (left) of our synthetic data set. Interpretations are made on a network with a normalisation range of *[0, 1]*.

are truly the most or only important features for the respective samples' classifications.

As a reminder to the reader, here the input features are normalised between *[0, 1]*. This combination of a zero lower boundary input value, which coincidentally in the case of MNIST represents the background features (black features), along with LRP's multiplicative nature, causes the interpreter to not take any of the background features into consideration. In other words, any weight and redistributed relevance value, when multiplied with a zero activation value, will remain zero.

We demonstrate this behaviour in the following section by visually comparing the attribution maps of the same synthetic network, but with different input value normalisations. If true, the 'background' features will become relevant if they correspond to non-zero feature values.

3.5.2 Synthetic model normalisations

Here we visualise the attributions of the same model as described in Section 3.3.2, but with the input features normalised between various ranges. This is done to test whether a normalisation range that includes a zero-value boundary truly limits the number of input features available to the interpreter as seen in Section 3.5.1. To do this, we change the normalisation range of the network from *[0, 1]*, by scaling each input feature to a new normalisation range through using specifically chosen a and b values along with Eq. 3.4. The a and b for each network's normalisation ranges are shown in Table 3.11, along with the values of the performance metrics of these networks.

$$normalised\_value = \frac{feature\_value - a}{b} \tag{3.4}$$

Where *normalised_value* refers to the input feature *feature_value* after being scaled to a new normalisation range from the normalisation range *[0, 1]*.

Using these normalisation ranges, we retrain our synthetic network and generate another

Table 3.11: Different synthetic network performances and normalisation values.

Range:	*a*	*b*	**Epochs:**	**Train acc:**	**Train loss:**	**Valid acc:**	**Valid loss:**
[0, 1]	0	1	1	1.00000	0.00025	1.00000	0.00024
[-1, 1]	0.5	0.5	1	1.00000	0.00013	1.00000	0.00013
[-2, 2]	0.5	0.25	1	1.00000	0.00005	1.00000	0.00005
[-4, 4]	0.5	0.125	1	1.00000	0.00004	1.00000	0.00004
[-1, 0]	1	1	1	1.00000	0.00015	1.00000	0.00015

set of attribution maps for each normalisation range, as shown in Figure 3.6. Changing the input feature range to *[-1, 1]* immediately has a dramatic impact on the interpretations. By minimising the number of zero-value features, the interpreter is capable of attributing relevance without being restricted to specific areas of the sample.

These interpretations, which are generated using most to all input features, can be considered more meaningful than those generated using only specific groups of input features as seen in Figure 3.5. For MNIST-like data specifically, it not only shows which input features relevance is assigned to, but also shows that the absence of foreground features in sections of the background is also important for classifying samples. In general, by not using a normalisation range with a zero-value boundary, the interpretations might become more informative.

Additionally, we performed the experiment on samples normalised between *[-2, 2]* and *[-4, 4]* to see whether the scaling of the input features would affect the network. This is also shown in Figure 3.6. The interpretations of these different normalisations allocated relevance similar to the same locations, but they are not exact replicas of one another. Although there are slight visual differences between the interpretations of these two ranges they are not significant enough to warrant further investigation. The overall interpretations remained similar, with all normalisation ranges containing non-zero boundaries attributing relevance to similar regions for the different classes.

It can also be noted that the interpretations generated using a normalisation range without

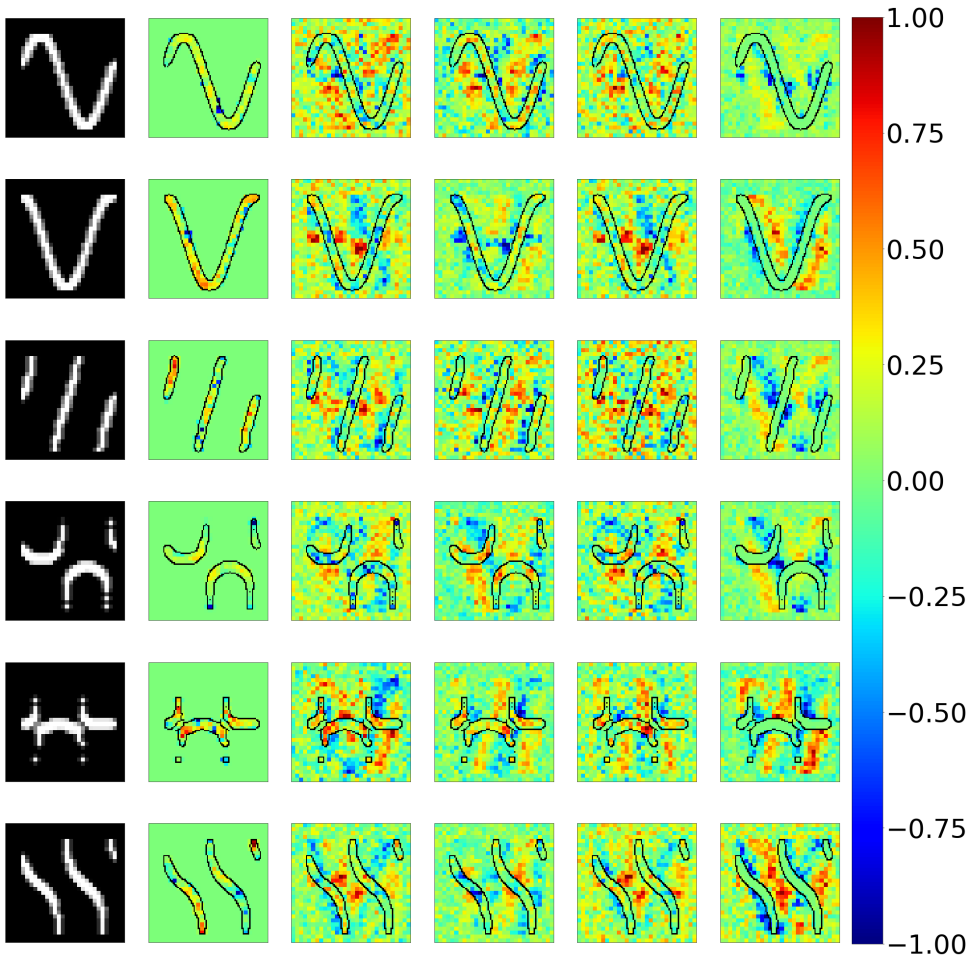


Figure 3.6: Interpretations for our synthetic network for different normalisation ranges. First column shows one sample for each class (rows) before interpretation. Afterwards, from left to right, each column represents the interpretations from the synthetic network for an input feature range of *[0, 1]*, *[-1, 1]*, *[-2, 2]*, *[-4, 4]* and *[-1, 0]*, respectively.

a zero-value boundary are able to highlight features that distinguish classes from one another as per the expectation of the data set stated in Section 3.3.1. For instance, the interpretations of the sine function (row one in Figure 3.6) highlight the lack of features in the bottom left and top right regions, while the interpretations of the cosine function (row two) focus on the lack of features in the middle and in either one of the two side regions. From the results provided it can be seen that the interpreter highlights a set of class-defining features for all samples belonging to that given class.

Lastly, we analyse the interpretations generated for input features normalised between *[-1, 0]*, also shown in Figure 3.6. This has an inverse effect to the *[0, 1]* normalisation range,

and causes the interpreter to restrict its view solely to the background features - completely disregarding any of the important foreground features. Again, although the interpretation is correct, the model itself does not take all input features into consideration. The classifier is only focusing on a specific subset of elements to the detriment of others, causing the interpreter to produce weaker interpretations than those that consider all input features.

Of course, these attribution maps can be highly variable across different samples within a certain class, but due to the nature and simplicity of the MNIST data set and our synthetic data set, the distinction between 'background' and 'foreground' features will remain consistent. Since we have shown that input feature scaling does not affect the interpretations of a network, but normalisation ranges with boundaries of *[0, n]* or *[n, 0]*, where $n \in \mathbb{R}$, do: all experiments following this chapter will be performed with input features normalised between *[-1, 1]*, unless otherwise specified.

3.5.3 Synthetic model interpretation analysis

Validation of the LRP interpreter will not only include visual judgement of the interpretations, but also determining the correctness of its feature relevance attributions [10], [60]. This is done by correlating the effects of removing relevant input features to the subsequent decrease of the classification accuracy.

The attribution maps provided thus far are themselves not enough to verify and ensure the validity of our in-house interpreter. Because of this, we perform a perturbation-based validation technique [4], [10] (henceforth referred to as a relevance test), which provides a clear analytical way to assess the relative accuracy and fidelity of an interpreter. During this test, the top $x\%$ of relevant contributing input features, as determined by the interpreter, will be systematically masked. Masking essentially removes the targeted input features from the network's classification considerations. We do this by replacing the selected input features with zero-values, which are then subsequently viewed as unimportant by the interpreter and are not taken into account during the classification process, as is seen in Section 3.5.2. The reasoning is that if the interpreter is correct in its inter-

pretation, then masking the highest contributing input features will result in the largest decrease in the network's classification accuracy. This would indicate whether the interpreter and its interpretations are accurate and reliable for the data set and model in question.

Following what was done by several different authors [4], [10], [46], we use our interpreter to calculate the relevance of all input features, and average the results for five samples across five seeds for all classes. We then systemically mask the top 14% relevant contributing input features, predicting and storing the target output activation value of the modified sample each time. The number of input features masked is chosen in accordance with what other researchers have done in related work [10], [60], but for the most part this may be chosen arbitrarily.

Figure 3.7 shows the relevance test graphs of different classes for our synthetic network normalised between the values of *[-1, 1]*. As expected, the graph clearly shows a decrease in the target output activation values as features that were deemed relevant are removed. There is a slight decrease in the slope of the curve as less relevant features are removed.

All graphs seem to start at some high positive activation value that monotonically decreases as inputs are masked. This shows that the interpreter successfully attributed relevance to the correct inputs and functions as intended. A positive activation value suggests that the input features contain evidence supporting the model's classification decision, while a negative value indicates contradictory evidence. This can be interpreted as the network becomes more unsure of its classification decision as the activation value decreases. After the activation value becomes negative, the network no longer supports the specified class as a possible classification decision, except in the scenario where all other class options produce an even more negative activation value.

3.5.4 MNIST model attribution maps

Figure 3.8 and Table 3.12 show the interpretations and performance values of our trained MNIST networks for different normalisation ranges boundaries. Similar to the synthetic

network, the results clearly show that the use of a normalisation range with a zero maximum or minimum value, restricts the interpretation of the network to either the fore-

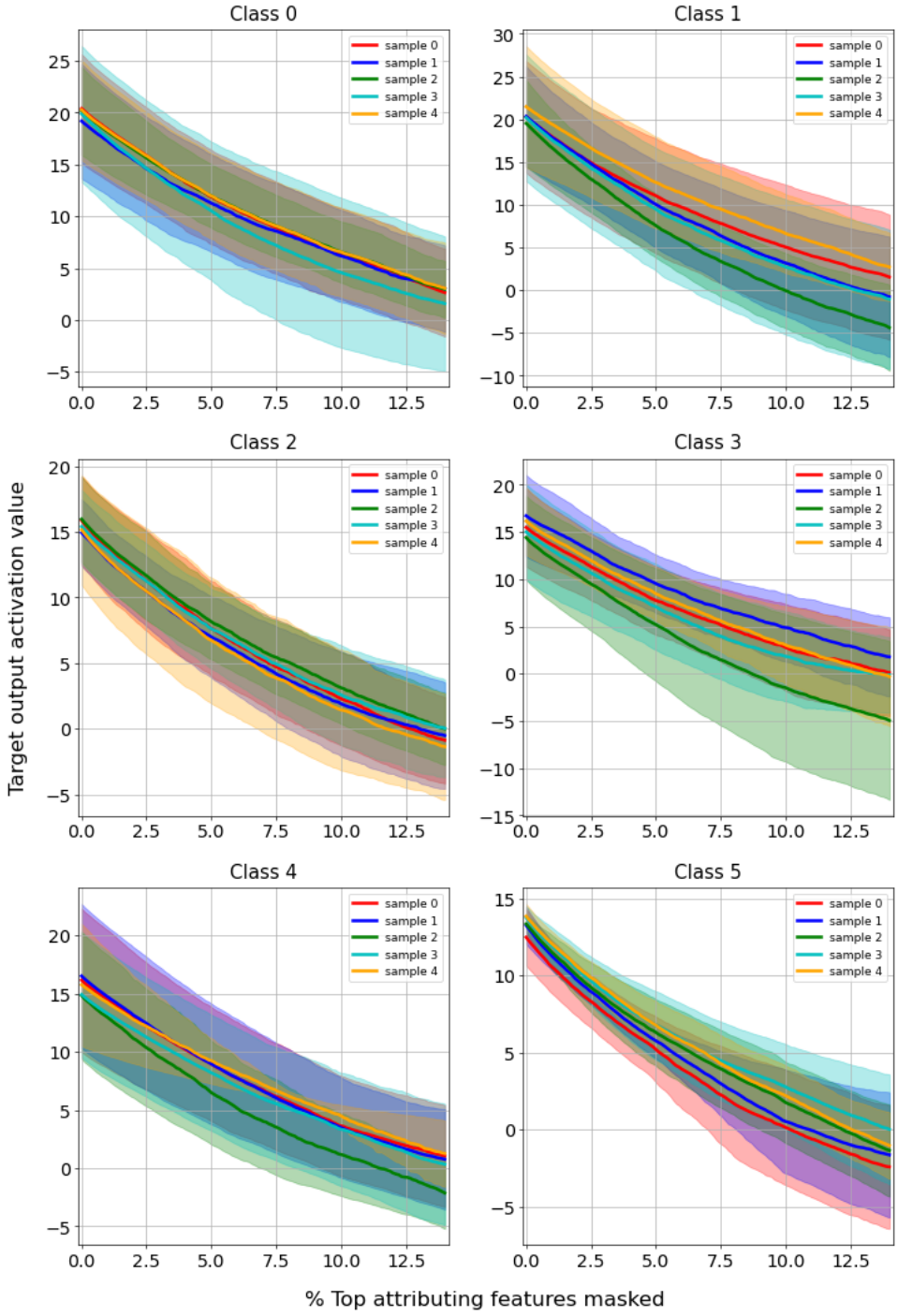


Figure 3.7: Relevance test performed on the synthetic network with a normalisation range of *[-1, 1]*. Shows the target output activation values of different classes as more input features become masked. Each quadrant shows the results for five different samples averaged over five seeds for different classes.

ground or background features, depending on whether a zero value is used for the lower or upper boundary. Also, all networks with normalisation ranges that use non-zero intervals attribute relevance to similar regions in the attribution maps, regardless of the interval scale. This means that input feature scaling does not affect the interpretations of the networks trained on MNIST.

Considering the results shown in Figures 3.6 and 3.8, we can make the following statements concerning the MNIST data set and our synthetic data set. A normalisation range containing a zero-valued maximum or minimum can be viewed as an artefact value, which restricts the view of the interpreter to either:

1. Foreground features when a zero-valued minimum is used; and
2. Background features when a zero-valued maximum is used.

Image data sets such as ImageNet [82] or Canadian Institute For Advanced Research (CIFAR)-10 [83], [84] that do not have this clear distinction between 'foreground' and 'background' features in their samples, may experience the same issues, but to a lesser degree, as the samples should have less zero-value input features.

Table 3.12: Different MNIST network performances and normalisation values. We omit the evaluation performance values, as we are only interested in the validation performance values and if the network achieved interpolation (100% training accuracy).

Range:	μ	σ	**Epochs:**	**Train acc:**	**Train loss:**	**Valid acc:**	**Valid loss:**
[0, 1]	0	1	91	1.00000	0.00003	0.98100	0.19161
[-1, 1]	0.5	0.5	81	1.00000	0.00017	0.98400	0.11806
[-2, 2]	0.5	0.25	80	1.00000	0.00005	0.98400	0.15910
[-4, 4]	0.5	0.125	81	1.00000	0.00012	0.98260	0.12716
[-1, 0]	1	1	167	1.00000	0.00011	0.98020	0.14643

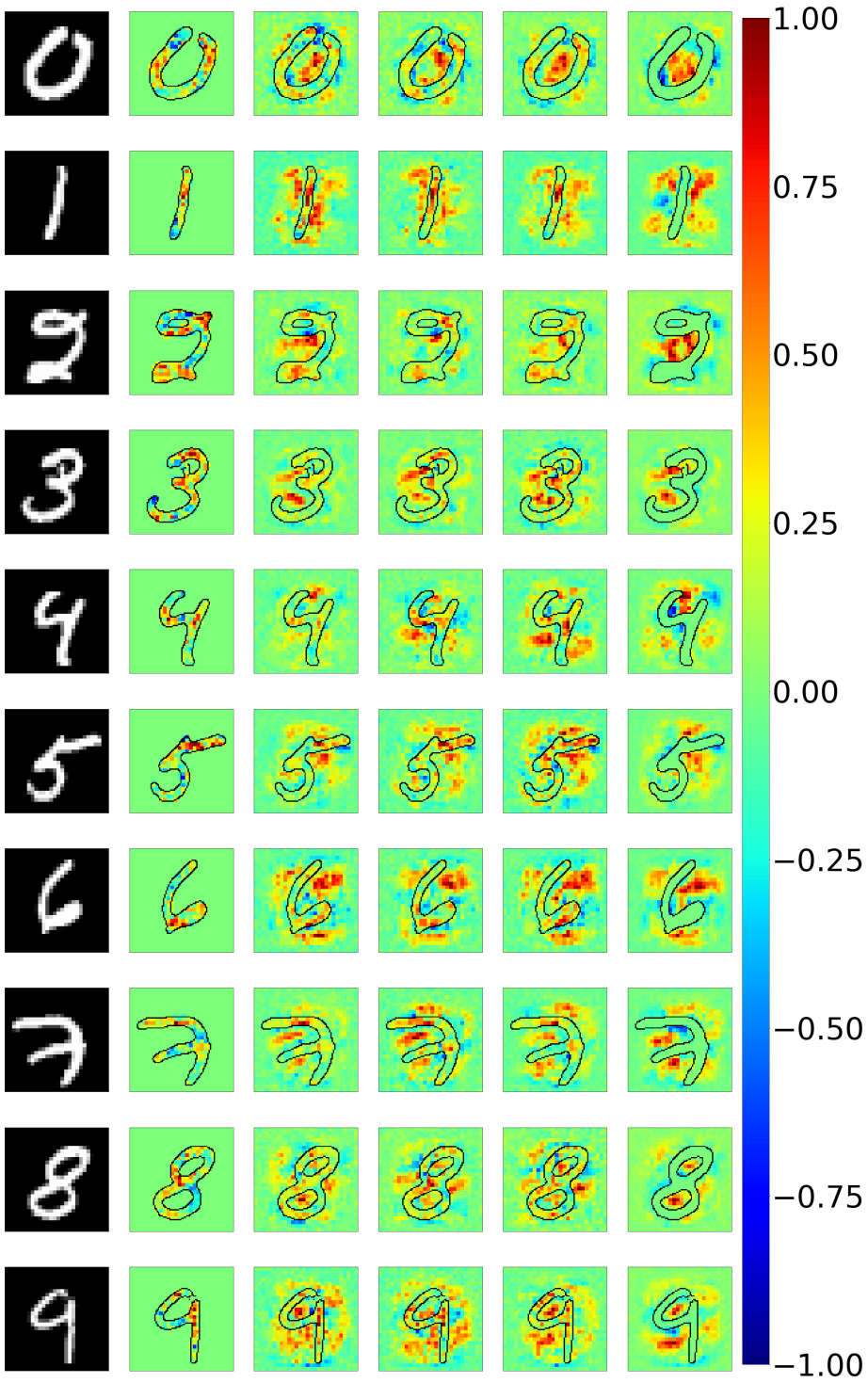


Figure 3.8: Interpretations on our MNIST network for different normalisation ranges. First column shows one sample for each class (rows) before interpretation. Afterwards, from left to right, each column represents the interpretations from the synthetic network for an input feature range of *[0, 1]*, *[-1, 1]*, *[-2, 2]*, *[-4, 4]* and *[-1, 0]*.

3.5.5 MNIST model interpretation analysis

Here we perform a relevance test as described in Section 3.4.2. Similar to our synthetic network, we mask the top 14% relevant contributing input features for five samples averaged over five seeds for four different classes in descending order. Figure 3.9 shows the results of the test as performed on the MNIST network with a normalisation range of *[-1, 1]*. Also, similar to the results from our synthetic network in Figure 3.7, the target output activation value of each class decreases as more inputs are masked.

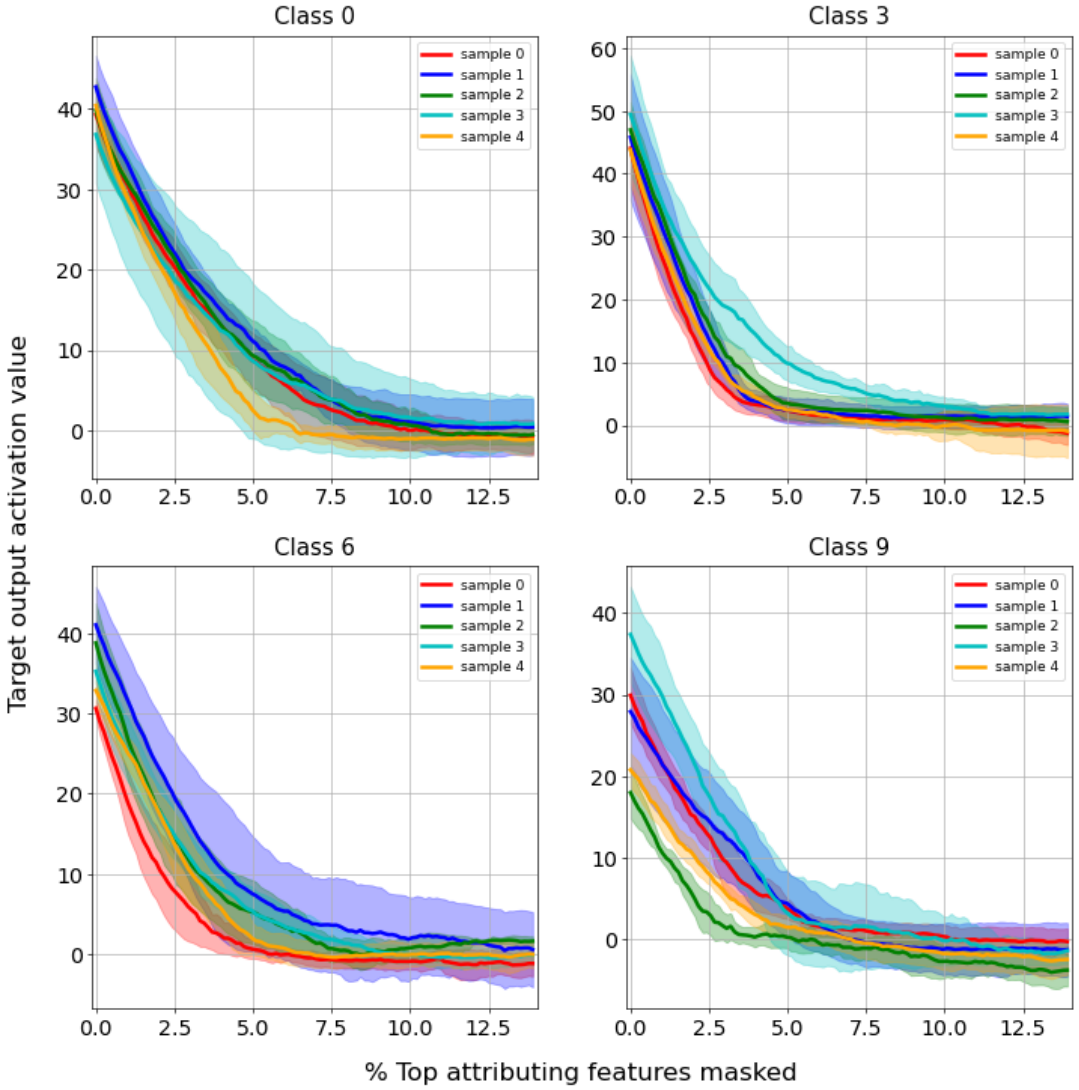


Figure 3.9: Activation values for the different classes of the MNIST network normalised between the values of *[-1, 1]* as the top 14% relevant contributing input features are masked. Each quadrant shows the results for five different samples averaged over five seeds for different classes.

The curve of the graphs is much more prominent than in Figure 3.7, which should be noted. A reason for this could be because MNIST contains a few generally very relevant features and many irrelevant features, resulting in a steeper downward curve as the most important input features are removed. Since the input features are masked in descending order of their relevance, the rate at which the target output activation values drops lessens significantly as the masked input features become less relevant to the overall classification. This results in the target output activation eventually asymptoting after enough input features have been masked. These results show that the interpretations of our interpreter on the MNIST network successfully pinpoints the important features.

3.6 Conclusion

In this chapter we primarily described the experimental environment setup for this study. We stated that our development environment consists of a Python Integrated Development Environment (IDE) working in conjunction with the PyTorch framework. A description of the different data sets used throughout the upcoming chapters was provided along with the definitions of the different networks we intend to use and gave an overview of the LRP interpreter setup. We provided reasons as to why we opted to construct our own LRP codebase, rather than just using an already established pre-existing baseline. Afterwards, we verified the implementation of our LRP codebase and continued to evaluate its efficacy and correctness by analysing its interpretations on our defined networks.

In addition, we discovered an unexpected fault in the implementation of the LRP-$\alpha\beta$ rule in the widely-used Captum.ai Python package which caused the interpreter to incorrectly attribute negative relevancies to the input features for a specific pair of α and β values. It was also demonstrated that feature normalisation ranges have a disproportionate effect on feature relevance in the presence of strong background or foreground features.

Chapter 4

Node sample sets

Initial investigations regarding the inner workings of our trained MNIST network using sample sets analysis and the Jaccard similarity index.

4.1 Introduction

In Chapter 3, we described the LRP interpretation technique, as well as our implementation of an in-house LRP interpreter. Additionally, we justified the use of our own in-house interpreter by comparing it to the performance of other available implementations. We also laid the groundwork for the use of this tool by showing the attribution maps it generates in a simplified context.

In this chapter we first develop an intuition surrounding the properties and behaviours of sample sets [12], [13], [15] within a DNN and secondly determine whether the sample set clusters formed at the different nodes can additionally cluster the interpretations of the set's samples in meaningful ways. The experiments described in the following sections will show the results from the MNIST network, as described in Section 3.3.3, with a

normalisation range of *[-1, 1]*, the performance values of which are provided in Table 3.12.

We use the works of Davel et al. [12] as a starting point for investigating sample sets, by replicating the behaviour observed in their networks, with regard to the way sample set sizes change from shallower to deeper layers, in Section 4.2. Secondly, in Section 4.3, we calculate the number of active nodes per hidden layer for each class. This is based on another experiment conducted by the same authors [12]. After these initial experiments, we investigate the distribution of class-specific samples between nodes in Section 4.4 using the Jaccard similarity index.

4.2 Sample set distribution

In this section we replicate the results found by Davel et al. [12], which show that the decision-making processes of nodes tend to become more class-specific towards the deeper layers of a network. This provides a starting point for future experiments, and will also help us to gain a better intuition of how sample sets function within a network.

In Figure 4.1, for each class separately, we show the percentage of samples in the training, validation and evaluation sets for which each node is activated. Classes are differentiated by colour, while nodes are arranged, by layer, from left to right according to their distance from the input layer. As a reminder to the reader: here class-specific sample sets refer to the set of samples for a given class that activate a specific node within the network.

The results also seem to indicate a trend where the class-specific sample set size for all nodes either significantly increases or decreases towards the deeper layers of the network. Nodes in the deeper layers of the network seem to activate for close to all or none of any specific class's samples. This is indicative of the idea in LRP (as discussed in Section 2.6.2) that class information becomes more entwined within deeper layers [61]. A similar trend was also observed in [12].

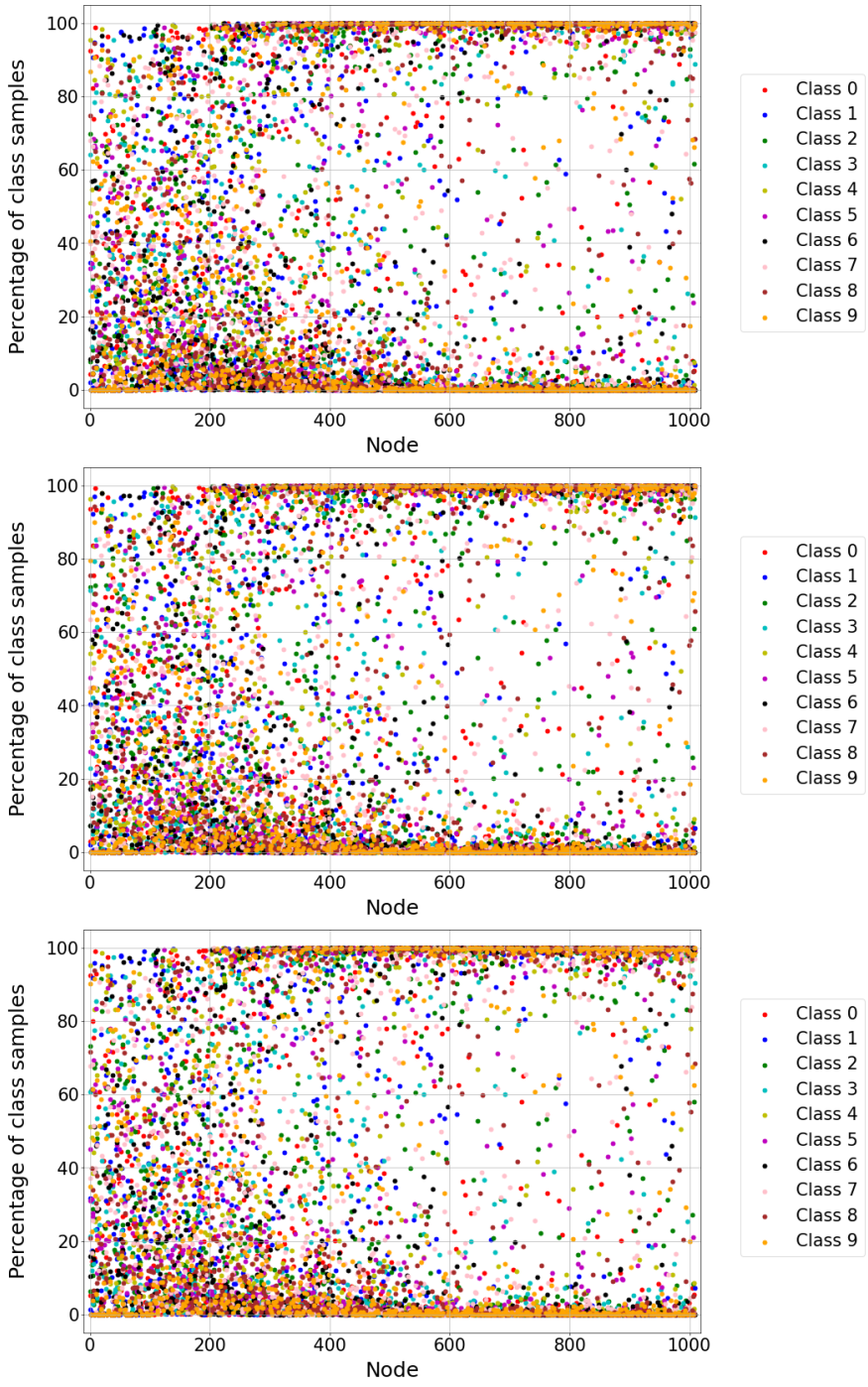


Figure 4.1: Percentage of class samples activated per node for every class. From top to bottom, the figures show the results for the training, validation and evaluation sets respectively. Nodes are arranged according to layer, where nodes $0 - 99$ represent the first layer, nodes $100 - 199$ represent the second layer, etc. Other seeds in Appendix A, Figures A.1 to A.3.

4.3 Investigating active nodes

Davel et al. [12] used the *binary* pattern of node activations (referred to as an activation pattern) as a sample representation. Since different samples are able to share the same activation pattern, they were able to provide a rough estimate of the number of activation patterns necessary to model all samples for a particular class. Their results showed a decrease in the number of required activation patterns in the deeper layers of a network.

Here we further investigate sample set distributions using a similar idea. Rather than using activation patterns, we determine whether this behaviour is reflected by the number of active nodes per hidden class. Because the activation pattern is based on the number of active nodes, by simply analysing the active nodes themselves, we might gain a better understanding of how samples interact within a network.

The number of active nodes per layer for each class of the training, validation and evaluation sets is shown in Figure 4.2, where it can be seen that, similar to the number of activation patterns in [12], it does indeed steadily decrease toward the deeper layers of the network. This correlates well with the changes in sample set size as seen in Figure 4.1. This could be an indication that most of the information processing happens in the first few layers, whereafter fewer nodes are necessary for the classification process.

Oddly enough, the network starts with fewer active nodes than expected before the number of active nodes steeply increases in the second layer. This observation is not investigated further as part of the current study.

4.4 Jaccard similarity analyses

In this section we investigate how samples are distributed among sample sets by analysing the similarity of class-specific sample sets formed at the different active nodes. These experiments were performed across several classes in order to ensure repeatability. Although only the MNIST class zero is shown in the main text for the sake of brevity, similar trends

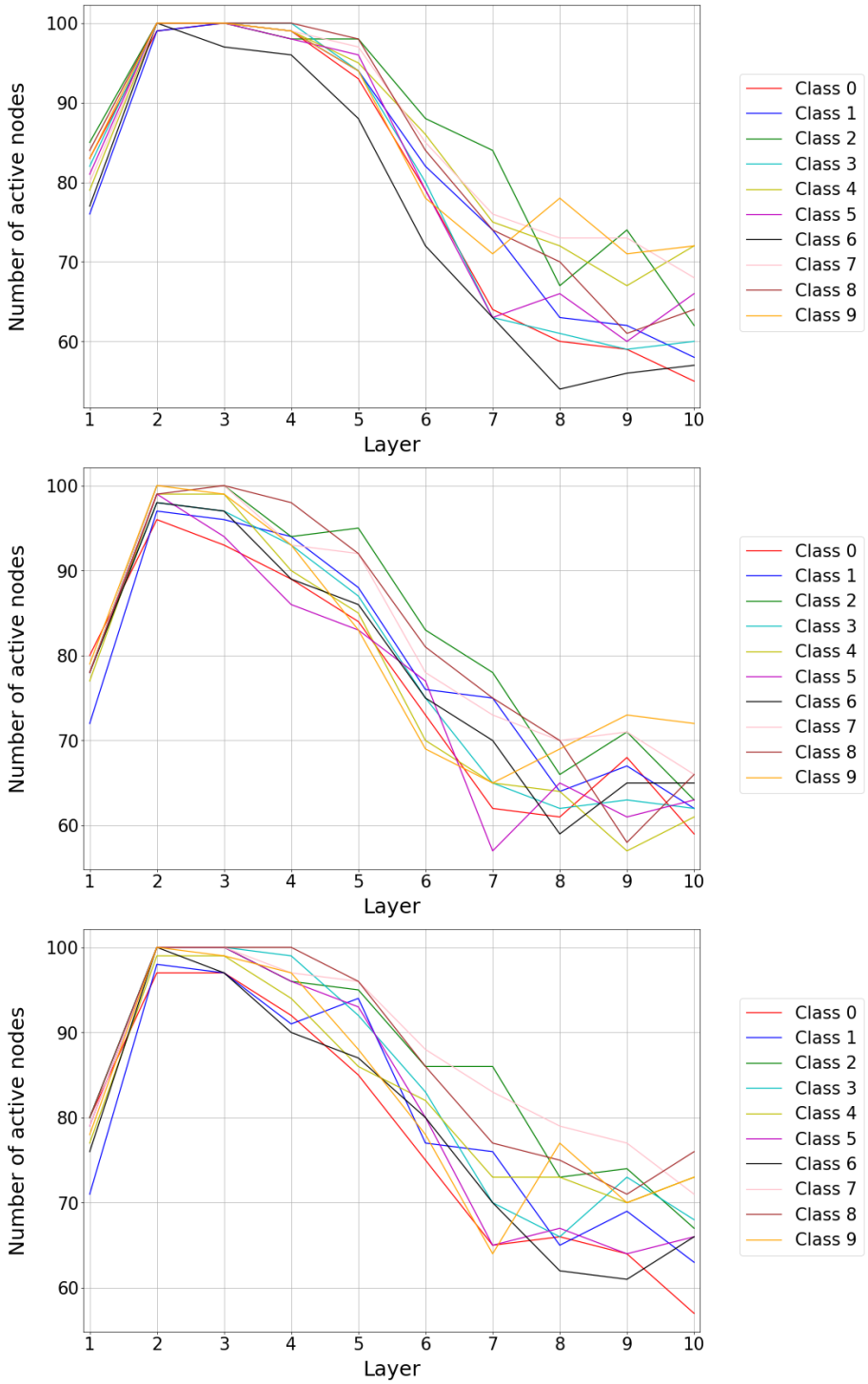


Figure 4.2: Number of active nodes per hidden layer for every class. From top to bottom, the figures show the results for the training, validation and evaluation sets respectively. Other seeds in Appendix A, Figures A.4 to A.6.

were observed across all classes. (Additional results are included in the appendices, as indicated.)

We use the Jaccard similarity index [85], [86] to measure the similarity between two sets of values. When given two sets, the Jaccard similarity index is defined in Eq. 4.1 as the size of the intersection (the set of overlapping elements) of the two sets, divided by the size of the union (the set of all elements) of the two sets. As the number of overlapping elements between the two sets increases, their Jaccard similarity also increases. Also, note that when using two sets of different sizes, the size of the intersection can only become as large as the the size of the smaller of the two sets.

$$J(A, B) = \frac{|A \cap B|}{|A \cup B|} \tag{4.1}$$

This similarity index is used in the following experiments to investigate how samples are distributed among sample sets, by analysing the similarity of class-specific sample sets formed at the different active nodes.

First, we determine the similarity between active nodes of the same layer, per layer of the network, in Section 4.4.1. Secondly, in Section 4.4.2, we determine the similarity between all active nodes of the network, where all active nodes are ordered by sample set size in ascending order. Lastly, in Section 4.4.3, we determine the similarity between all active nodes of the network, where all nodes are first ordered by their respective layers, and are then ordered by sample set size in ascending order.

4.4.1 Node similarity per layer

In this experiment we determine the Jaccard similarity index of sample sets between active nodes in the same layer. As shown in Figure 4.3, we plot a similarity matrix for each layer of the network based on the Jaccard similarity index, where the active nodes for each layer are ordered by sample set size in ascending order. As expected, the similarity between active nodes is proportional to the sizes of their sample sets, with the largest similarities being found between active nodes with large sample set sizes. The exception to this rule is the node pairs around the $x = y$ axis that always show at least some similarity, especially

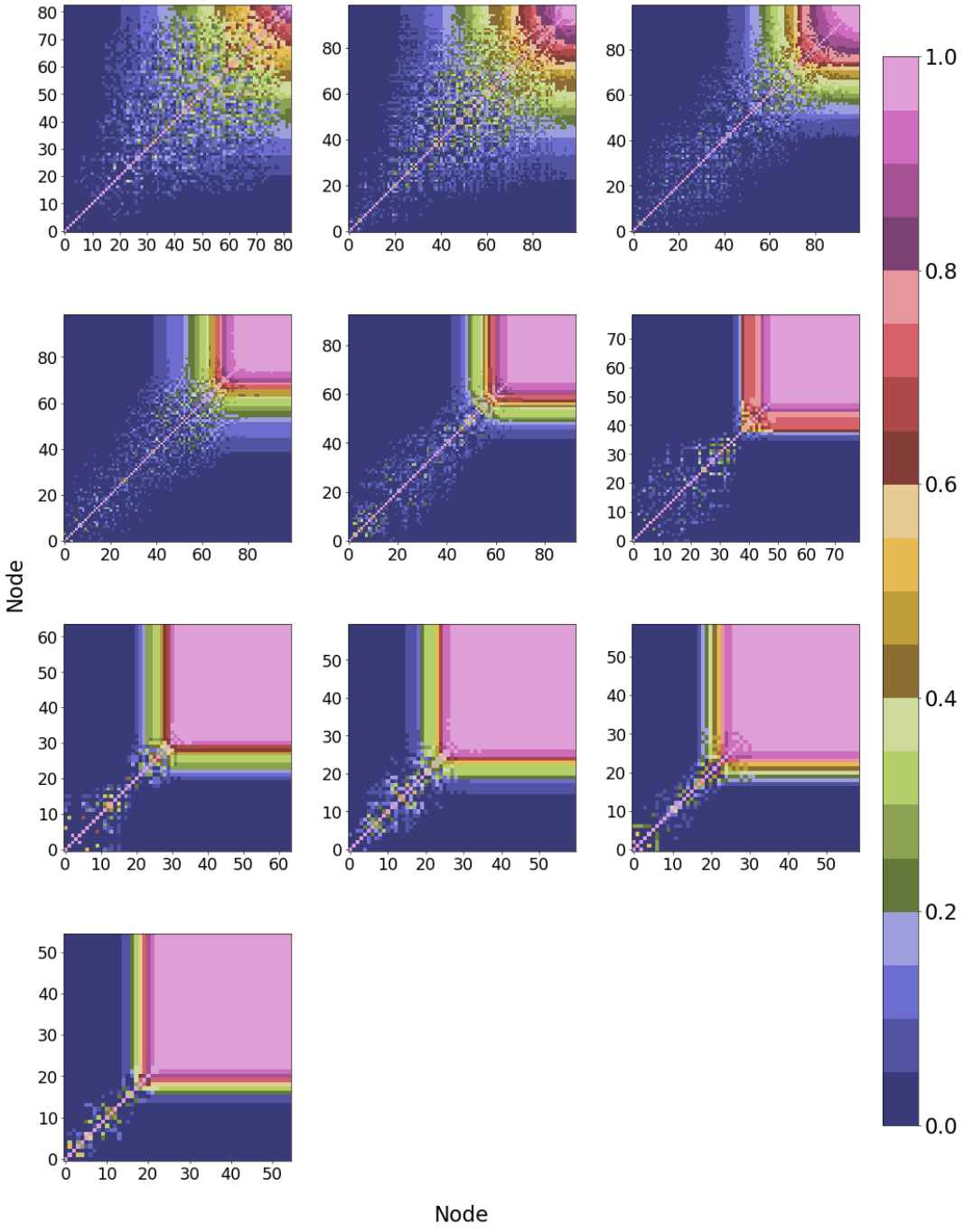


Figure 4.3: The Jaccard similarity index between every pair of active nodes for all layers MNIST class zero, where all nodes are sorted in ascending order with regard to their sample set size. Each matrix represents a layer in the network from shallow to deep and is read from left to right, top to bottom. Note that each matrix is mirrored on the $x = y$ axis. Other classes in Appendix A, Figures A.7 to A.9.

in the deeper layers. Keep in mind that these results are influenced by the Jaccard index's sensitivity to the sample set size of the nodes [87]. We also noticed that the number of

active nodes with large similarities increases with each subsequent layer.

Interestingly, the behaviour observed here in Figure 4.3 seems to resemble that of Figure 4.1. The graduality with which the nodes of each layer transition from having little similarity to being nearly identical becomes very abrupt in the deeper layers of the network. We also know from Figure 4.1 that the nodes with the largest sample set sizes activate for practically all samples of a given class, meaning that, not only are there nodes that activate for almost every sample of a provided class, but that the number of these nodes increases with each passing layer and inevitably becomes the majority node type in each layer.

4.4.2 Node similarity across layers

In this experiment we determine the Jaccard similarity index between all active nodes in the network, where all nodes are ordered by sample set size in ascending order. As shown in Figure 4.4 we make a scatter-plot for the sample sets of every active node pair. Each dot is coloured according to the Jaccard index between the two nodes.

Figure 4.4 supports the results found in Section 4.4.1, by also showing that the Jaccard similarity index increases as the sample set sizes of both nodes increase. However, just like in Section 4.4.1, these results could also be due to the Jaccard index's dependency on the sample set size of the nodes. Regardless of this, Figure 4.4 clearly shows that the nodes with the highest sample set similarity are generally the ones with the largest sample set sizes. As we see in Figure 4.1, these sizes equate to practically all samples for the given class.

4.4.3 Node similarity per and across layers

For the final experiment in this section, we determine the Jaccard similarity index between all active nodes in the network, where all nodes are ordered first by layer and secondly by sample set size in ascending order. Similarly to Section 4.4.1, we generate a similarity

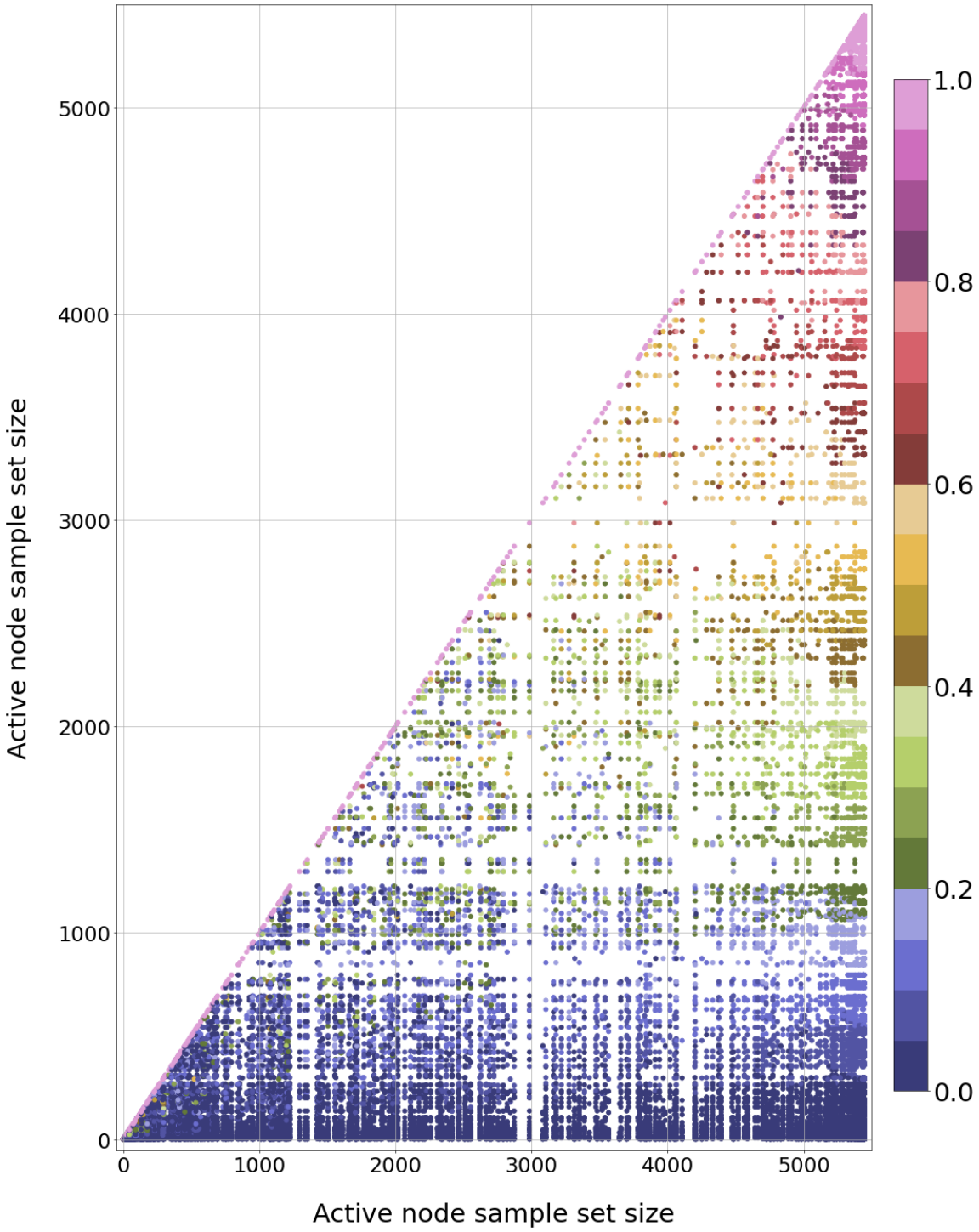


Figure 4.4: The Jaccard similarity index of every active node pair for MNIST class zero. Node pairs are sorted in ascending order according to the size of their respective sample sets. Other classes in Appendix A, Figures A.10 to A.12

matrix, as shown in Figure 4.5, but with the above-mentioned ordering of active node pairs. Alongside the similarity matrix, Figure 4.6 shows the sample set size of every

active node, where the nodes have the same ordering as in Figure 4.5. This way, we can compare the measurement of every node at their respective layer with every other node in the network, according to their sample set size.

Similar to the experiments in Sections 4.4.1 and 4.4.2, these results could be due to the Jaccard index's dependency on the sample set size of the nodes. From the sample set sizes we can see that with every subsequent layer, the disparity between nodes with very large or very small sample set sizes increases substantially. When investigating Figure

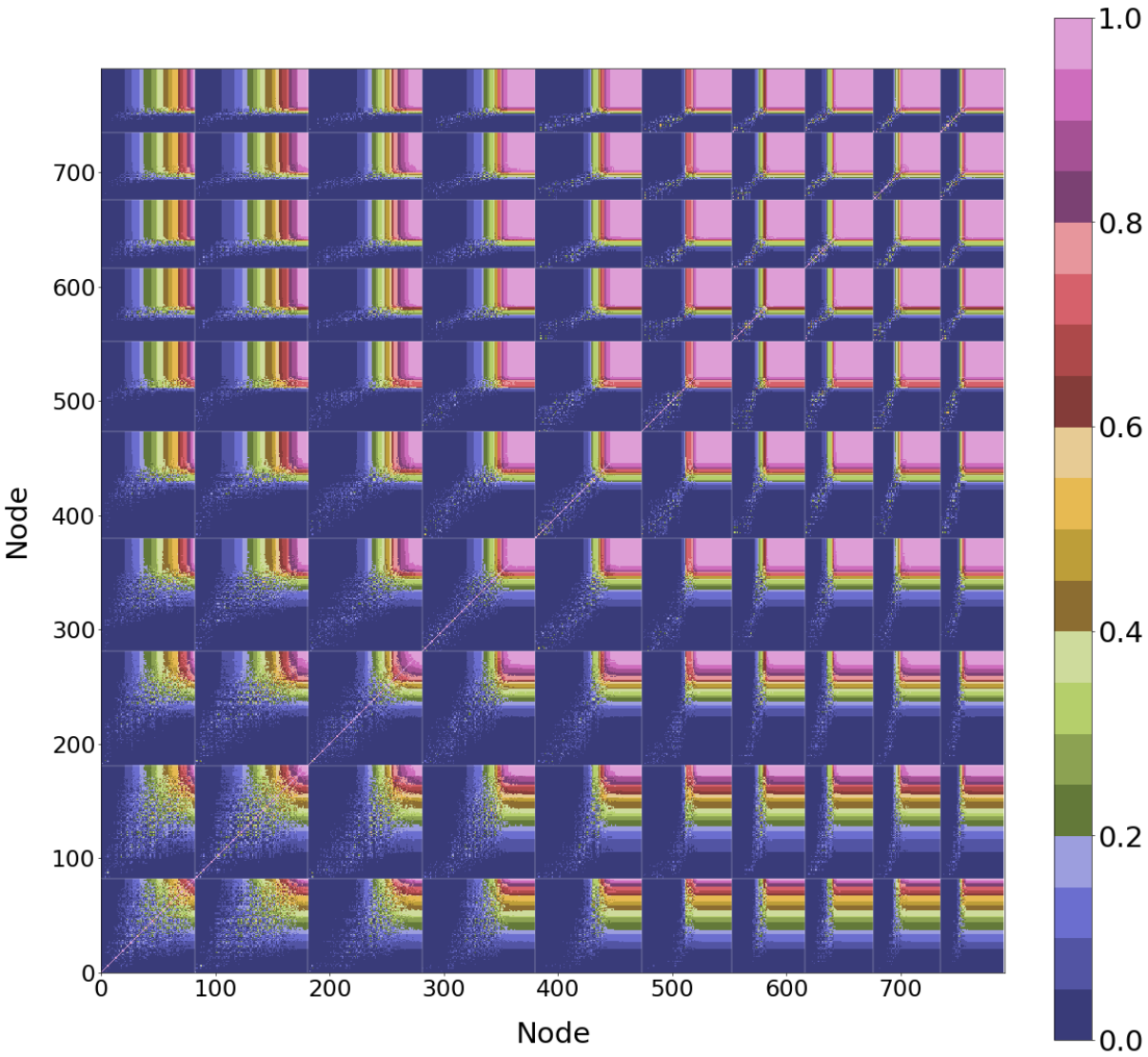


Figure 4.5: The Jaccard similarity matrix of every possible node pair in the network for MNIST class zero. Nodes are ordered first according to their layer (first to last) and secondly according to their sample set size (small to large) for each layer. Other classes in Appendix A, Figures A.13, A.15 and A.17.

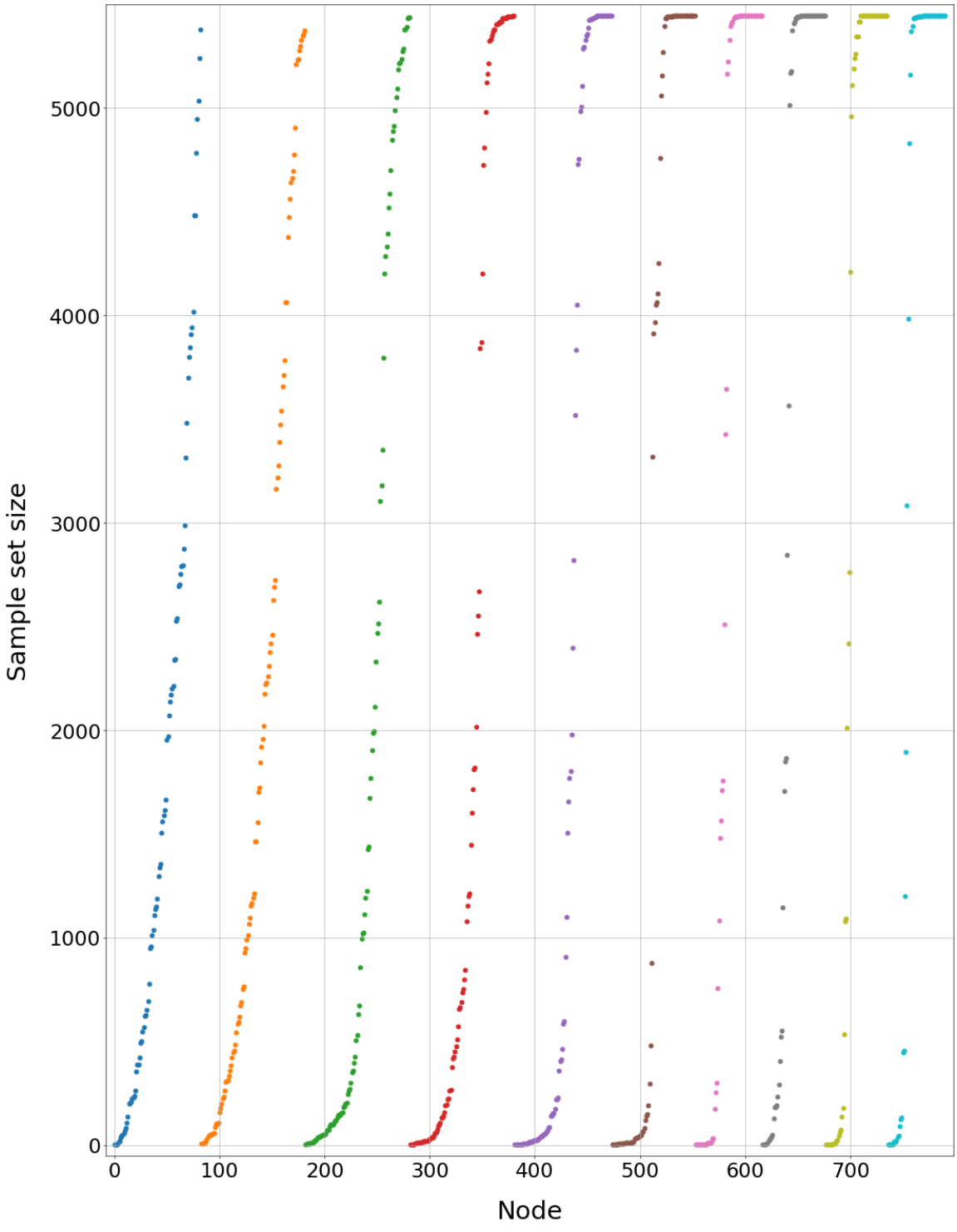


Figure 4.6: The sample set size of every node in the network for MNIST class zero. Nodes are ordered in the same way as in Figure 4.5. Other classes in Appendix A, Figures A.14, A.16 and A.16.

4.5, high similarity regions seem to correspond well, with both layers having nodes with large sample sets. The inverse is true for low similarity regions. These results provide

similar observations to those in Section 4.3 and 4.4.

4.4.4 Discussion

From the experiments conducted in Section 4.4, we primarily saw that 1) high Jaccard similarity indices are almost exclusive to nodes with large sample sets, and 2) these large sample sets encompass practically all the samples for a given class. Individually, these nodes with large sample sets do not offer additional, meaningful information when it comes to grouping the interpretations of the network, but could do so when they are analysed in conjunction with one another.

Utilising sample set analysis of multiple nodes, in order to identify which nodes activate for the majority of samples, in conjunction with LRP's ability to untwine the class information as represented by nodes deeper in the network (see Section 2.6.2) could be advantageous when interpreting a DNN. This may allow us to identify and interpret the subcomponents (nodes or combinations thereof) that are most fundamentally responsible for a model's classification decisions. We further explore this in Chapter 5.

4.5 Conclusion

In this chapter we explored whether sample set analysis can group sample interpretations through examining the inner workings of our MNIST network. We replicated the results of Davel et al. [12] to use as a starting point for our exploration, by first examining the network's percentage sample set size per node and secondly by analysing the number of active nodes per hidden layer. Afterwards, we investigated the Jaccard similarity index between active nodes for different scenarios with regard to their class-specific sample sets. From the results given in Section 4.4, we made the following observations:

- The similarity between active nodes increases in proportion to their respective sample set sizes.

- The number of samples contained within active nodes with large sample set sizes tends to be close to the number of samples in that specific class.
- The number of active nodes with large sample set sizes significantly increases within the deeper layers of the network.
- The gap between active nodes with very large and very small sample sets increases with every subsequent layer.

These observations seem to indicate that the class-specific sample sets created by active nodes are not that useful with regard to interpretability when studied individually. Rather, it suggests that it would be more beneficial to investigate the sample sets created through combinations of nodes working in conjunction with one another. This premise forms the basis of Chapter 5, where we will show that samples can be grouped according to which nodes they activate in a given layer.

Chapter 5

Encoding sample sets

We introduce and describe the concept of encoding sample sets, core nodes, and variation nodes, while investigating their functionalities within our neural network.

5.1 Introduction

In Chapter 4 we explored the idea of sample sets. A sample set is associated with a specific node in the network and consists of all the samples in the training set that activates that specific node. Therefore, the emphasis is very much on the sets that activate single nodes.

In this chapter, instead of investigating the activation of a single node, the emphasis is now on the activation of the nodes in a single layer of the DNN. In a single layer, a single sample from the training set will activate some of the nodes, but not others. These patterns can be encoded as binary vectors, and all the samples that activate a given pattern form an encoding sample set. Throughout the experiments in this chapter, we analyse encoding sample sets within the context of neural networks and devise a technique to make use of them when interpreting DNNs. Specifically, we build upon the work of

the previous chapter by introducing the concept of encoding sample sets as a novel way to group samples based on the way they are represented internally by a DNN.

In Section 5.2 we first define the concept of encoding sample sets, along with additional new terminology. In Section 5.3, we examine the number of encodings per layer, individual encoding sample set sizes, and their relationships with one another, in order to determine whether encoding sample sets can be used to group sample interpretations in meaningful ways. In Section 5.4 we use LRP in conjunction with encoding sample sets to develop the concepts of *core* and *variation nodes*, the concepts which we further use to explore the inner workings of the trained models. Lastly, in Section 5.5, we explain our approach, and demonstrate that it can enhance the capabilities of interpretability techniques such as LRP.

5.2 Terminology

Here we define the novel terms that will be used to investigate and explain the results obtained throughout this chapter.

5.2.1 Encoding sample sets

A *sample encoding* describes the binary pattern of nodes that did, and did not, activate for a given sample at a certain layer. We provide a visual aid to help describe encodings in Figure 5.1. As mentioned in Section 2.6.2, each activated node can be seen as a single feature of the current layer that is formed by combining a number of features from the previous layer that are connected to that active node [15], [16]. Thus, an active node in a layer could be seen as a consolidated set of preceding features that are, in some fashion, relevant to the provided sample. If the sample encodings at a specific layer differ between two samples, we can assume that the samples have at least some differing important features.

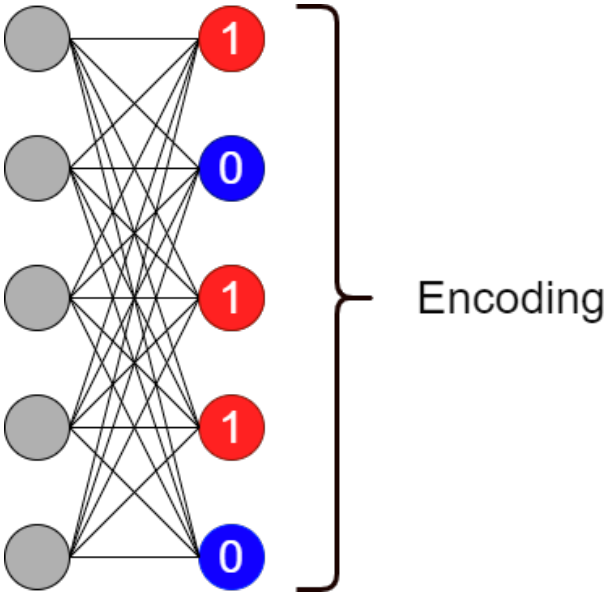


Figure 5.1: Example of a sample encoding within an arbitrary layer of a network. Encodings are identified according to their binary pattern of node activations, thus we refer to this encoding as a 10110 encoding. Red nodes (1) indicate active nodes, while blue nodes (0) indicate inactive nodes.

Building off the previous chapters and on the work done by Davel et al. [12], [13], [15], we propose that sample sets may be defined in one of two ways, namely:

- ***Node sample sets:*** The definition of sample sets as described in Section 2.4, that is, the set of samples a given node activates for.
- ***Encoding sample sets:*** A set of samples that produces an identical corresponding encoding at a specific layer.

5.2.2 Core and variation nodes

With regard to encoding sample sets, we define the term *core nodes* as the nodes in a certain layer that activate for all class-specific sample encodings in that layer. Here class-specific sample encodings simply refer to the sample encodings activated by the samples belonging to a specific class. Any node that only activates for a subset of class-specific sample encodings in a layer, and not for all encodings, is referred to as a *variation node*.

5.3 Encoding analysis

In this section, we perform our initial encoding sample set analyses on the MNIST network in a similar manner to Section 4.4.

We begin by determining the number of unique sample encodings per layer for every class in Section 5.3.1. Afterwards, we investigate the cardinality of the determined encoding sample sets, and analyse the similarity of different encoding sample sets in Sections 5.3.2 and 5.3.3, respectively. Lastly, we analyse the model's unique encodings in Section 5.3.4, while examining their similarities in Section 5.3.5.

In order to lessen the computational workload that comes from analysing the large number of encodings in the the first few layers of the network, we only investigate the layers after the number of encoding sample sets begins to decrease significantly. From Figure 5.2, we can ascertain that this transition occurs around layer five.

5.3.1 Unique encodings per layer

We begin our analysis of encoding sample sets by investigating the number of unique sample encodings per hidden layer for every class in the training set. Figure 5.2 shows that the number of unique encodings is close to, or equal to, the number of samples in each class, in the shallower layers of the network. After the first few hidden layers, however, the number of unique encodings experiences a sharp decrease, leaving only a few 100 encoding sample sets per class in the last hidden layers. These results follow the same pattern as the investigation of the number of active nodes per hidden layer for node sample sets as initially explored in Section 4.3.

5.3.2 Encoding sample set size

In Section 5.3.1, we showed that similarly to node sample sets, the number of unique encodings per layer drastically decreases at some point through the network's layers. We

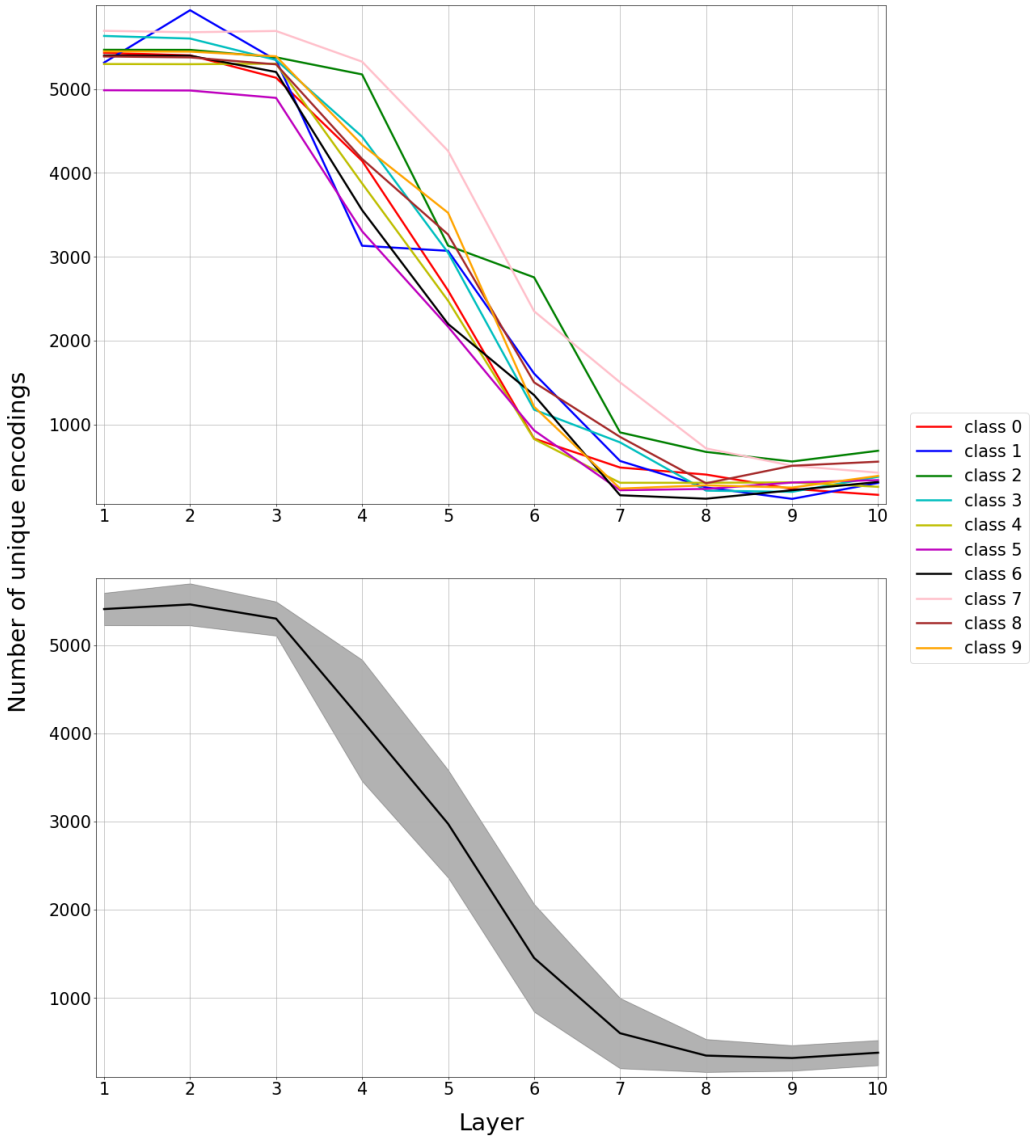


Figure 5.2: Number of unique encodings per hidden layer for every class (top) and averaged over all classes (bottom) with the shaded regions indicating the standard deviation across classes. Other seeds in Appendix B, Figures B.1 to B.3.

investigate whether encoding sample sets will continue to mimic the behaviour of node sample sets by measuring their size for the deeper layers of the network over several classes. This is shown in Figure 5.3, where the colour of the dots denotes the following:

- ***Red:*** Encoding sample set with a size of 1.
- ***Green:*** Encoding sample set with a size between 1 and 100.
- ***Blue:*** Encoding sample set with a size greater than 100.

Note that the difference in size between the largest and smallest blue encoding sample sets is much larger than the same difference for the encoding sample sets marked in either green or red (where all sets are of size one). Table 5.1 provides the actual number of encoding sample sets represented by the red, blue and green dots.

As shown by Figure 5.3 and Table 5.1, the number of small encoding sample sets (marked in red and green) decrease in the deeper layers of the network, and the number of large encoding sample sets (marked in blue) increase in the deeper layers of the network. Note that, per layer, the largest few encoding sample sets contain more samples than all other encoding sample sets combined. This indicates that the majority of samples within a class

Table 5.1: Encoding sample set sizes for the different encoding sample set groups as shown in Figure 5.3.

Layer	**Num. of red dots**	**Num. of green dots**	**Num. of blue dots**
		Class 0	
5	2013	582	2
6	528	295	7
7	245	228	12
8	188	199	14
9	123	97	10
10	78	72	9
		Class 3	
5	2340	708	0
6	793	376	6
7	463	316	6
8	108	95	6
9	97	90	8
10	181	180	13

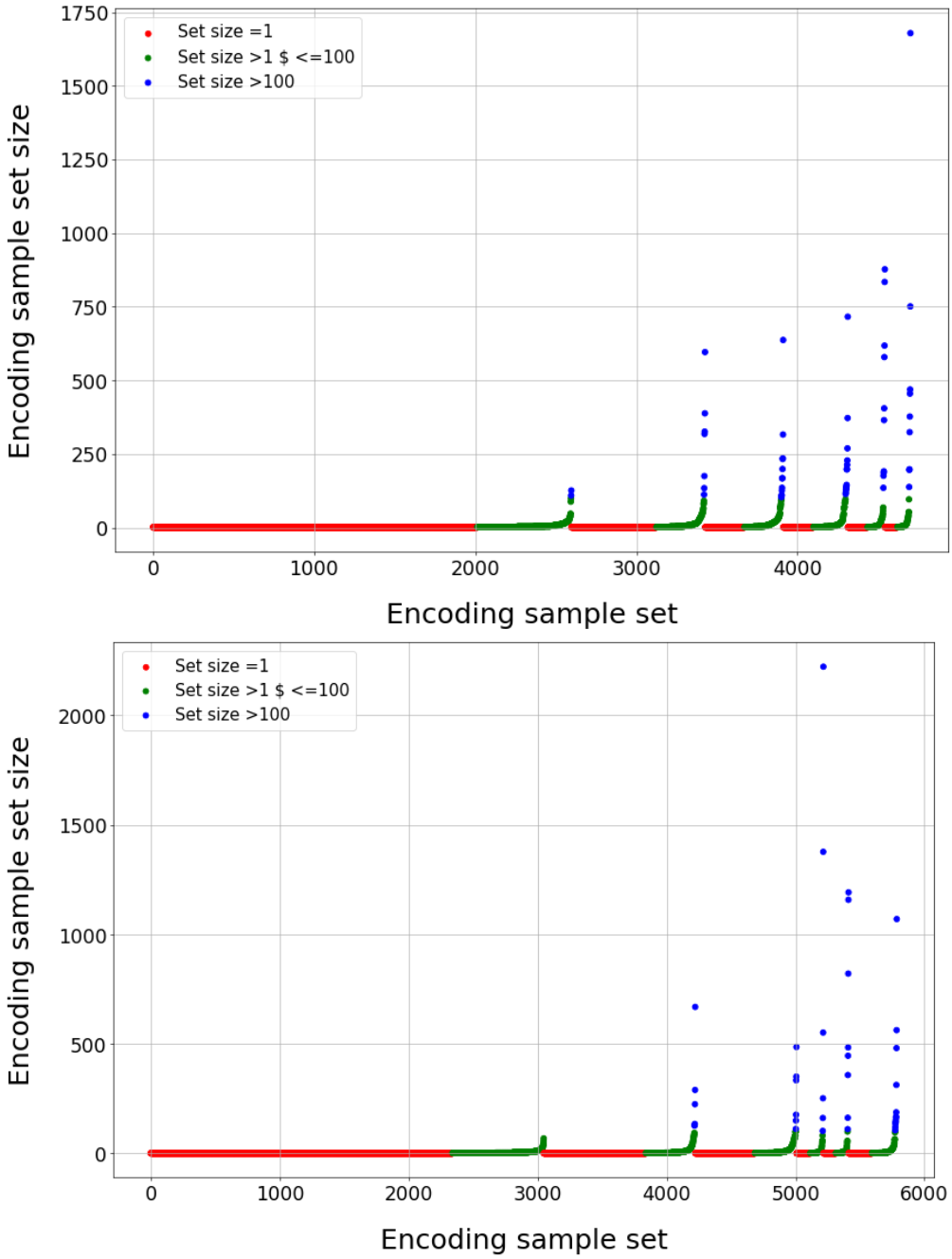


Figure 5.3: Encoding sample set sizes for the MNIST classes zero (top-image) and three (bottom-image), between the hidden layers five to ten. Encoding sample sets are first ordered by layer and then by sample set size in ascending order. Encoding sample sets are colour coded as follows: red indicates a size of one, green indicates a size of greater than one but less than or equal to 100 and blue indicates a size of greater than 100. Other classes in Appendix B, Figure B.4.

can be found within a few of the largest encoding sample sets, while all other encoding sample sets contain much smaller groups of samples, often in singletons.

5.3.3 Encoding sample set similarity

Here we analyse the degree of sample overlap between encoding sample sets of different layers for MNIST class zero. This is an alternative version of the analysis conducted in Section 4.4.3, but here we focus on encoding sample sets rather than on node sample sets. We similarly use the Jaccard similarity index to determine the level of overlap between the samples of different encoding sample sets across different layers for a single class.

Figure 5.4 shows these Jaccard similarity indices for every encoding sample set pair in the network from layers five to ten, with all encoding sample sets ordered by layer, and then by set size. These results show the similarity between encoding sample sets substantially decreases as their sample set sizes increase, with the exception of a few encoding sample sets as shown on the bottom-most figure. This is opposite to the results seen in Section 4.4, where we saw on multiple occasions that the similarity between node sample sets increases in proportion to their sample set sizes. The only exception to this, are typically very few of the largest encoding sample sets as seen on the bottom graph of Figure 5.4.

Note that when comparing Figure 5.3 and 5.4, a significant portion of the initial encoding sample sets for every layer contains only one sample, meaning they will either fully correlate to a sample encoding in another layer, or not at all. This is the cause of the purple diagonal lines we see in Figure 5.4. It indicates singleton encoding sample sets in one layer that correspond to singleton encoding sample sets in another layer. This might suggest that some samples are regarded as 'different' enough from any other to require a more unique representation throughout the model. Similar trends were seen with other classes.

Given a layer, it is impossible for a sample to exist within multiple encoding sample sets at the same time, as a sample cannot have multiple binary node activation patterns in the same layer. Therefore, when comparing encoding sample sets with regard to their samples, the similarity score of an encoding sample set will be either 0% when compared against other encoding sample sets of the same layer, or 100% when compared against itself. Because of this, we do not compare the encoding sample sets of a single layer with

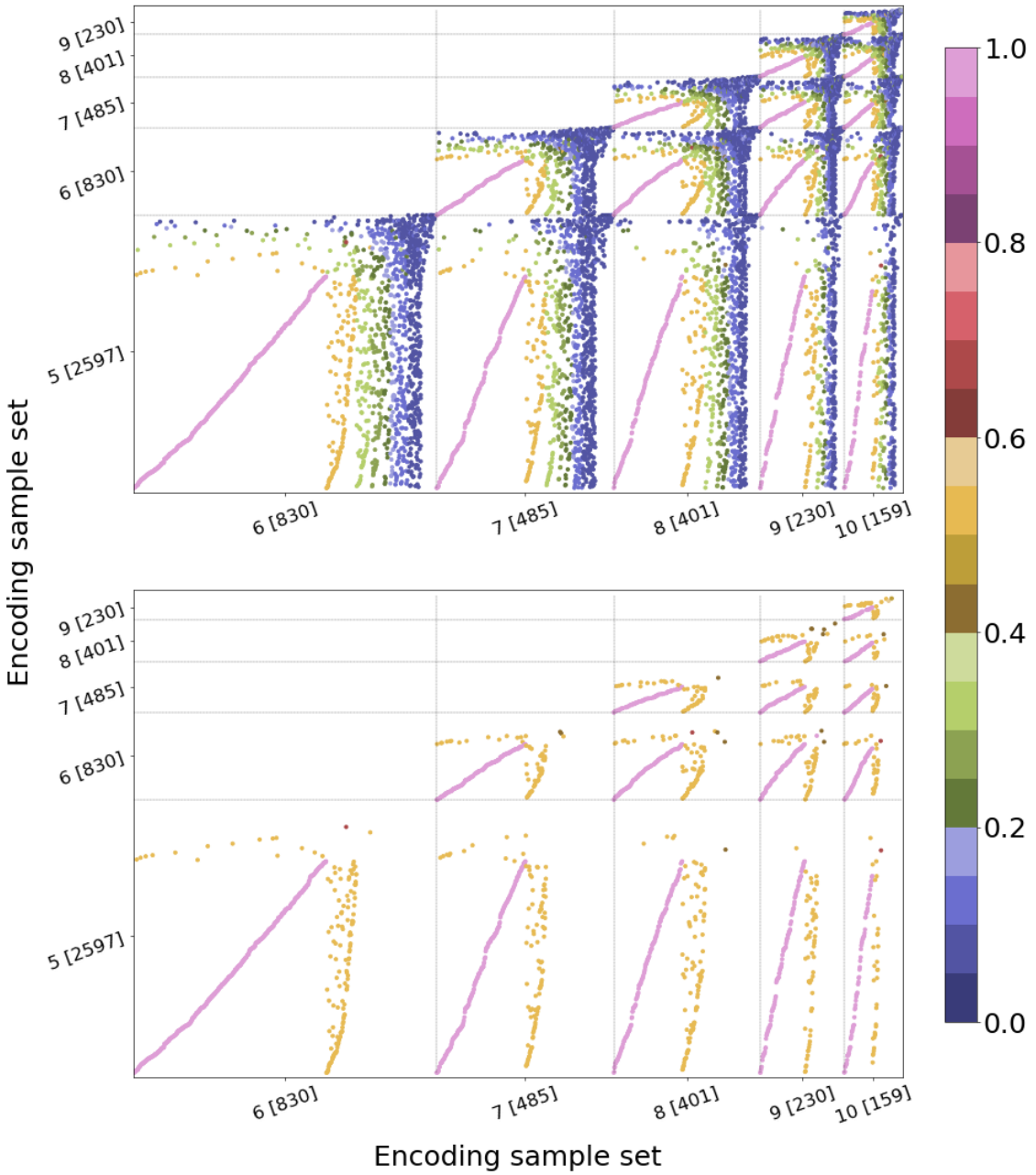


Figure 5.4: Jaccard similarity between every pair of encoding sample sets in the deeper layers of the network. The plots show the same graph, but only for similarities higher than 0.05 (top) and 0.6 (bottom) respectively. Each tick represents the layer and the number of encoding sample sets it contains in brackets, while the black lines are used to separate layers. Encoding sample sets are first ordered by layer and then by sample set size, both in ascending order. Other classes in Appendix B, Figures B.5 to B.7.

one another in this fashion.

5.3.4 Encoding analyses

Here we investigate the sample encodings for all unique encoding sample sets in hidden layers zero, five and ten, for the MNIST classes zero and three, as shown in Figure 5.5. The active nodes for every encoding sample set are coloured according to size of that encoding sample set. The prescribed colouring is shown by the colour bar. Blue is used to indicate the inactive nodes for a given encoding sample set, while the super-imposed green highlights the nodes that are active for all encoding sample sets (core nodes). Nodes are ordered by the largest number of encoding activations, and encodings are ordered by their sample set size from large to small.

Figure 5.5 shows a clear increase in core nodes (highlighted in green) as we progress to the deeper layers of the network. This increase does, however, gradually abate and the number of core nodes stabilises, and even falls of slightly, in the deeper layers of the network, as shown by Figure B.8, in Appendix B. It also gives an indication of encoding sample set behaviour in the first hidden layer of the network, which wasn't explored in the initial experiments detailed in Section 5.3.3, as was motivated in Section 5.3.1.

The increase in these core nodes could be explained as follows: Active nodes in a certain layer base the information they learn on the features of active nodes in preceding layers. This allows the information represented by the active nodes in a network to become more abstract and intertwined as they continuously learn from the nodes in previous layers. Therefore, more information can be represented using fewer nodes, which subsequently results in encoding sample sets containing fewer active nodes that are exclusive to themselves and sharing more active nodes among each other on average as we progress through the network.

5.3.5 Encoding comparisons

Having observed core and variation nodes in Section 5.3.4, we now analyse the similarity of encoding sample sets with regard to their overlapping nodes.

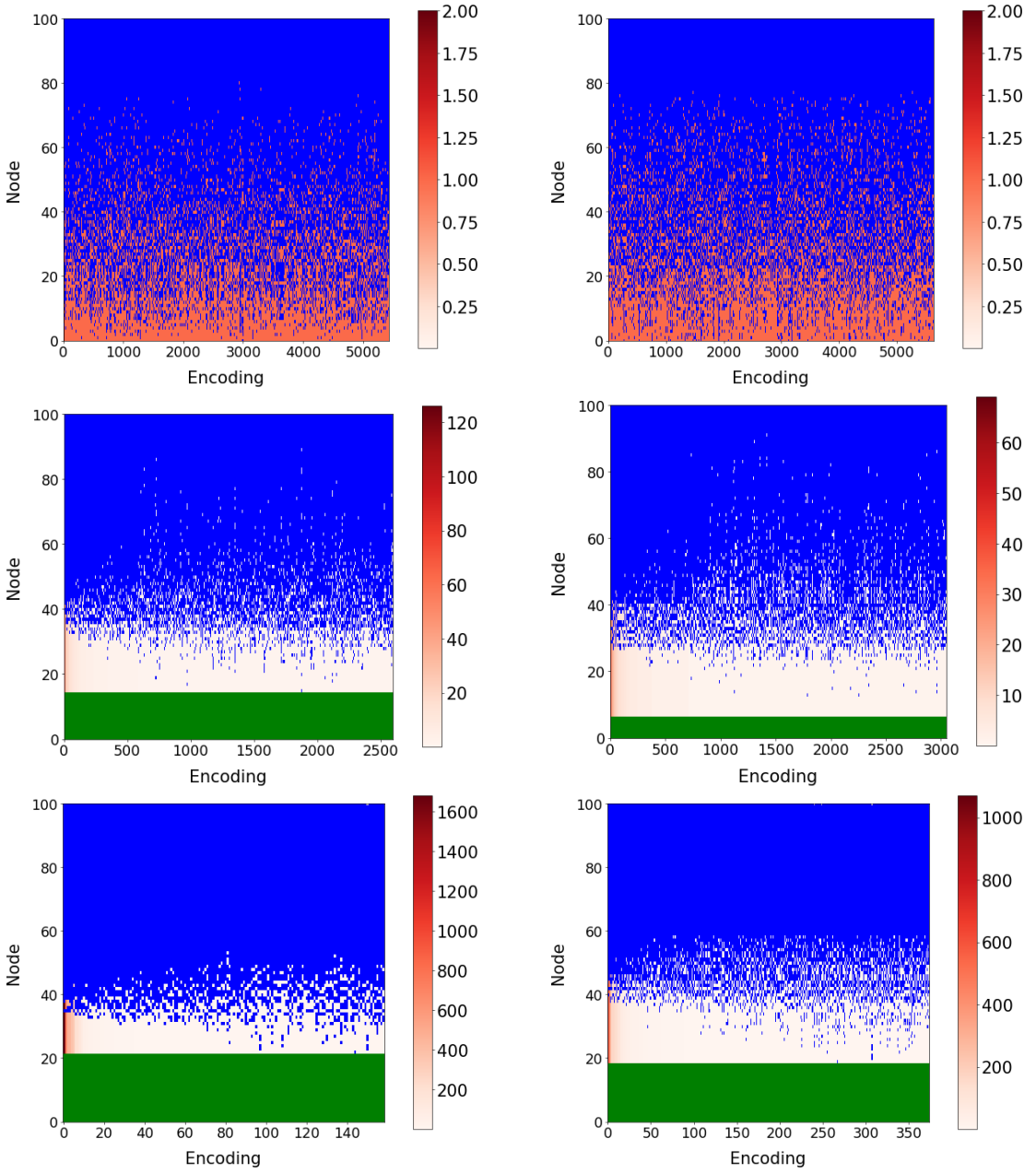


Figure 5.5: Visualisation of the sample encodings for every encoding sample set. The results are shown for MNIST class zero (left) and class three (right). From top to bottom the results are given for hidden layers one, five and ten respectively. Nodes are ordered according to the largest number of encoding activations and encodings are ordered according to their sample set size from large to small. Positive node activations are coloured according to the size of the encoding sample set, while no activations are blue, and core nodes are shown in green.

Using the Jaccard similarity index, we generate similarity matrices to compare the sample encodings for each encoding sample set pair in layer seven, for MNIST classes zero, three

and six, where the set sizes are greater than 100. The resulting similarity matrices are shown in Figure 5.6. The core and variation nodes were identified, and are shown for classes zero, three and six, along with the size of each encoding sample set in Table 5.2. Similar to our previous experiments, we use layer seven as it most clearly shows our results.

The similarity matrix highlights large similarities between every encoding sample set pair. This is further shown in Table 5.2, which indicates the presence of a large number of core nodes with very few variation nodes for each encoding sample set. These results seem to suggest that the class contains a set of specific features that are shared by samples across all encoding sample sets. The features specific to each encoding sample set are then, presumably, the variation nodes used to distinguish samples from different encodings.

5.3.6 Discussion

Similarly to the the results of Section 4.3, Figure 4.2 shows how the number of unique encoding sample sets decrease with each subsequent layer. We further showed in Figures 5.4 and 5.3 that the similarities between encoding sample sets decrease in proportion to their set size, and that the majority of samples are contained within a few of the largest encoding sample sets.

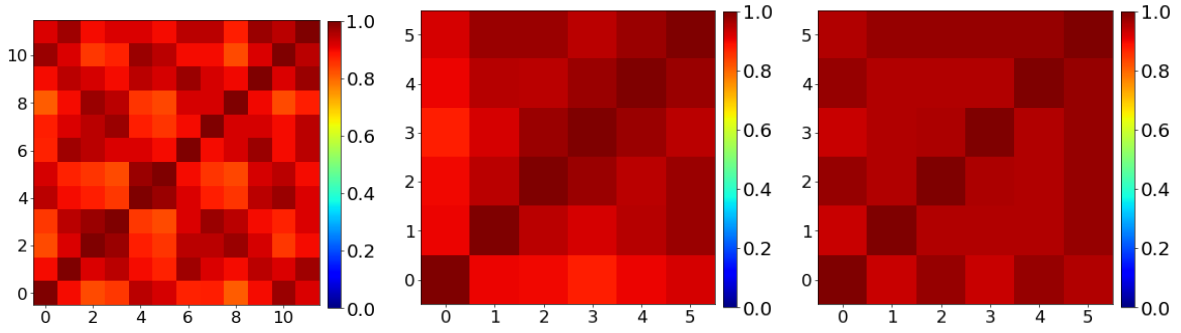


Figure 5.6: The Jaccard similarity index of the active nodes between each encoding pair for all class zero (left), three (middle) and six (right) encoding sample sets in layer seven of the MNIST network, where all encoding sample sets have a size greater than 100. Other layers in Appendix B, Figures B.9 to B.11.

Table 5.2: The number of variation nodes and sample set sizes for all encoding sample sets for MNIST classes zero, three, and six, with a set size greater than 100, and the number of core nodes per class. This table represents the results for layer seven of the MNIST network.

	Set size			**No. of variation nodes**		
Classes	**0**	**3**	**6**	**0**	**3**	**6**
	103	112	120	3	3	2
	109	150	148	1	3	1
	110	177	347	4	1	3
	126	334	448	3	2	3
	135	351	544	2	3	1
	167	486	3158	4	2	2
	168			2		
	199			4		
	233			5		
	235			3		
	316			2		
	637			2		

Class	**0**	**3**	**6**
No. of core nodes	34	36	43

Figure 5.5 showed how the number of core nodes increase with each subsequent layer. By analysing the encoding sample sets in a given layer more closely, we found that these core nodes make up the majority of active nodes in that layer as shown in Table 5.2.

These observations could indicate that the large and highly similar node sample sets from Section 4.4 can be combined to form only one or two encoding sample sets that encompass most, if not all, samples of a class. Because a node can be seen as a single feature of the

layer it is contained in, these encoding sample sets could then theoretically be viewed as a combination of the features that best describe the majority of class samples. Smaller encoding sample sets would then contain the features that allow some samples to exist within the same class, but differentiate them enough from the samples contained within other encodings that they require additional modelling.

5.4 Investigating encodings with LRP

From Section 5.3.1, we saw that the number of unique layer encodings decreases significantly in the deeper layers of the network. More importantly, we saw from Section 5.3.2 that the majority of the encoding sample sets only contain a small subset of the total class samples, while the majority of all class samples are captured by only a handful of encoding sample sets.

In the following sections, we investigate the differences between the sample groupings of different encoding sample sets in the same layer for the MNIST network. We explore the visual differences between encoding sample sets via LRP interpretations in Section 5.4.1. In Section 5.4.2, we analyse and compare the model's core and variation nodes. Note that similar trends were noticed among other classes besides the ones provided in the following investigations. If not provided in a particular section, the results for other classes may be found in Appendix B.

5.4.1 Interpreting encoding sample sets with LRP

The goal of this section is to compare LRP interpretations of samples from different encoding sample sets in the same layer. We do to this by interpreting the averaged input features over five randomly selected samples belonging to the same encoding sample set in a specified layer, for all encoding sample sets with a size greater than 100. The results are displayed in Figure 5.7, with the averaged input features on the left and their respective LRP interpretations on the right. We specifically display the interpretations of layer seven

as it best illustrates our findings.

Subjectively, it appears that the model focuses on largely the same feature regions for all these encoding sample sets, while also displaying slight differences between the interpretations of the different encodings from the same class. This indicates that encoding sample sets do indeed group samples with similar properties and major features, while separating them based on small differentiating features.

This strengthens the argument made in Section 5.3.3 that the largest encoding sample set contains the features that are most prominently associated with a particular class. All smaller encoding sample sets then indicate more sporadically occurring samples with more obscure or deviating features. This is visually motivated in our investigation of core and variation nodes in the following section.

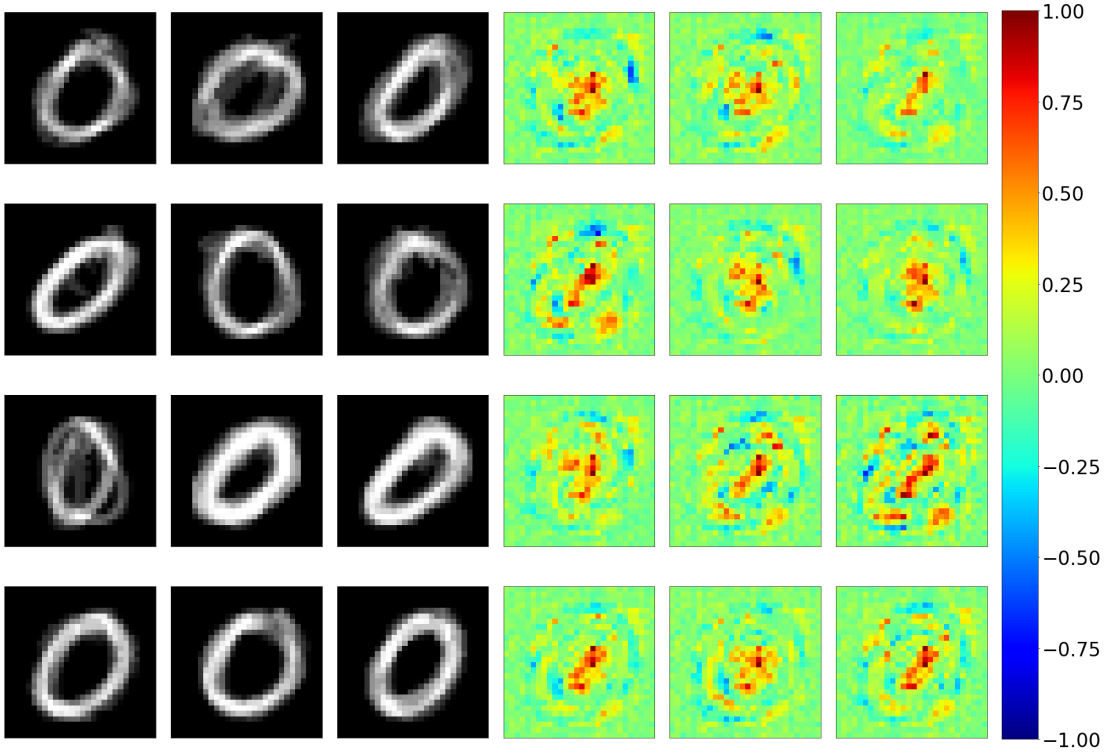


Figure 5.7: Visual depiction of the averaged input features (left) and their respective LRP interpretations (right) over five samples for all encoding sample sets in layer seven with a sample set size of greater than 100. Other classes in Appendix B, Figures B.15 to B.22.

5.4.2 Core and variation nodes

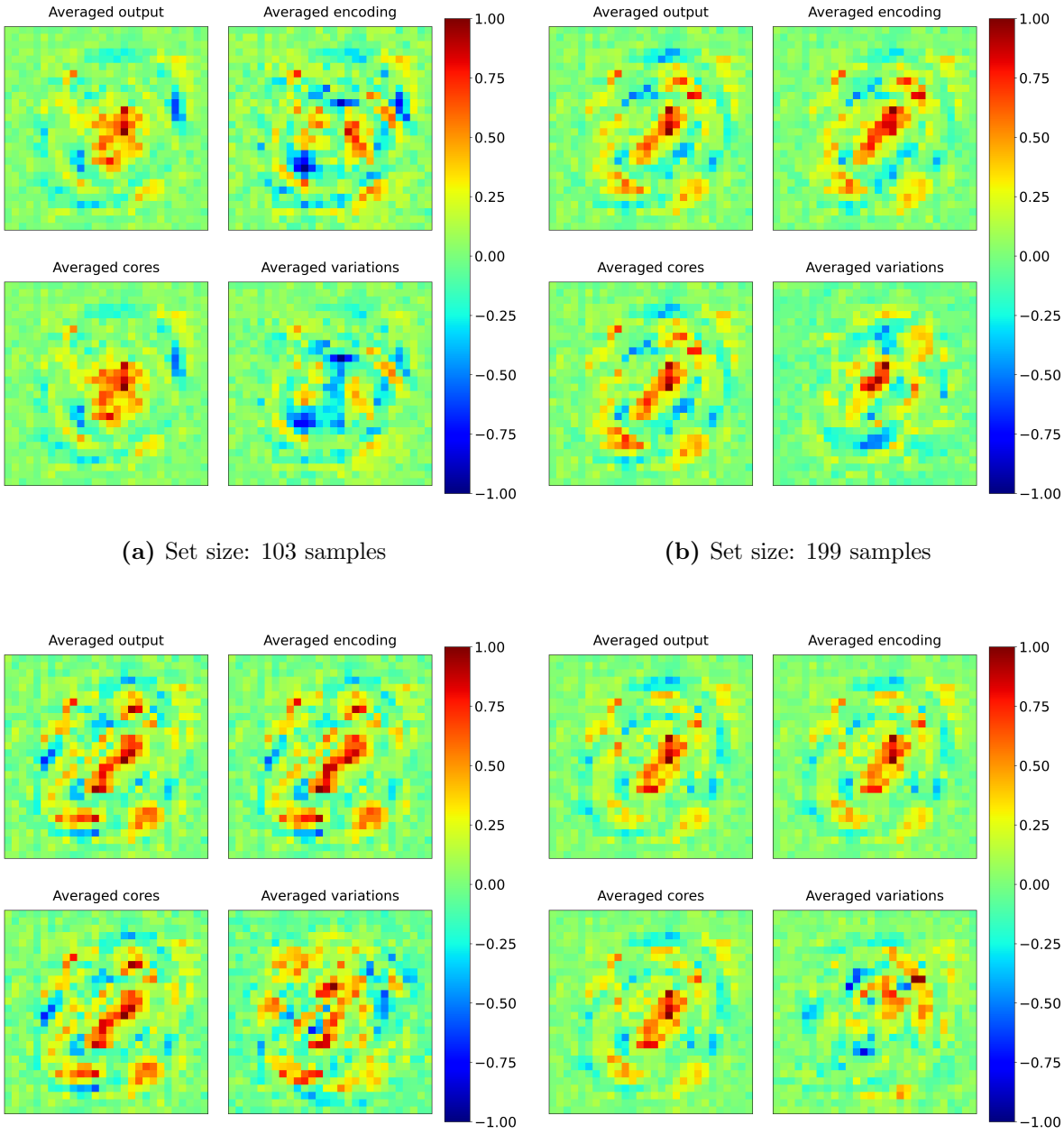
Given the results from Sections 5.3.3, 5.3.4 and 5.4.1, we hypothesise that the class-related information in a model can largely be defined by a core set of nodes in each layer. Variation nodes are then used to represent any non class-specific features that differentiate groups of samples within a certain class from one another. Our definition of core and variation nodes is found in Section 5.2.

Here we further explore the above-mentioned hypothesis by comparing LRP interpretations generated by one of the following procedures. For any randomly selected set of samples S, belonging to the same class c and producing the same encoding $\mathbf{e}$ at some layer l, we average their input features to produce $\mathbf{x}_S$ and:

- **Averaged output**: Calculate the LRP interpretation of $\mathbf{x}_S$ at the c^{th} output node. This corresponds to a standard LRP interpretation of the average sample in the set.
- **Averaged encoding**: Calculate the averaged LRP interpretation of $\mathbf{x}_S$ at all active nodes in $\mathbf{e}$.
- **Averaged core**: Calculate the averaged LRP interpretation of $\mathbf{x}_S$ at all core nodes in $\mathbf{e}$.
- **Averaged variation**: Calculate the averaged LRP interpretation of $\mathbf{x}_S$ at all variation nodes in $\mathbf{e}$.

Using the encoding sample sets in layer seven, as it most clearly shows our results, we generate and analyse the attribution maps for four randomly selected encoding sample sets of MNIST class zero in Figure 5.8. Remarkably, the averaged core interpretations share a striking resemblance to the averaged output interpretations.

Table 5.3 provides the cosine similarity scores between the four different types of attribution maps. According to these scores, the averaged output interpretations and averaged core interpretations share a similarity score of at least 0.980 throughout all results.



(a) Set size: 103 samples

(b) Set size: 199 samples

(c) Set size: 233 samples

(d) Set size: 637 samples

Figure 5.8: Visual comparisons of the four different interpretation types for four different encodings found in layer seven. From left to right, top to bottom, each plot corresponds to an encoding sample set with sizes 103, 199, 233 and 637. These results are shown for other classes and layers using the same network in Appendix B, Figures B.23 to B.27.

These results also highlight a few other notable comparisons. The averaged variation interpretations typically share a weak similarity score of lower than 0.500 with any other interpretation type, with the possible minor exception of the averaged encoding interpre-

Table 5.3: Cosine similarity comparisons between the different LRP interpretations shown in Figure 5.8. Encoding sample sets are represented according to their set sizes. The 'output', 'encoding', 'core' and 'variation' scores correspond to the different interpretations types presented in Figure 5.8.

	Set size 103	Set size 199	Set size 233	Set size 637
output vs **encoding:**	0.59917	0.90164	0.98315	0.99153
output vs **cores:**	0.98813	0.99225	0.99679	0.99339
output vs **variations:**	0.17070	-0.16340	0.52673	0.42981
encoding vs **cores:**	0.56302	0.90273	0.97738	0.99618
encoding vs **variations:**	0.88835	0.26238	0.66469	0.49035
cores vs **variations:**	0.12068	-0.17828	0.49166	0.41239

tations, which have reached a maximum similarity of 0.888 for encoding sample set (a). The averaged encoding interpretations share a significant similarity score of above 0.900 with both the averaged core interpretations and averaged output interpretations. These similarities however, are not as strong or as consistent (seen by the 0.563 similarity score for encoding sample set (a) in Figure 5.8) as the similarities seen between the averaged output and the averaged core interpretations, which have consistently been above 0.980.

5.4.3 Discussion

With the results provided in Figure 5.7, we showed that encoding sample sets are capable of grouping samples based on how they are internally represented by a DNN. Additionally, in Figure 5.8, we revealed the existence of a core group of nodes whose interpretations share large similarities (Table 5.3) with the interpretations generated by a standard LRP interpretation on the network's target output node.

We now further refine the arguments made in Sections 5.3.3 and 5.4.1. As previously mentioned in Section 5.3.1, a layer can be described as its own feature space, where every node is used to describe some feature within that space [15]. Knowing this, we propose that each class may be predominantly defined by a core group of features *(the*

core nodes) at a given layer. Along with these core nodes, some samples may also possess additional features that distinguish themselves from other samples in that same class. These distinguishing features are significant enough to require additional representation within the layer *(variation nodes)*. Through interpreting over both the core and variation nodes (thus over all active nodes for a given encoding sample set), we gain an overview of the features used to group samples by both types of nodes.

5.5 Set interpretations

In Section 5.4.2 we demonstrated that an interpretation generated on the average of several samples belonging to the same encoding sample set provides a comprehensive view of the most important features for the samples belonging to that set. Here we formally define our technique for using encoding sample sets as a means to enhance LRP by performing such group interpretations on class samples.

In Section 5.5.1, we define the technique itself. We then demonstrate the encoding of sample sets group samples according to their features through using this newly defined set interpretation technique on a modified version of the synthetic and MNIST networks in Sections 5.5.2 and 5.5.3.

5.5.1 Technique discussion

Here we formalise the method for our interpreter enhancing technique: *set interpretations* and describe how we personally used this technique.

Description

We define set interpretations as follows:

Set interpretation: *An interpretation that is generated on the averaged input features*

of several samples belonging to a particular encoding sample set for a given layer.

Set interpretations utilise the way encoding sample sets group samples based on how they are internally represented by a DNN to simplify the interpretation process. By interpreting the encoding sample sets themselves, rather than each individual sample, we generate interpretations on the local-to-global continuum. Each interpretation then represents numerous class samples.

These interpretations can be performed at any hidden layer, allowing you to theoretically 'see' the network's decision-making processes from layer to layer as groups of samples are classified. As shown in Section 5.4.2, when performed on encoding sample sets at deeper hidden layers, the set interpretations share great similarities with the respective samples' individual interpretations at the output layer.

Set interpretations can also be performed on the core and variation nodes as identified by the encoding sample sets in a given layer (Section 5.4.2). Interpreting core nodes will identify the most prominent features associated with each class, while variation node interpretations can identify the differences between sample groupings within that class in a single layer.

Usage

As discussed in Section 2.4 and by Davel et al. [12], the decision-making processes of nodes tend to become more class-specific towards the deeper layers of a network. Thus, when performing set interpretations, we generally use:

- The layers closest to the output layer, but not the output layer itself.
- The encoding sample sets with set size above 100.
- Class-specific encoding sample sets.

For each class, the output layer will consist of a singular active node, denoting which class the samples belong to, while all other nodes are inactive. Because of this, the output layer will only have one encoding sample set per class that contains all samples. This defeats the purpose of having multiple encoding sample sets divide the samples into meaningful groups and is why the output layer is not taken into consideration when performing set interpretations.

As shown in Section 5.4.1, the set interpretations are then generated by interpreting the average input features over a specified number of samples sharing the same encoding. Although we believe that increasing the number of samples averaged over will increase the accuracy of the set interpretation, it has not been demonstrated and is thus delegated to future work.

After generating the set interpretations, we generate additional set interpretations exclusively on the core nodes and variation nodes of the encoding sample set the interpretation was initially generated on using the same procedure as before. By doing so, we generate three interpretations that explain the following:

- ***Interpretation generated exclusively on core nodes:*** Highlights the features most prominently associated with the specific class of the encoding sample set.
- ***Interpretation generated exclusively on variation nodes:*** Highlights the features that distinguishes the samples from one another within the encoding sample set.
- ***Interpretation generated using all nodes of an encoding sample set:*** Interpretation which takes both the core nodes and variation nodes into consideration and provides a high-level explanation of the entire sample set.

5.5.2 Evaluation using synthetic data

We already examined interpretations of the core and variation nodes in Section 5.4.2. Here we analyse another important aspect of set interpretations, also initially demonstrated in

Section 5.4.2, which is the ability to generate interpretations that indicate the differences between sample groupings of the same class. In the following sections, we demonstrate the ability of encoding sample sets to separate class samples based on divergent features.

Experimental setup

We use a model trained on the same synthetic data set as in Section 3.3.2, but with one alteration. We reduce the number of classes from six to three, by relabelling classes that represented the different trigonometry functions, and essentially merging pairs of trigonometry function classes together. The classes are relabelled as follows:

- ***Class 1:*** Sine or Sec
- ***Class 2:*** Cosine or Cot
- ***Class 3:*** Tan or Cosec

We use the same model as described in Section 3.3.2 to train the network, the performance values of which are shown in Table 5.4.

Results

Here, similar to Section 5.4.1, we find and compare all encoding sample sets with a set size of greater than 100 for the second-to-last hidden layer (which most clearly shows our results). The results are shown in Figure 5.9, where we perform a set interpretation over five randomly selected samples for each encoding sample set.

Figure 5.9 shows representations of each encoding, and their respective set interpretations, for each class at the second-to-last hidden layer. The interpretations on the right do not clearly convey the visually most important features, and instead flag seemingly random background features as highly relevant. As established in Section 3.3.2, we purposefully chose a model with a smaller architecture, thus this could be due to the model not

Table 5.4: Architecture and performance values for the modified synthetic network.

Hyperparameter	**Value**	
Input dimensions	900	
Output dimensions	3	
Layer dimensions	20 10 10 10	
Epochs trained	1	
Sets	**Accuracy**	**Loss**
Train	1.00000	0.00013
Valid	1.00000	0.00013
Eval	1.00000	0.00013

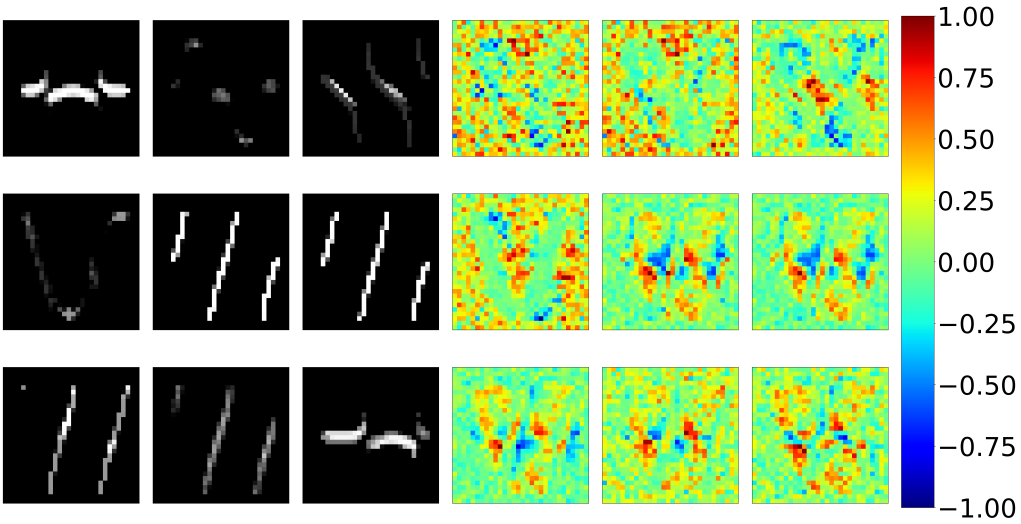


Figure 5.9: Visual depiction of the averaged input features (left) over five samples and their respective LRP interpretations (right) for all encoding sample sets in the second-to-last layer with a sample set size greater than 100 from the modified synthetic network. From left to right, the results of all the different classes are shown as follows: the first two images indicate the samples of class one, the second two images the samples of class two and the rest show the samples of class three. These results are shown for both the validation and evaluation sets using the same network in Appendix B, Figures B.28 and B.29.

having the flexibility to create clearly distinguishable interpretations consistently from the encoding sample sets.

The encoding sample sets on the left, however, seem to somewhat successfully detangle the original classes from one another. These encoding sample sets did contain classes with more visually distinguishable features than those sets that the interpreter was unable to separate. We continue our analysis using the MNIST data set in the following section.

5.5.3 Evaluation using MNIST data

Here we continue the analysis from Section 5.5.2 by performing a similar experiment with a model trained on a modified version of the MNIST data set.

Experimental setup

Similar to the synthetic network in Section 5.5.2, we modify the MNIST data set by relabelling classes as follows:

- ***Class 0:*** MNIST class 0 or MNIST class 1
- ***Class 1:*** MNIST class 2 or MNIST class 3
- ***Class 2:*** MNIST class 4 or MNIST class 5
- ***Class 3:*** MNIST class 6 or MNIST class 7
- ***Class 4:*** MNIST class 8 or MNIST class 9

This is done in order to create distinct groups within each class. We train the same model as specified in Section 3.3.3 on the new data set, which yielded the performance values shown in Table 5.5.

Results

In this section, we find and compare the encoding sample sets in the last hidden layer of our model that has a set size greater than 100. As in Section 5.5.2, we perform set

Table 5.5: Architecture and performance values for the modified MNIST network.

Hyperparameter	**Value**	
Input dimensions	784	
Output dimensions	5	
Layer dimensions	100 100 100 100 100 100 100 100 100 100	
Epochs trained	81	
Sets	**Accuracy**	**Loss**
Train	1.00000	0.00003
Valid	0.98100	0.18860
Eval	0.97960	0.26074

interpretations for each encoding sample set over of their respective five randomly selected samples.

Figure 5.10 shows the encoding sample sets for class zero of the modified data set. The encoding sample sets on the left clearly distinguish the original two MNIST classes from one another. Unlike the synthetic network in 5.5.2, the interpretations on the right also clearly distinguish between the original MNIST classes that the encoding sample set represents, by visualising the distinct features for both classes. This could indicate that the larger flexibility of the MNIST network helps to generate clear interpretations.

Figure 5.11 shows the same results, but for class one of our modified MNIST network. Despite the original MNIST classes shown here having many shared features, the encoding sample sets are generally still able to separate them. Interestingly, since so many features are shared between samples, the interpretations for samples from separate MNIST classes are very similar. These results demonstrate that encoding sample sets can group and categorise samples based on their features, even within the same class.

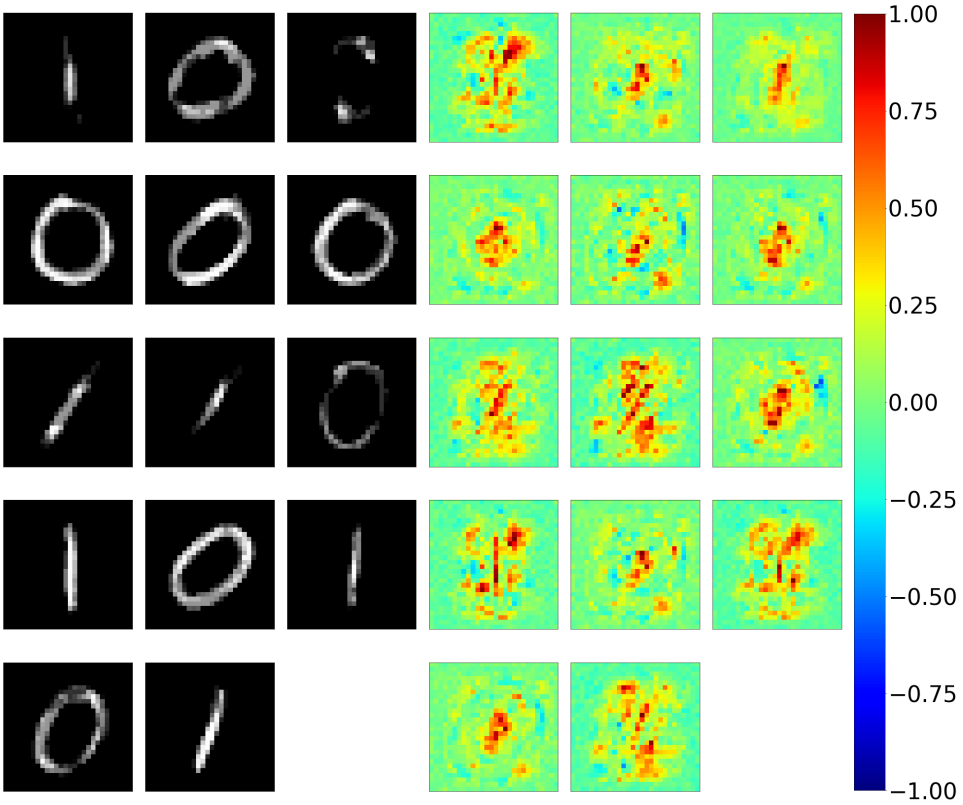


Figure 5.10: Visual depiction of the averaged input features (left) over five samples and their respective LRP interpretations (right) for all class zero encoding sample sets in the last layer with a sample set size greater than 100 from the modified MNIST network. These results are shown for both the validation and evaluation sets using the same network in Appendix B, Figures B.30 and B.31.

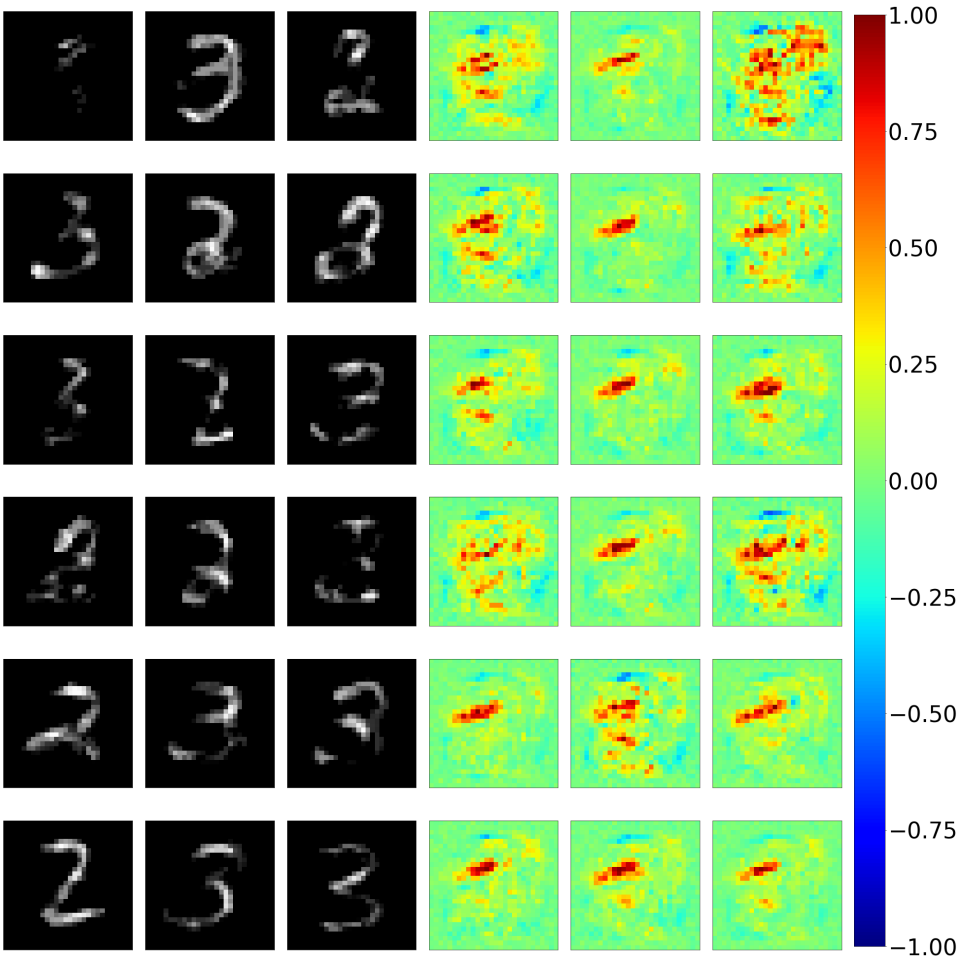


Figure 5.11: Visual depiction of the averaged input features (left) over five samples and their respective LRP interpretations (right) for all class one encoding sample sets in the last layer with a sample set size greater than 100 from the modified MNIST network.

5.6 Conclusion

In this chapter, we built upon the works of [12], [13], [15] by proposing *encoding sample sets* as a new category of sample sets. After some initial experiments we hypothesised that any class could be largely defined by a *core* group of nodes, and that additional *variation* nodes are used exclusively to represent the small differentiating features between samples of a certain class. We confirmed this hypothesis by calculating the cosine similarities between the interpretations generated on the averaged input features over several samples at the output node and the interpretations generated on the averaged input features over several samples at the core nodes of a given encoding at a particular layer as was shown in Figure 5.8. The results showed a constant similarity score of above 0.980 for all interpretations. Interpretations of non-core node groups showed little similarity to these core node groups or averaged output node interpretations.

The process whereby the averaged input features over several samples belonging the same encoding sample is used to generate a single interpretation representing the samples of said encoding sample sets is dubbed as a *set interpretation*. By analysing the layers of the networks, we found support for the idea that encoding sample sets are capable of categorising samples even within the same class. We tested this by analysing the set interpretations for modified versions of our synthetic and MNIST networks as originally described in Sections 3.3.2 and 3.3.3. The networks were modified to have a reduced number of classes that are the combination of class pairs from the original data sets. The results showed that the encoding sample sets were indeed capable of separating the original classes despite being combined into one super class.

Thus, through using encoding sample sets in conjunction with a modern interpretation technique (in this case LRP), we are able to simplify the interpretation process by generating single interpretations which can represent sets of samples. These *set interpretations* can then also be used to further indicate the features most responsible for the samples' classifications and/or the features that distinguish the samples from one another.

Chapter 6

Conclusion

A review of our study's key findings and the implications thereof along with what possible future research could hold.

6.1 Introduction

The original aim of this study was to measure the applicability of a niche methodology, sample set analysis, as a tool for enhancing the capabilities of existing interpretability methods. In this chapter, we conclude our study by examining the degree to which our results fulfil the original objectives as established in Chapter 1.

In Section 6.2 we discuss the key findings of our study, along with the implications of these findings in Section 6.3. Lastly, we discuss possible future research in Section 6.4.

6.2 Key findings

Here we list and discuss what we believe to be the key findings of our study.

Using zero-valued input features

We trained two benchmark MLP networks that were used during the course of this study. One network was trained on the MNIST data set, on which we performed most of our experiments, and the second network was trained on a custom synthetic data set to initially test our implemented interpreter and provide additional results. Additionally, we implemented our own in-house LRP codebase, which was verified by comparing its results to that of Captum's open-source codebase on our benchmark networks.

The results in Section 3.5.2 indicated that zero-value input features were not taken into consideration when generating interpretations using the LRP interpreter codebase. Any feature can only be attributed relevance if its outgoing z (see Eq. 2.7) value is greater than zero. This is a feature of a standard MLP, not LRP specifically.

This makes standard interpretations on data sets such as MNIST and our synthetic data sets, which typically contain a large number of zero-value features, less informative. We remedied this issue by normalising our data sets between the values *[-1, 1]* before use, which reduced the number of zero-value input features to a negligible figure. While this is a straightforward solution, it emphasises the significant impact scaling features can have on model development and interpretation, especially where a strong background or foreground is present.

The interpretability capabilities of node sample sets

Node sample sets refers to the sets of samples that cause a given node to activate. In Sections 4.4.1, 4.4.2 and 4.4.3 we used the Jaccard similarity index to analyse the relationship between the different node-specific sample sets of our MNIST network, per layer,

and across the entire network. Node sample sets did not seem to show promise as an interpretation enhancement technique. This could be due to the following reasons:

- Class-specific node sample sets deeper within the network tend to be either very large or very small, with few falling between these extremes. This was also observed in the works of Davel et al. [12].
- The sizes of larger sample sets significantly increases within the deeper layers of the network.
- The number of practically identical node sample sets, with sizes almost equal to the number of samples in a given class, also increases towards the deeper layers of the network.

The existence of a significant number of large, nearly identical, node sample sets was an important observation. With sample set sizes approximately equal to the number of class samples, we believe that these nodes play a fundamental role in the classification process.

Proposing the concept of encoding sample sets

Taking into account the observations made from our node sample sets analyses, we proposed a new sample set category called *encoding sample sets*. Encoding sample sets are defined as the sets of samples that activate a given sample encoding in a specific layer, where a sample encoding refers to the binary activation pattern of nodes in a layer that do or do not activate for the given samples. The definition of encoding sample sets (as further discussed in Section 6.3) made additional key findings possible.

While investigating encoding sample sets in Section 5.2, we discovered that, similar to node sample sets, the number of encoding sample sets decreases towards the deeper layers of the network. However, as shown in Figure 5.4, unlike the node sample sets studied in Section 4.4.3, the Jaccard similarity decreases in proportion to their sample set sizes with the exception of the largest encoding sample sets that still retain some similarity, of which there are only a few.

Through analysing the encoding sample sets of the MNIST network via the interpreter codebase, we observed the following:

- Encoding sample sets tend to group samples with similar features. This results in a few incredibly large encoding sample sets that represent the samples with the most common features associated with that class. Smaller encoding sample sets still belong to the same class, but contain features that differ slightly from the norm.
- All encoding sample sets within the deeper layers of the network typically share a large majority of the active nodes in a layer (core nodes). We demonstrated that these active nodes form the features most important for classification, while the active nodes not shared among all encoding sample sets (variation nodes) represent the features that differ between class samples.

6.3 Contributions

In this study we expanded on the work by Davel et al. [12], [13] by introducing the concept of encoding sample sets as mentioned in Section 6.2, and formally defined in Section 5.2. We discovered that encoding sample sets are capable of identifying the following sets of active nodes within a layer:

- *Core Nodes:* The active nodes shared among all encoding sample sets within a layer. These nodes contain the most important features necessary for classification.
- *Variation Nodes:* The active nodes not shared among all encoding sample sets within a layer. These nodes contain the features that differentiate the samples from the different encoding sample set groupings.

When used in conjunction with an appropriate interpreter, such as our LRP codebase, encoding sample sets can be used to generate *set interpretations* as defined in Section 5.5.1, where we interpret an encoding sample set by averaging the interpretations of samples

belonging to the same sample encoding. This way we can generate single interpretations, which are capable of representing multiple samples, and which we have demonstrated may share a high cosine similarity score with the interpretations performed per sample. Set interpretations can also be altered to highlight the core and variation nodes, which highlight the features crucial for classification and those features that distinguish sample groupings from one another, respectively.

Set interpretations comprise the most important contribution, as they fulfil the original objective of this study to determine whether sample set analysis can be used to enhance modern interpretation techniques. Set interpretations are capable of enhancing certain interpretation techniques by simplifying the interpretation process with the use of group interpretations.

An additional contribution is the development of an LRP interpreter codebase capable of extracting set interpretations. This codebase also fixes one of the implementation errors by Captum's LRP codebase, with which our codebase was compared. We refer the reader to Section 3.4.3 for more details on this.

6.4 Future work

This exploratory study only provides the initial foundation of using sample set analysis for interpretability purposes. Although we propose encoding sample sets and set interpretations as a new conceptual approach for neural network interpretability enhancement, additional work is required to refine and test these techniques in detail. This may include alternative methods for exploring encoding sample sets such as:

- *Averaging over interpretations rather than input features.* The process we followed when performing set interpretations was to generate the interpretation on the averaged input features over several samples belonging to the same encoding sample set. An alternative would be to produce a set interpretation by averaging the interpretations generated on the input features of each individual sample belonging to

an encoding sample set.

- *Investigating encoding sample sets with a variety of set sizes.* In our analyses of encoding sample sets, we mainly limited ourselves to encodings sample sets with set sizes larger than 100 samples for simplicity's sake. In future studies, we would like to further explore encoding sample sets for a variety of set sizes.

- *Determining how averaging over different numbers of samples effect set interpretations.* Although we believe that increasing the number of samples averaged over when performing set interpretations will increase the accuracy of said set interpretations, it is yet to be investigated.

Additional future research may include the following:

- *Performing set interpretations for more complex environments.* The next logical step would be to determine whether the results found during this study can be replicated for more difficult classification tasks such as CIFAR-10 [84]. This will naturally include the investigation of whether our set interpretation technique is compatible with more complex architectures such as CNNs.

- *Determining whether sample set analysis is capable of performing interpretation.* We have shown that sample set analysis is capable of enhancing certain interpretability methods, but we have yet to investigate its capabilities as an interpretability technique itself. We believe that this is possible through utilising the activation paths, as was briefly discussed in Section 2.4, generated via sample sets.

- *Examining the applicability of encoding sample sets, and subsequently set interpretations, with regard to other interpretability methods.* Our study was somewhat limited in the sense that our encoding sample set analyses were only used in conjunction with the backpropagating interpretation technique LRP. It would be beneficial to determine the range of interpretation methods our technique is compatible with, like the other techniques mentioned in Section 2.3.3.

- *Determining the usefulness of set interpretations with regard to other tasks such as regression.* This study has been solely focused on the topic of classification type problems. In future studies, we would like to explore whether set interpretations can be used for different deep neural network tasks.
- *Further exploring the concepts of core and variation nodes.*

6.5 Conclusion

The ability to explain the reasoning behind the classification processes of neural networks has been demonstrated to be invaluable for some areas of research. To this end, we used the niche methodology of sample set analysis to introduce a novel technique to help simplify the interpretation process for certain modern extrinsically interpretable techniques. We showed that this technique is capable of not only dividing class samples into groups based on their features, but can also generate single interpretations which represent these groups *en masse*. Additionally, our technique is also capable of identifying the set of features/nodes that have the most influence on the classification process. We believe that this study provides some of the foundations for interpreting DNNs with sample set analysis, and hope that future research regarding this topic will lead to significant advancement within the field.

Bibliography

[1] M. Bojarski, P. Yeres, A. Choromanska, K. Choromanski, B. Firner, L. Jackel, and U. Muller, "Explaining how a deep neural network trained with end-to-end learning steers a car," *arXiv preprint arXiv:1704.07911*, 2017.

[2] Z. Che, S. Purushotham, R. Khemani, and Y. Liu, "Distilling knowledge from deep networks with applications to healthcare domain," *arXiv preprint arXiv:1512.03542*, 2015.

[3] R. Caruana, Y. Lou, J. Gehrke, P. Koch, M. Sturm, and N. Elhadad, "Intelligible models for healthcare: Predicting pneumonia risk and hospital 30-day readmission," in *Proceedings of the ACM SIGKDD*, vol. 2015-August, 2015, pp. 1721–1730, ISBN: 9781450336642. DOI: 10.1145/2783258.2788613.

[4] S. Bach, A. Binder, G. Montavon, F. Klauschen, K.-R. Müller, and W. Samek, "On pixel-wise explanations for non-linear classifier decisions by layer-wise relevance propagation," *PLOS ONE*, vol. 10, no. 7, pp. 1–46, Jul. 2015. DOI: 10.1371/journal.pone.0130140.

[5] K. Simonyan, A. Vedaldi, and A. Zisserman, "Deep inside convolutional networks: Visualising image classification models and saliency maps," *CoRR*, vol. abs/1312.6034, 2014.

[6] L. Arras, F. Horn, G. Montavon, K.-R. Müller, and W. Samek, "'what is relevant in a text document?': An interpretable machine learning approach," *PloS one*, vol. 12, no. 8, e0181142, 2017.

[7] M. Raghu and E. Schmidt, "A survey of deep learning for scientific discovery," *CoRR*, vol. abs/2003.11755, 2020. arXiv: 2003.11755.

[8] A. Barredo Arrieta, N. Díaz-Rodríguez, J. Del Ser, A. Bennetot, S. Tabik, A. Barbado, S. Garcia, S. Gil-Lopez, D. Molina, R. Benjamins, R. Chatila, and F. Herrera, "Explainable Explainable Artificial Intelligence (XAI): Concepts, taxonomies, opportunities and challenges toward responsible AI," *Information Fusion*, vol. 58, pp. 82–115, 2020, ISSN: 15662535. DOI: 10.1016/j.inffus.2019.12.012. arXiv: 1910.10045.

[9] W. Samek, G. Montavon, S. Lapuschkin, C. J. Anders, and K.-R. Müller, "Explaining deep neural networks and beyond: A review of methods and applications," *Proceedings of the IEEE*, vol. 109, no. 3, pp. 247–278, 2021.

[10] G. Montavon, W. Samek, and K.-R. Müller, "Methods for interpreting and understanding deep neural networks," *Digital Signal Processing*, vol. 73, pp. 1–15, 2018, ISSN: 1051-2004. DOI: https://doi.org/10.1016/j.dsp.2017.10.011.

[11] A. Adadi and M. Berrada, "Peeking inside the black-box: A survey on explainable artificial intelligence (XAI)," *IEEE Access*, vol. 6, pp. 52 138–52 160, 2018. DOI: 10.1109/ACCESS.2018.2870052.

[12] M. Davel, M. Theunissen, A. Pretorius, and E. Barnard, "DNNs as layers of cooperating classifiers," in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 34, 2020, pp. 3725–3732.

[13] M. Davel, "Activation gap generators in neural networks," in *South African Forum for Artificial Intelligence Research (FAIR)*, 2019, pp. 64–76.

[14] M. H. Davel, "Using summary layers to probe neural network behaviour," *South African Computer Journal*, vol. 32, no. 2, pp. 102–123, 2020.

[15] M. W. Theunissen, M. Davel, and E. Barnard, "Insights regarding overfitting on noise in deep learning," in *South African Forum for Artificial Intelligence Research (FAIR)*, 2019, pp. 49–63.

[16] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning*. MIT Press, 2016, pp. 1–26, 118–120, 164–223, 271–325, 96–161, http://www.deeplearningbook.org.

[17] Y. LeCun, C. Cortes, and C. Burges, *MNIST database of handwritten digits*, http://yann.lecun.com/exdb/mnist/.

[18] A. Pretorius, M. Davel, and E. Barnard, "ReLU and sigmoidal activation functions," in *FAIR 2020*, 2019. [Online]. Available: http://ceur-ws.org/Vol-2540/FAIR2019_paper_52.pdf.

[19] R. Paulus, C. Xiong, and R. Socher, "A deep reinforced model for abstractive summarization," *ArXiv*, vol. abs/1705.04304, 2018.

[20] A. Graves, A.-r. Mohamed, and G. Hinton, "Speech recognition with deep recurrent neural networks," in *2013 IEEE international conference on acoustics, speech and signal processing*, Ieee, 2013, pp. 6645–6649.

[21] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, and I. Polosukhin, "Attention is all you need," *Advances in Neural Information Processing Systems*, vol. 30, 2017.

[22] X. Wang, R. Girshick, A. Gupta, and K. He, "Non-local neural networks," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2018, pp. 7794–7803.

[23] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," *Advances in Neural Information Processing Systems*, vol. 25, 2012.

[24] V. Nair and G. E. Hinton, "Rectified linear units improve restricted boltzmann machines," in *ICML*, 2010, pp. 807–814, ISBN: 9781605589077. [Online]. Available: https://icml.cc/Conferences/2010/papers/432.pdf.

[25] C. Nwankpa, W. Ijomah, A. Gachagan, and S. Marshall, "Activation functions: Comparison of trends in practice and research for deep learning," *arXiv preprint arXiv:1811.03378*, 2018.

[26] A. Zhang, Z. C. Lipton, M. Li, and A. J. Smola, "Dive into deep learning," *arXiv preprint arXiv:2106.11342*, pp. 97–102, 87–96, 152–159, 142–152, 2021.

[27] Y. LeCun, Y. Bengio, and G. Hinton, "Deep learning," *nature*, vol. 521, no. 7553, pp. 436–444, 2015.

[28] R. Yamashita, M. Nishio, R. Do, and K. Togashi, "Convolutional neural networks: An overview and application in radiology," *Insights into Imaging*, vol. 9, Jun. 2018. DOI: 10.1007/s13244-018-0639-9.

[29] A. Graves, *Supervised Sequence Labelling with Recurrent Neural Networks*, ser. Studies in Computational Intelligence. Springer Berlin Heidelberg, 2012, pp. 368, 388, 407, 455, ISBN: 9783642247972.

[30] D. E. Rumelhart, G. E. Hinton, and R. J. Williams, "Learning representations by back-propagating errors," *nature*, vol. 323, no. 6088, pp. 533–536, 1986.

[31] K. J. Lang, A. H. Waibel, and G. E. Hinton, "A time-delay neural network architecture for isolated word recognition," *Neural networks*, vol. 3, no. 1, pp. 23–43, 1990.

[32] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural computation*, vol. 9, no. 8, pp. 1735–1780, 1997.

[33] R. Reed and R. J. MarksII, *Neural smithing: supervised learning in feedforward artificial neural networks*. Mit Press, 1999, pp. 155–156.

[34] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," *CoRR*, vol. abs/1412.6980, 2015.

[35] S. Russell and P. Norvig, "Artificial intelligence: A modern approach," p. 124, 2002.

[36] J. Bergstra, R. Bardenet, Y. Bengio, and B. Kégl, "Algorithms for hyper-parameter optimization," *Advances in Neural Information Processing Systems*, vol. 24, 2011.

[37] B. Shahriari, K. Swersky, Z. Wang, R. P. Adams, and N. De Freitas, "Taking the human out of the loop: A review of bayesian optimization," *Proceedings of the IEEE*, vol. 104, no. 1, pp. 148–175, 2015.

[38] S. K. Kumar, "On weight initialization in deep neural networks," *CoRR*, vol. abs/1704.08863, 2017. arXiv: 1704.08863.

[39] X. Glorot, A. Bordes, and Y. Bengio, "Deep sparse rectifier neural networks," in *Proceedings of the fourteenth International Conference on Artificial Intelligence and Statistics*, ser. Proceedings of Machine Learning Research, `http://proceedings.mlr.press/v15/glorot11a/glorot11a.pdf`, vol. 15, Fort Lauderdale, FL, USA: PMLR, Apr. 2011, pp. 315–323.

[40] K. He, X. Zhang, S. Ren, and J. Sun, "Delving deep into rectifiers: Surpassing human-level performance on imagenet classification," *2015 IEEE International Conference on Computer Vision (ICCV)*, pp. 1026–1034, 2015.

[41] PyTorch, *Torchvision packaged datasets.* `https://pytorch.org/docs/stable/torchvision/datasets.html`.

[42] S. Ioffe and C. Szegedy, "Batch normalization: Accelerating deep network training by reducing internal covariate shift," *CoRR*, vol. abs/1502.03167, 2015. arXiv: `1502.03167`.

[43] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov, "Dropout: A simple way to prevent neural networks from overfitting," *The journal of machine learning research*, vol. 15, no. 1, pp. 1929–1958, 2014.

[44] G. E. Hinton, N. Srivastava, A. Krizhevsky, I. Sutskever, and R. R. Salakhutdinov, "Improving neural networks by preventing co-adaptation of feature detectors," *arXiv preprint arXiv:1207.0580*, 2012.

[45] C. M. Bishop and N. M. Nasrabadi, "Pattern recognition and machine learning," *J. Electronic Imaging*, vol. 16, p. 049 901, 2007.

[46] W. Samek, A. Binder, G. Montavon, S. Lapuschkin, and K.-R. Müller, "Evaluating the visualization of what a deep neural network has learned," *IEEE Transactions on Neural Networks and Learning Systems*, vol. 28, pp. 2660–2673, 2017.

[47] M. D. Zeiler and R. Fergus, "Visualizing and understanding convolutional networks," in *European conference on computer vision*, Springer, 2014, pp. 818–833.

[48] W. Samek and K.-R. Müller, "Towards explainable artificial intelligence," in *Explainable AI: interpreting, explaining and visualizing deep learning*, Springer, 2019, pp. 5–22.

[49] M. Everingham, L. Van Gool, C. K. Williams, J. Winn, and A. Zisserman, "The pascal visual object classes (voc) challenge," *International journal of computer vision*, vol. 88, no. 2, pp. 303–338, 2010.

[50] S. Lapuschkin, A. Binder, G. Montavon, K.-R. Muller, and W. Samek, "Analyzing classifiers: Fisher vectors and deep neural networks," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2016, pp. 2912–2920.

[51] S. Lapuschkin, S. Wäldchen, A. Binder, G. Montavon, W. Samek, and K. Müller, *Unmasking clever hans predictors and assessing what machines really learn. nat. commun., 10, 1096*, 2019.

[52] M. Sundararajan, A. Taly, and Q. Yan, "Axiomatic attribution for deep networks," in *International conference on machine learning*, PMLR, 2017, pp. 3319–3328.

[53] C. Molnar, "A guide for making black box models explainable," *URL: https://christophm. github. io/interpretable-ml-book*, 2018.

[54] V. Schetinin, J. E. Fieldsend, D. Partridge, T. J. Coats, W. J. Krzanowski, R. M. Everson, T. C. Bailey, and A. Hernandez, "Confident interpretation of bayesian decision tree ensembles for clinical applications," *IEEE Transactions on Information Technology in Biomedicine*, vol. 11, no. 3, pp. 312–319, 2007.

[55] B. Letham, C. Rudin, T. H. McCormick, and D. Madigan, "Interpretable classifiers using rules and bayesian analysis: Building a better stroke prediction model," *The Annals of Applied Statistics*, vol. 9, no. 3, pp. 1350–1371, 2015.

[56] S. Sarkar, T. Weyde, A. Garcez, G. G. Slabaugh, S. Dragicevic, and C. Percy, "Accuracy and interpretability trade-offs in machine learning applied to safer gambling," in *CEUR Workshop Proceedings*, CEUR Workshop Proceedings, vol. 1773, 2016.

[57] P. Linardatos, V. Papastefanopoulos, and S. Kotsiantis, "Explainable ai: A review of machine learning interpretability methods," *Entropy*, vol. 23, no. 1, p. 18, 2020.

[58] Y. Zhang, P. Tiňo, A. Leonardis, and K. Tang, "A survey on neural network interpretability," *IEEE Transactions on Emerging Topics in Computational Intelligence*, 2021.

[59] Z. Yang, A. Zhang, and A. Sudjianto, "Gami-net: An explainable neural network based on generalized additive models with structured interactions," *Pattern Recognition*, vol. 120, p. 108 192, 2021.

[60] S. Lundberg and S.-I. Lee, "A unified approach to interpreting model predictions," *CoRR*, vol. abs/1705.07874, 2017. arXiv: 1705.07874.

[61] G. Montavon, A. Binder, S. Lapuschkin, W. Samek, and K.-R. Müller, "Layer-wise relevance propagation: An overview," in. Sep. 2019, pp. 193–209, ISBN: 978-3-030-28953-9. DOI: 10.1007/978-3-030-28954-6_10.

[62] M. Ancona, E. Ceolini, A. C. Öztireli, and M. H. Gross, "Towards better understanding of gradient-based attribution methods for deep neural networks," *CoRR*, vol. abs/1711.06104, 2017. arXiv: 1711.06104.

[63] L. M. Zintgraf, T. S. Cohen, T. Adel, and M. Welling, "Visualizing deep neural network decisions: Prediction difference analysis," *CoRR*, vol. abs/1702.04595, 2017. arXiv: 1702.04595.

[64] B. Zhou, A. Khosla, A. Lapedriza, A. Oliva, and A. Torralba, "Learning deep features for discriminative localization," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2016, pp. 2921–2929.

[65] R. R. Selvaraju, M. Cogswell, A. Das, R. Vedantam, D. Parikh, and D. Batra, "Grad-CAM: Visual explanations from deep networks via gradient-based localization," in *Proceedings of the IEEE international conference on computer vision*, 2017, pp. 618–626.

[66] M. T. Ribeiro, S. Singh, and C. Guestrin, ""why should i trust you?": Explaining the predictions of any classifier," *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pp. 1135–1144, 2016.

[67] A. Shrikumar, P. Greenside, and A. Kundaje, "Learning important features through propagating activation differences," in *International conference on machine learning*, PMLR, 2017, pp. 3145–3153.

[68] J. T. Springenberg, A. Dosovitskiy, T. Brox, and M. A. Riedmiller, "Striving for simplicity: The all convolutional net," *CoRR*, vol. abs/1412.6806, 2015.

[69] G. Montavon, S. Lapuschkin, A. Binder, W. Samek, and K.-R. Müller, "Explaining nonlinear classification decisions with deep taylor decomposition," *Pattern Recognition*, vol. 65, pp. 211–222, 2017.

[70] M. Robnik-Šikonja and M. Bohanec, "Perturbation-based explanations of prediction models," in *Human and machine learning*, Springer, 2018, pp. 159–175.

[71] A. Brasoveanu, M. Moodie, and R. Agrawal, "Textual evidence for the perfunctoriness of independent medical reviews.," in *KiML@ KDD*, 2020, pp. 1–9.

[72] J. Adebayo, J. Gilmer, M. Muelly, I. Goodfellow, M. Hardt, and B. Kim, "Sanity checks for saliency maps," *Advances in Neural Information Processing Systems*, vol. 31, 2018.

[73] Y. Yang, V. Tresp, M. Wunderle, and P. A. Fasching, "Explaining therapy predictions with layer-wise relevance propagation in neural networks," in *2018 IEEE International Conference on Healthcare Informatics (ICHI)*, IEEE, 2018, pp. 152–162.

[74] D. Slack, S. Hilgard, E. Jia, S. Singh, and H. Lakkaraju, "Fooling lime and shap: Adversarial attacks on post hoc explanation methods," in *Proceedings of the AAAI/ACM Conference on AI, Ethics, and Society*, 2020, pp. 180–186.

[75] D. G. Haasbroek and M. H. Davel, "Exploring neural network training dynamics through binary node activations," *Proceedings of the Southern African Conference for Artificial Intelligence Research*, 2020, Accepted for publication.

[76] W. Samek, G. Montavon, A. Binder, S. Lapuschkin, and K.-R. Müller, "Interpreting the predictions of complex ml models by layer-wise relevance propagation," *ArXiv*, vol. abs/1611.08191, 2016.

[77] G. Van Rossum and F. L. Drake, *Python 3 Reference Manual*. Scotts Valley, CA: CreateSpace, 2009, ISBN: 1441412697.

[78] J. D. Hunter, "Matplotlib: A 2d graphics environment," *Computing in Science & Engineering*, vol. 9, no. 3, pp. 90–95, 2007. DOI: 10.1109/MCSE.2007.55.

[79] N. Kokhlikyan, V. Miglani, M. Martin, E. Wang, B. Alsallakh, J. Reynolds, A. Melnikov, N. Kliushkina, C. Araya, S. Yan, and O. Reblitz-Richardson, *Captum: A unified and generic model interpretability library for pytorch*, 2020. arXiv: 2009.07896 [cs.LG].

[80] M. Alber, S. Lapuschkin, P. Seegerer, M. Hägele, K. T. Schütt, G. Montavon, W. Samek, K.-R. Müller, S. Dähne, and P.-J. Kindermans, "iNNvestigate neural networks!" *J. Mach. Learn. Res.*, vol. 20, no. 93, pp. 1–8, 2019.

[81] M. S. Charikar, "Similarity estimation techniques from rounding algorithms," in *Proceedings of the thiry-fourth annual ACM symposium on Theory of computing*, 2002, pp. 380–388.

[82] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei, "Imagenet: A large-scale hierarchical image database," in *2009 IEEE conference on computer vision and pattern recognition*, Ieee, 2009, pp. 248–255.

[83] A. Krizhevsky, G. Hinton, *et al.*, "Learning multiple layers of features from tiny images," 2009.

[84] A. Krizhevsky, V. Nair, and G. Hinton, *CIFAR10 dataset*, https://www.cs.toronto.edu/~kriz/cifar.html.

[85] P. Jaccard, "The distribution of the flora in the alpine zone. 1," *New phytologist*, vol. 11, no. 2, pp. 37–50, 1912.

[86] R. Real and J. M. Vargas, "The probabilistic basis of Jaccard's index of similarity," *Systematic biology*, vol. 45, no. 3, pp. 380–385, 1996.

[87] S. Salvatore, K. Rand, I. Grytten, E. Ferkingstad, D. Domanska, L. Holden, M. Gheorghe, A. Mathelier, I. Glad, and G. Sandve, *Beware the Jaccard: The choice of metric is important and non-trivial in genomic colocalisation analysis.* Nov. 2018. DOI: 10.1101/479253.

Appendix A

Supplemental content: Chapter 4

In this chapter we provide supplementary results for Chapter 4.

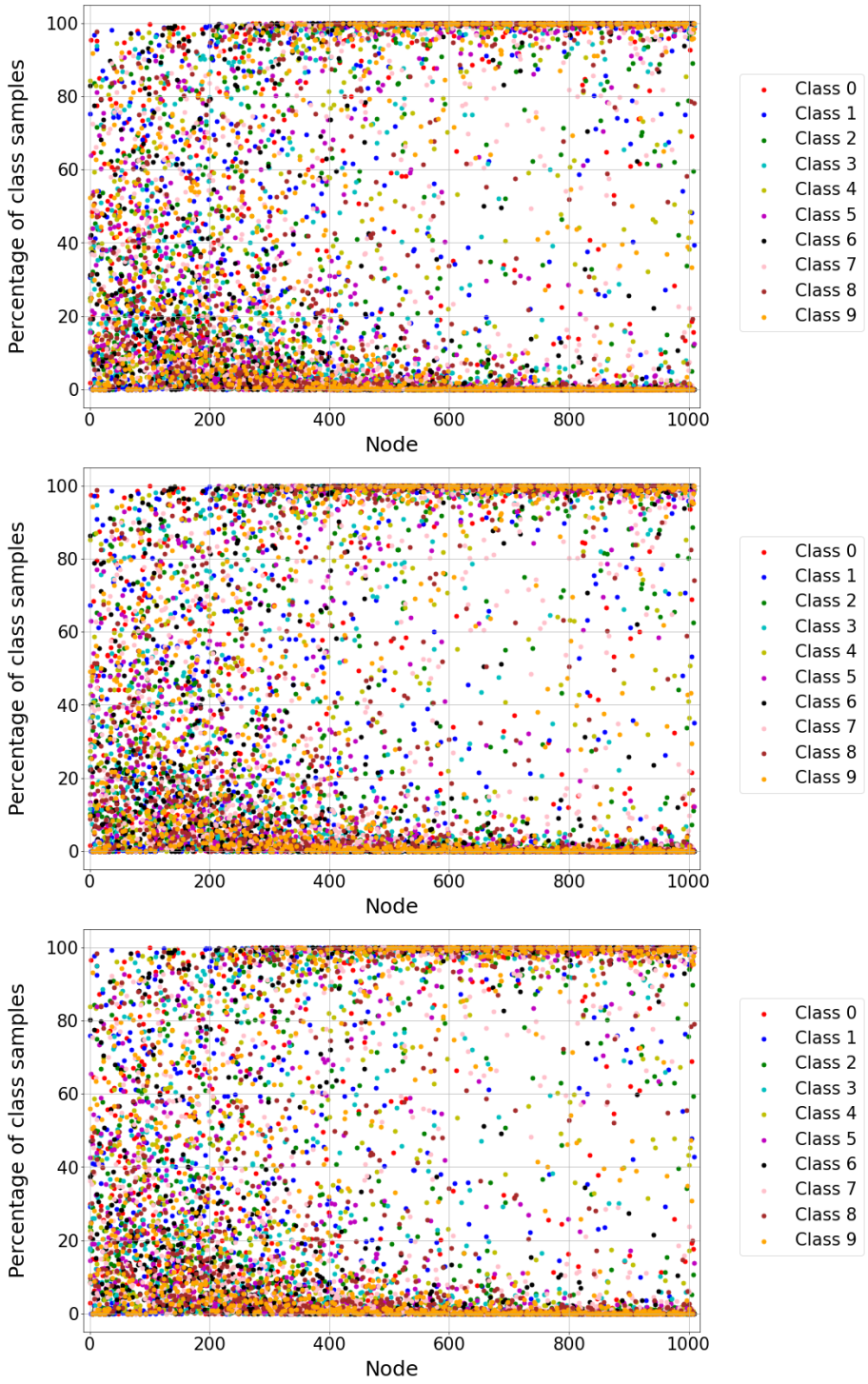


Figure A.1: Percentage of class samples activated per node for every class using the MNIST network (seed 3). From top to bottom, the figures show the results for the training, validation and evaluation sets respectively. Nodes are arranged according to layer, where node 0-99 represent the first layer, node 100-199 represent the second layer, etc.

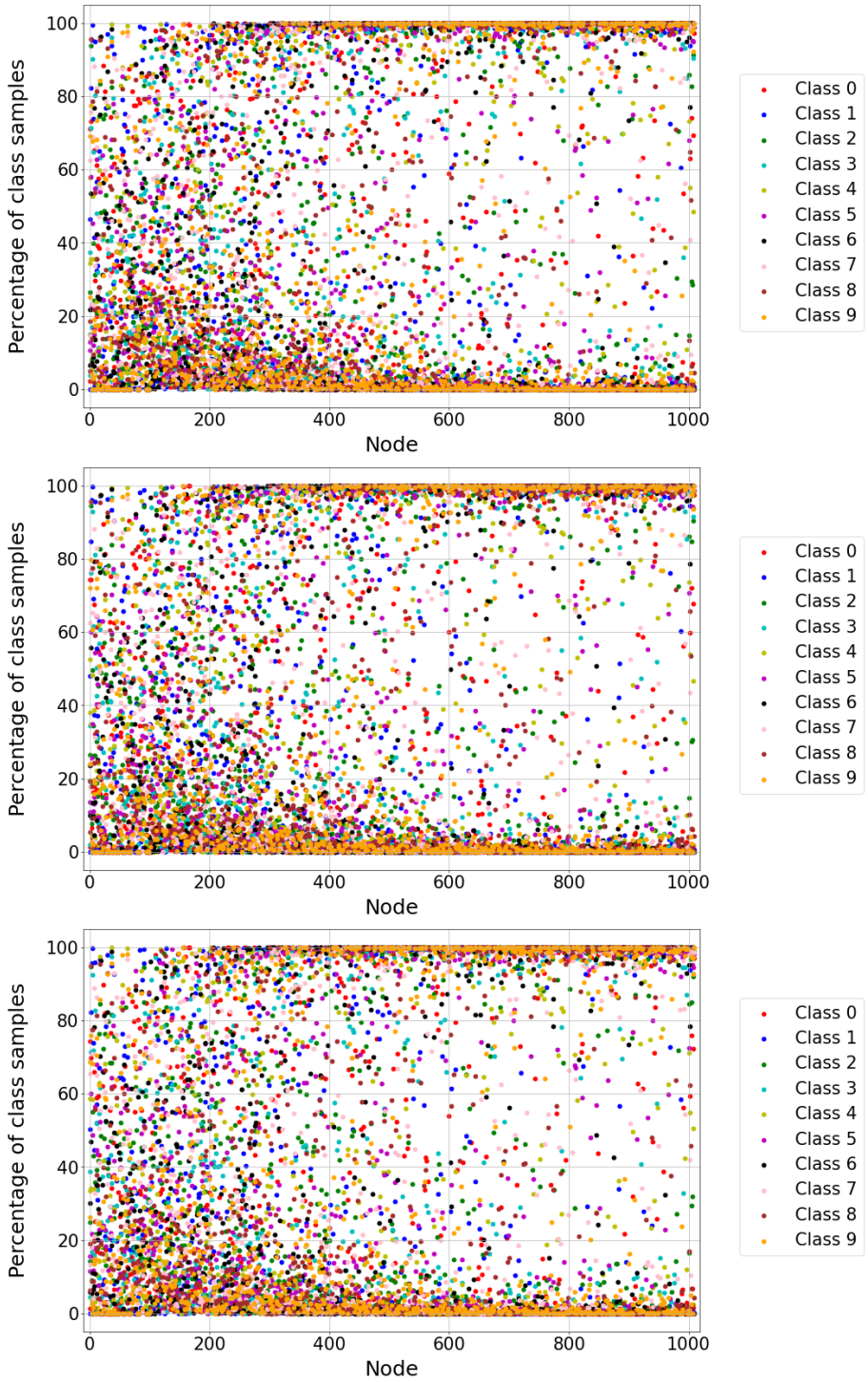


Figure A.2: Percentage of class samples activated per node for every class using the MNIST network (seed 5). From top to bottom, the figures show the results for the training, validation and evaluation sets respectively. Nodes are arranged according to layer, where node 0-99 represent the first layer, node 100-199 represent the second layer, etc.

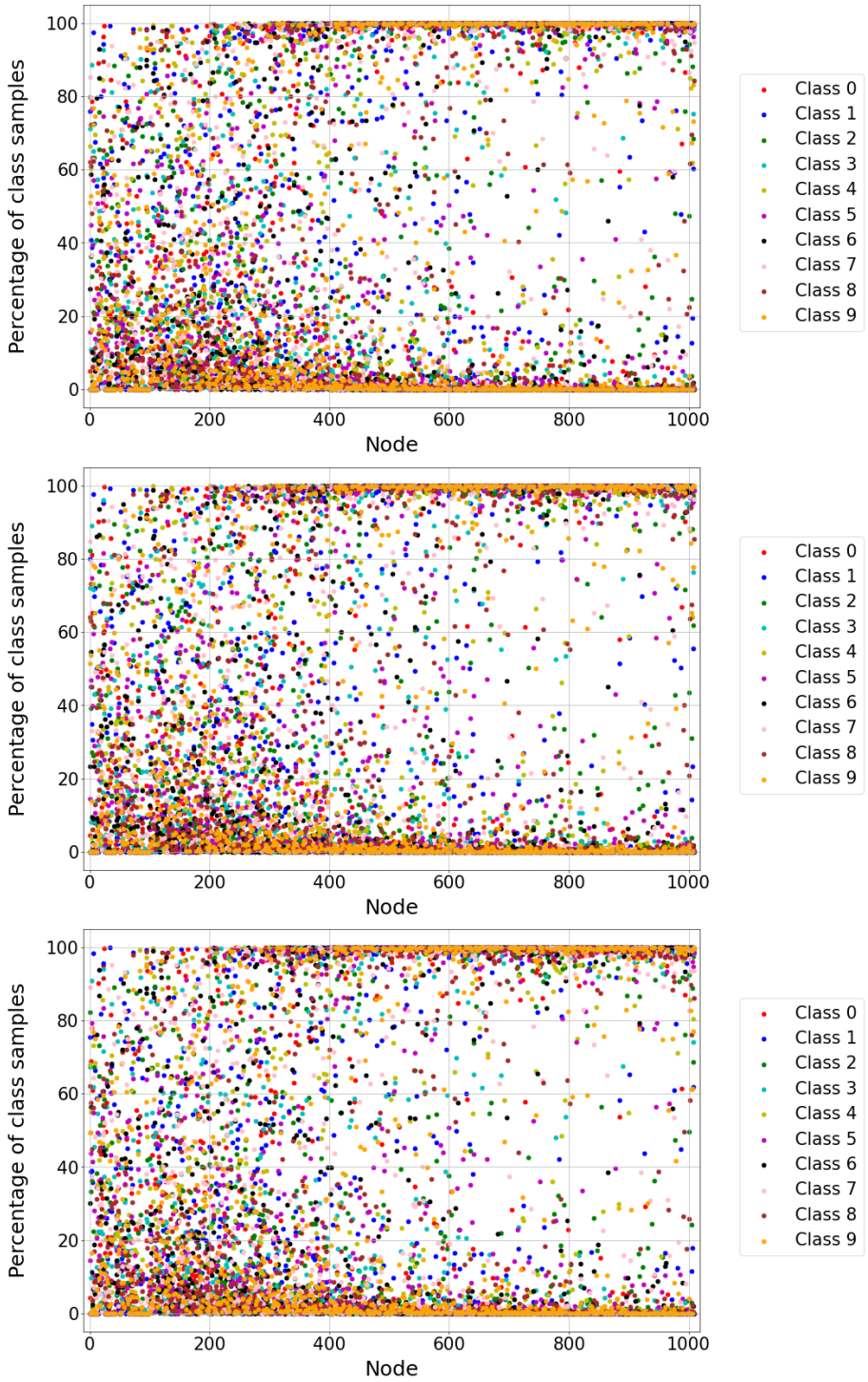


Figure A.3: Percentage of class samples activated per node for every class using the MNIST network (seed 10). From top to bottom, the figures show the results for the training, validation and evaluation sets respectively. Nodes are arranged according to layer, where node 0-99 represent the first layer, node 100-199 represent the second layer, etc.

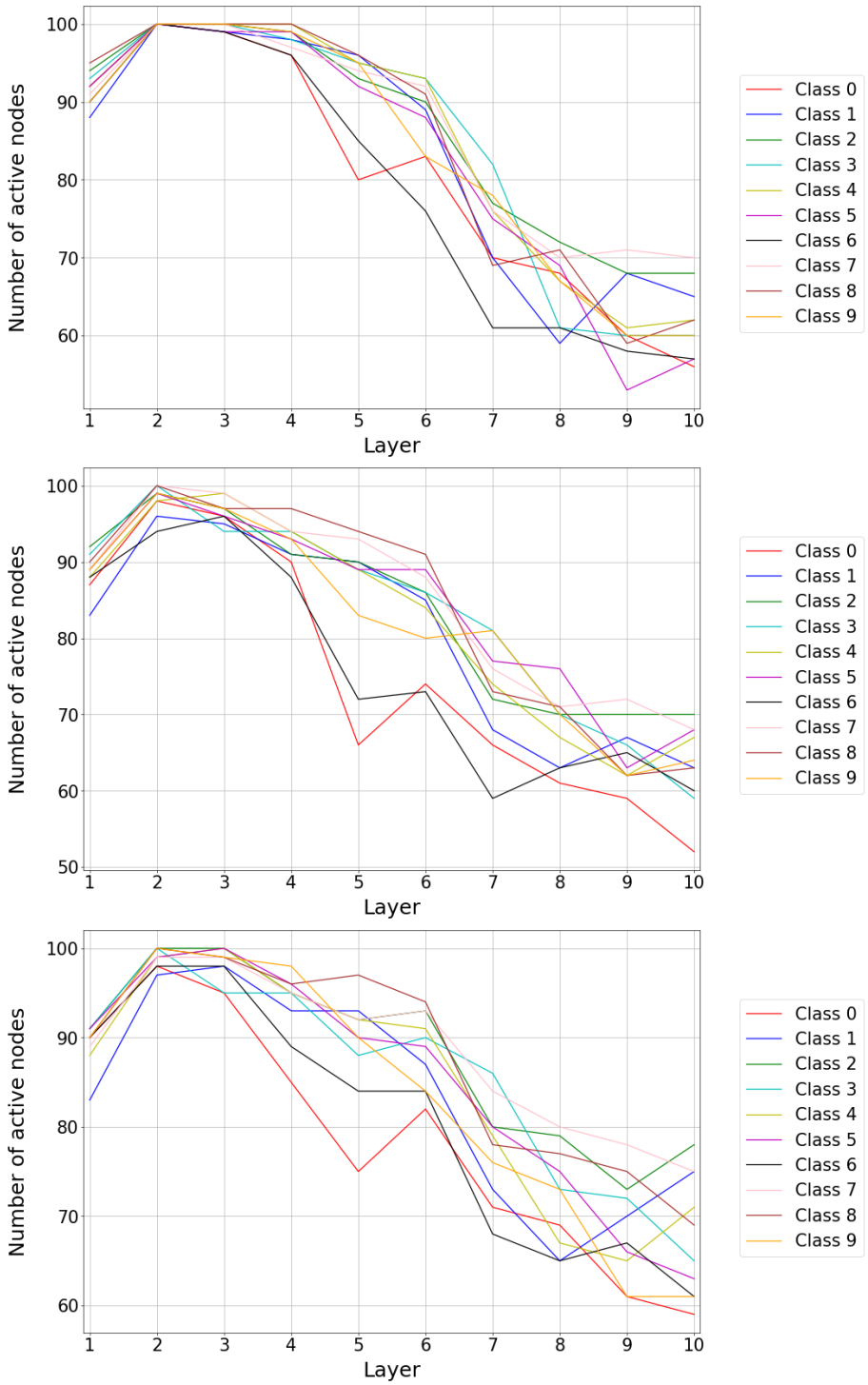


Figure A.4: Number of active nodes per hidden layer for every class using the MNIST network (seed 3). From top to bottom, the figures show the results for the training, validation and evaluation sets respectively.

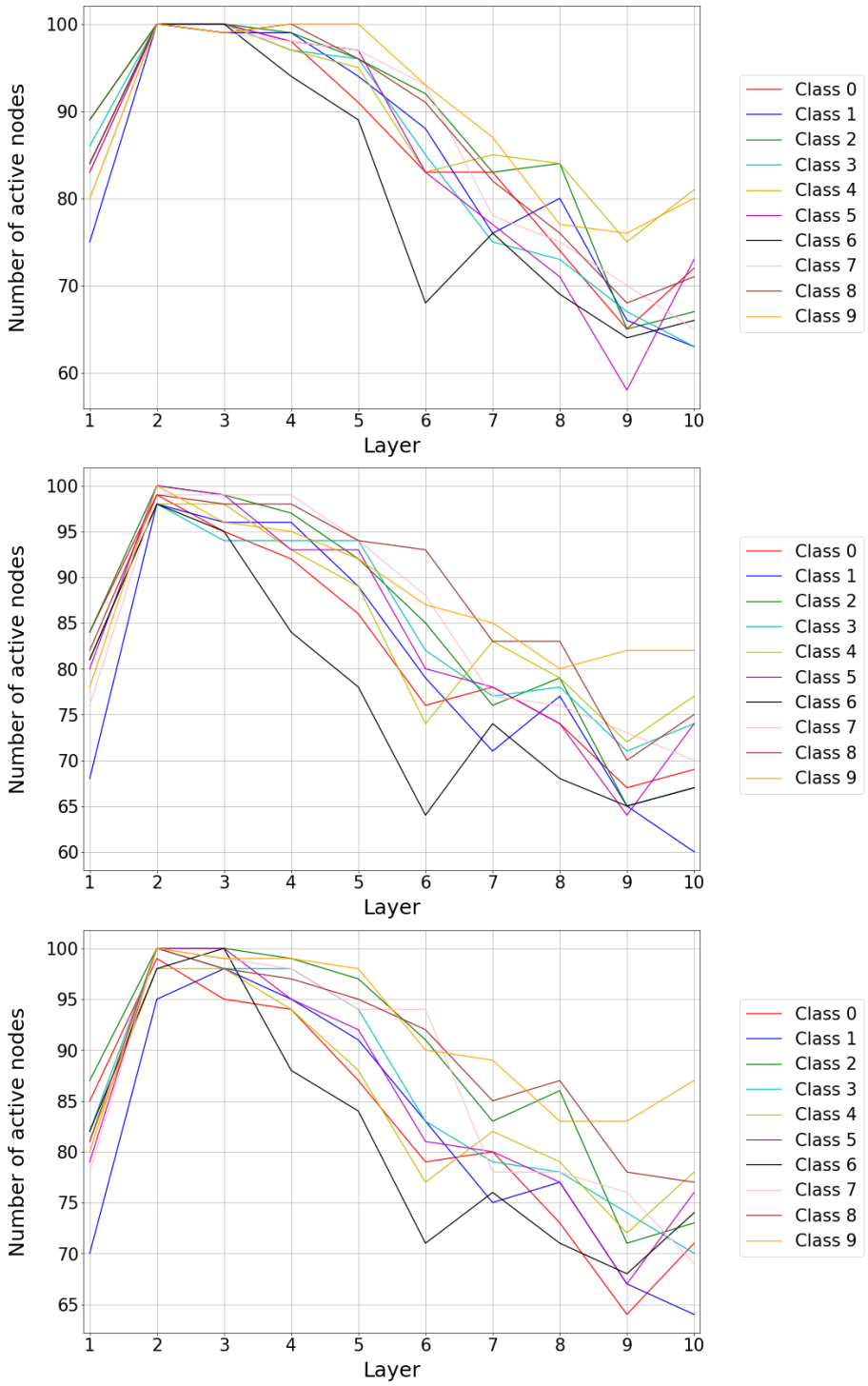


Figure A.5: Number of active nodes per hidden layer for every class using the MNIST network (seed 5). From top to bottom, the figures show the results for the training, validation and evaluation sets respectively.

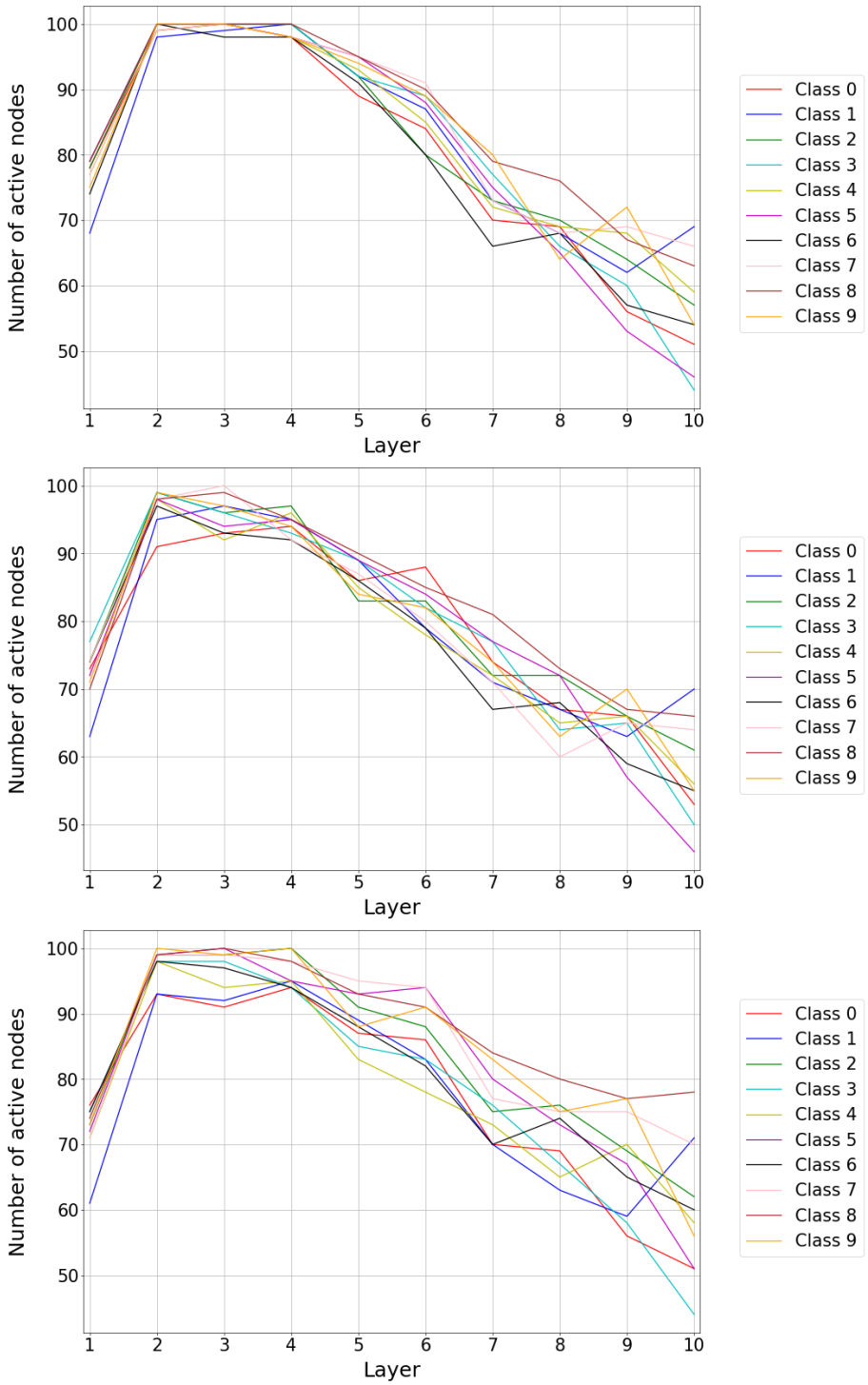


Figure A.6: Number of active nodes per hidden layer for every class using the MNIST network (seed 10). From top to bottom, the figures show the results for the training, validation and evaluation sets respectively.

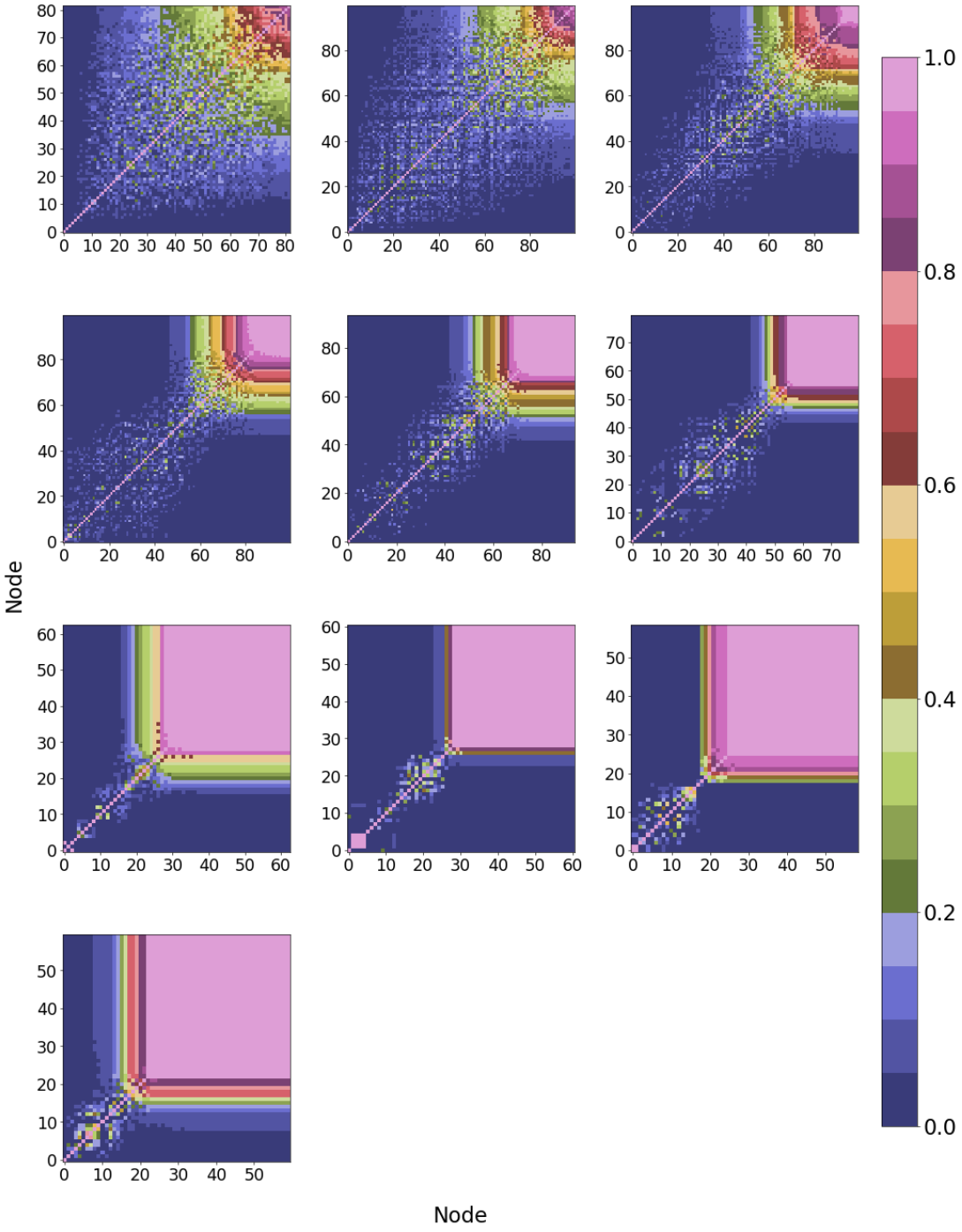


Figure A.7: The Jaccard similarity index between every pair of active nodes for all layers of MNIST class three, where all nodes are sorted in ascending order with regards to their sample set size. Each matrix represents a layer in the network from shallow to deep and is read from left to right, top to bottom. Note that each matrix is mirrored on the $x = y$ axis.

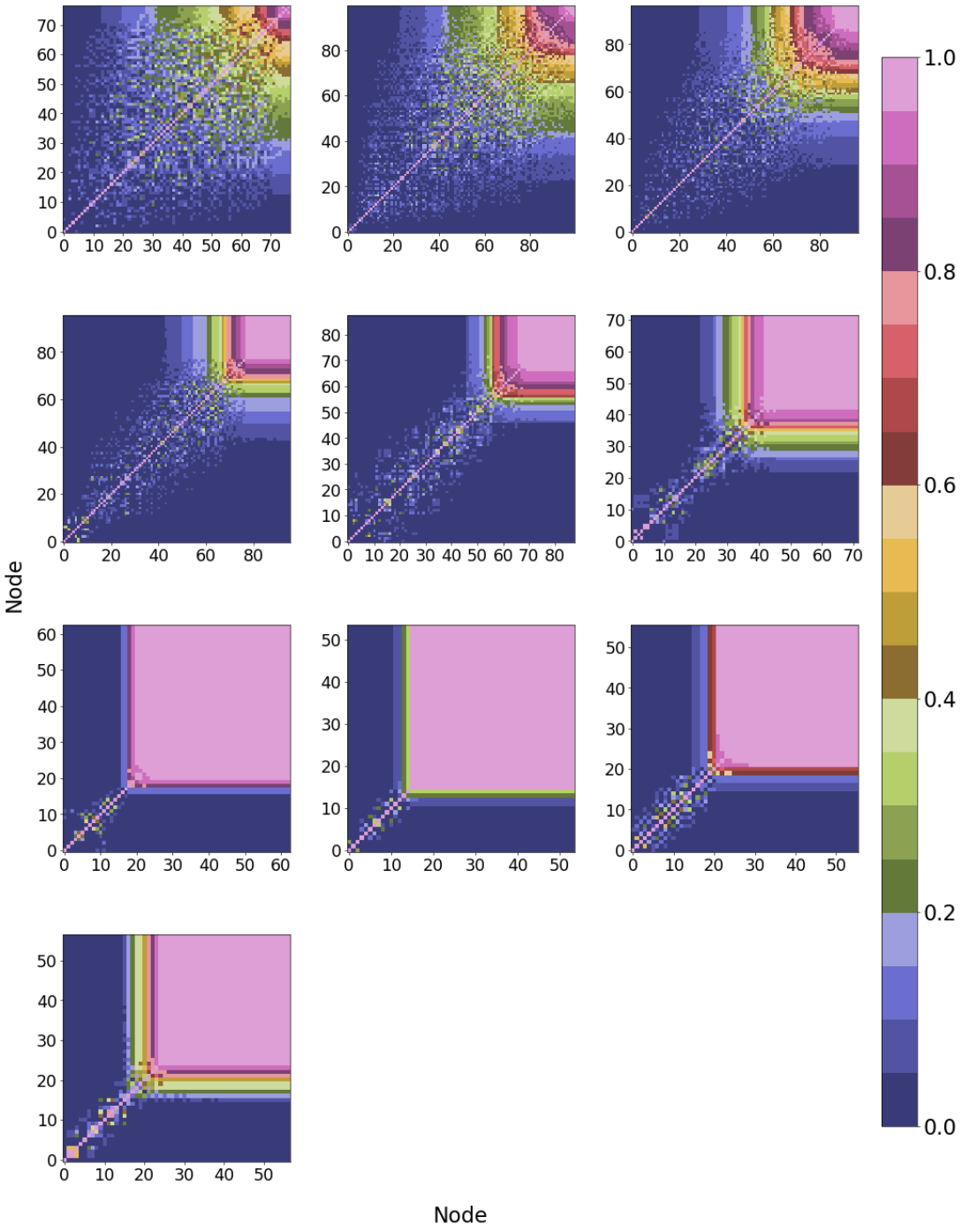


Figure A.8: The Jaccard similarity index between every pair of active nodes for all layers of MNIST class six, where all nodes are sorted in ascending order with regards to their sample set size. Each matrix represents a layer in the network from shallow to deep and is read from left to right, top to bottom. Note that each matrix is mirrored on the $x = y$ axis.

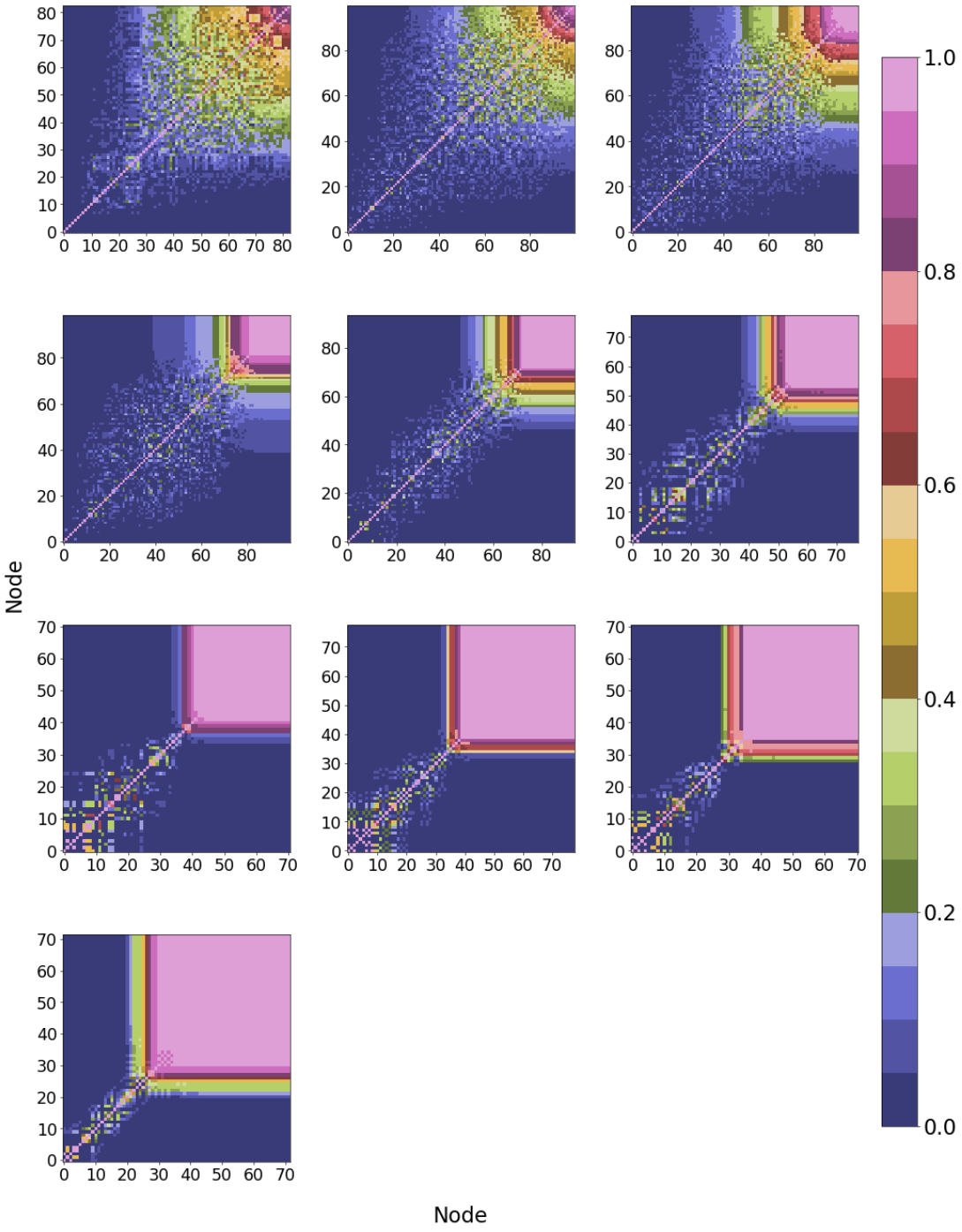


Figure A.9: The Jaccard similarity index between every pair of active nodes for all layers of MNIST class nine, where all nodes are sorted in ascending order with regards to their sample set size. Each matrix represents a layer in the network from shallow to deep and is read from left to right, top to bottom. Note that each matrix is mirrored on the $x = y$ axis.

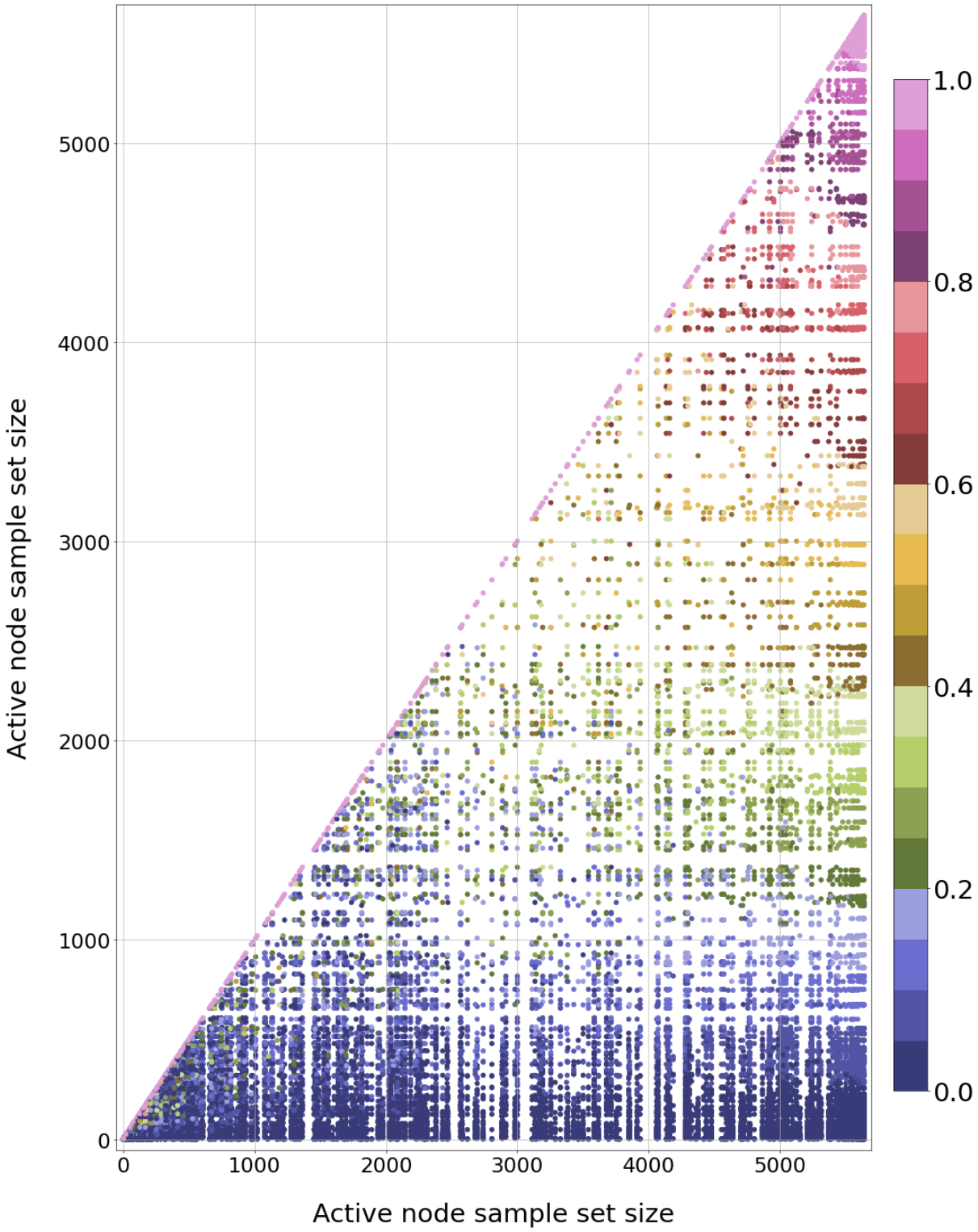


Figure A.10: The Jaccard similarity index of every active node pair for MNIST class three. Node pairs are sorted in ascending order according to the size of their respective sample sets.

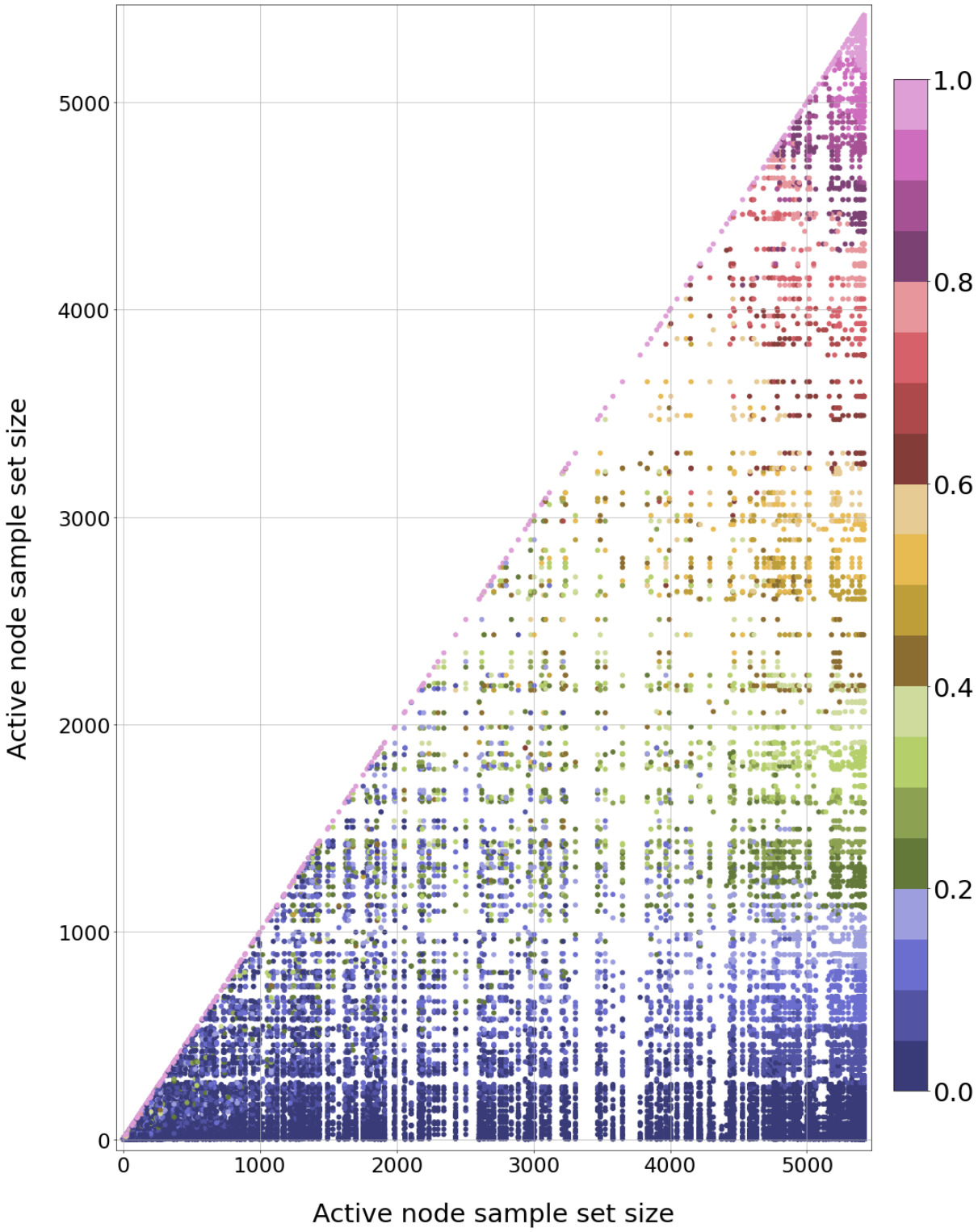


Figure A.11: The Jaccard similarity index of every active node pair for MNIST class six. Node pairs are sorted in ascending order according to the size of their respective sample sets.

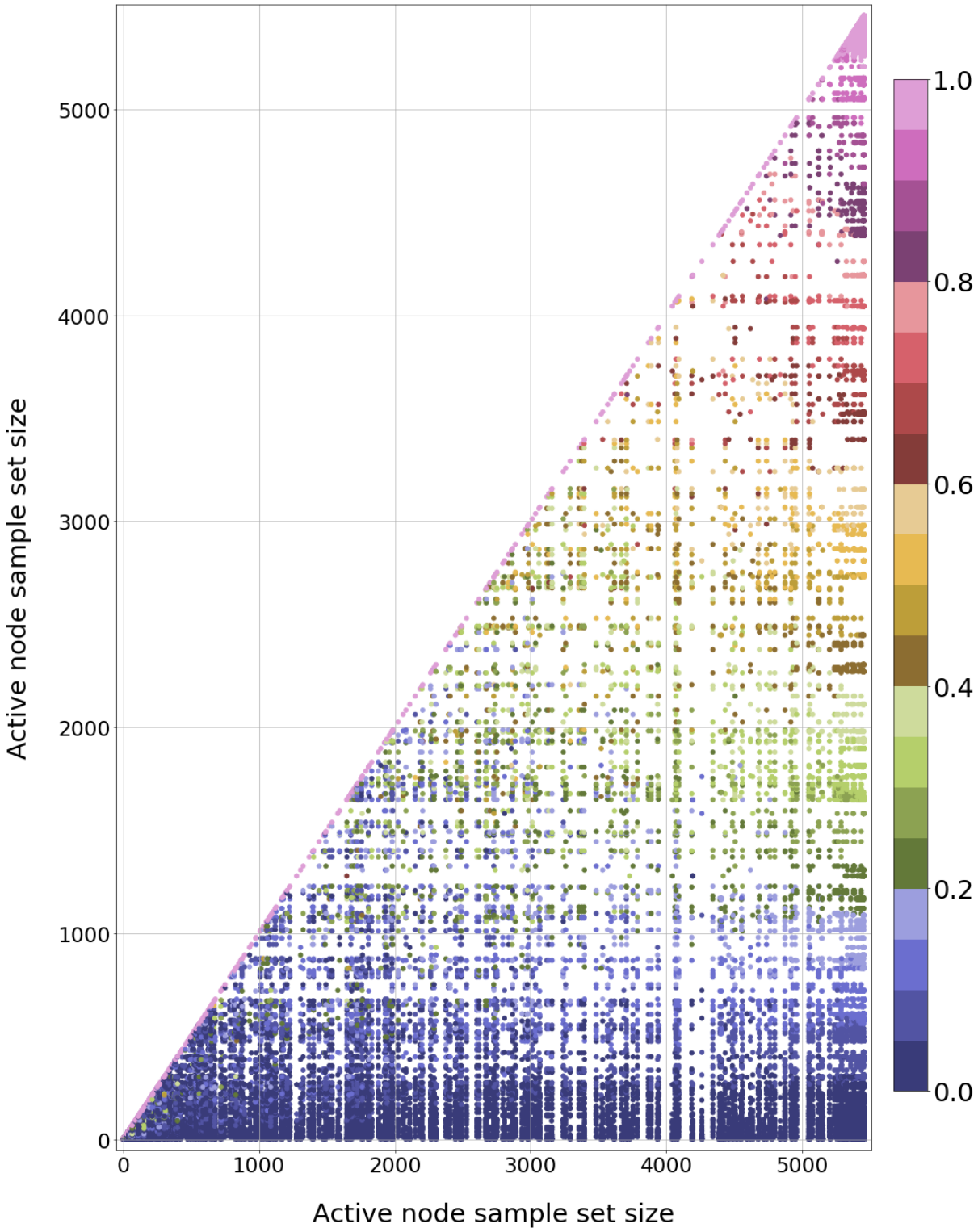


Figure A.12: The Jaccard similarity index of every active node pair for class MNIST nine. Node pairs are sorted in ascending order according to the size of their respective sample sets.

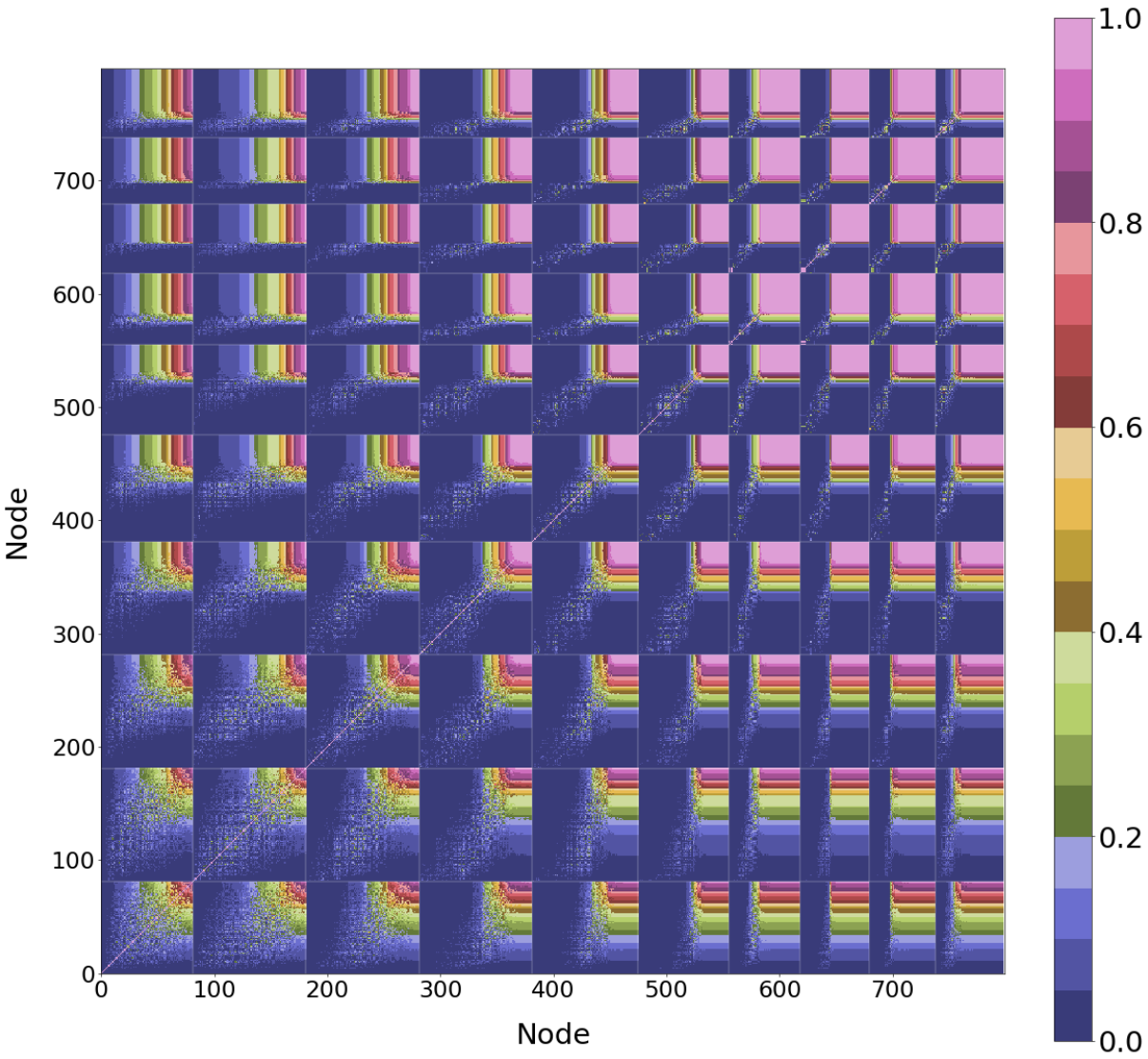


Figure A.13: Jaccard similarity matrix for every possible node pair in the network for MNIST class three. Nodes are ordered first according to their layer (first to last) and secondly according to their sample set size (small to large) for each layer.

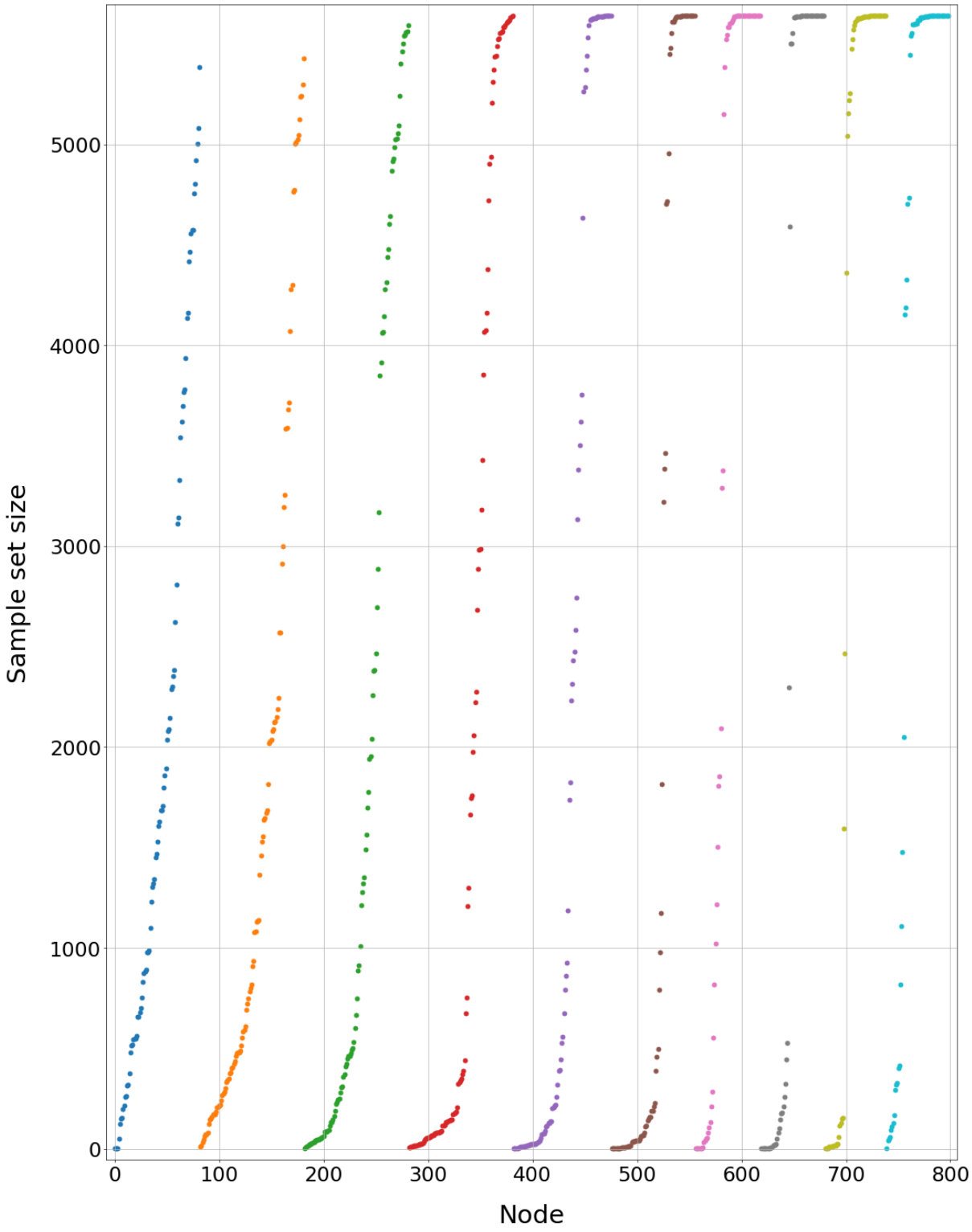


Figure A.14: Sample set size for every node in the network for MNIST class three. Nodes are ordered in the same way as in Figure A.13.

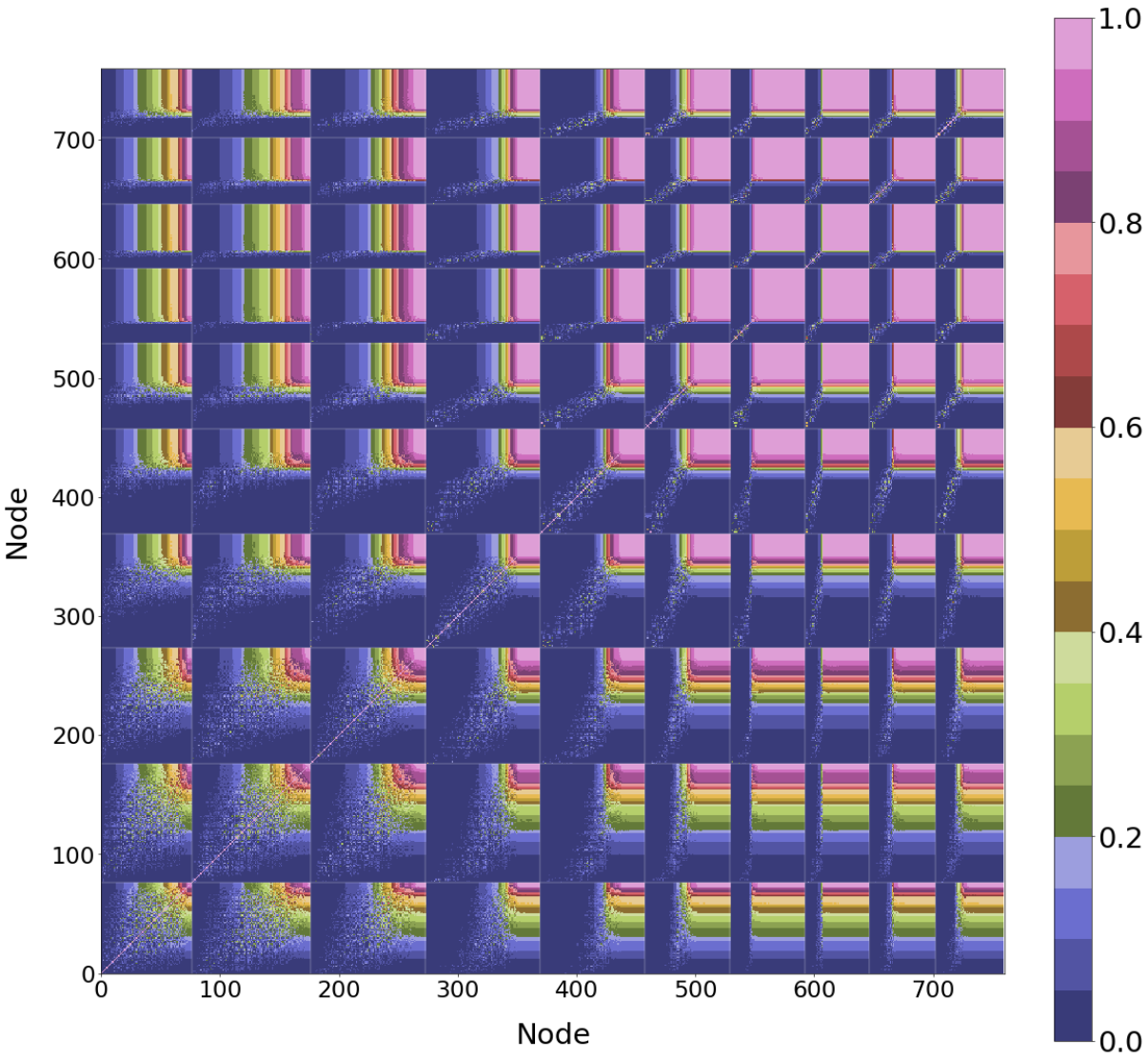


Figure A.15: Jaccard similarity matrix for every possible node pair in the network for MNIST class six. Nodes are ordered first according to their layer (first to last) and secondly according to their sample set size (small to large) for each layer.

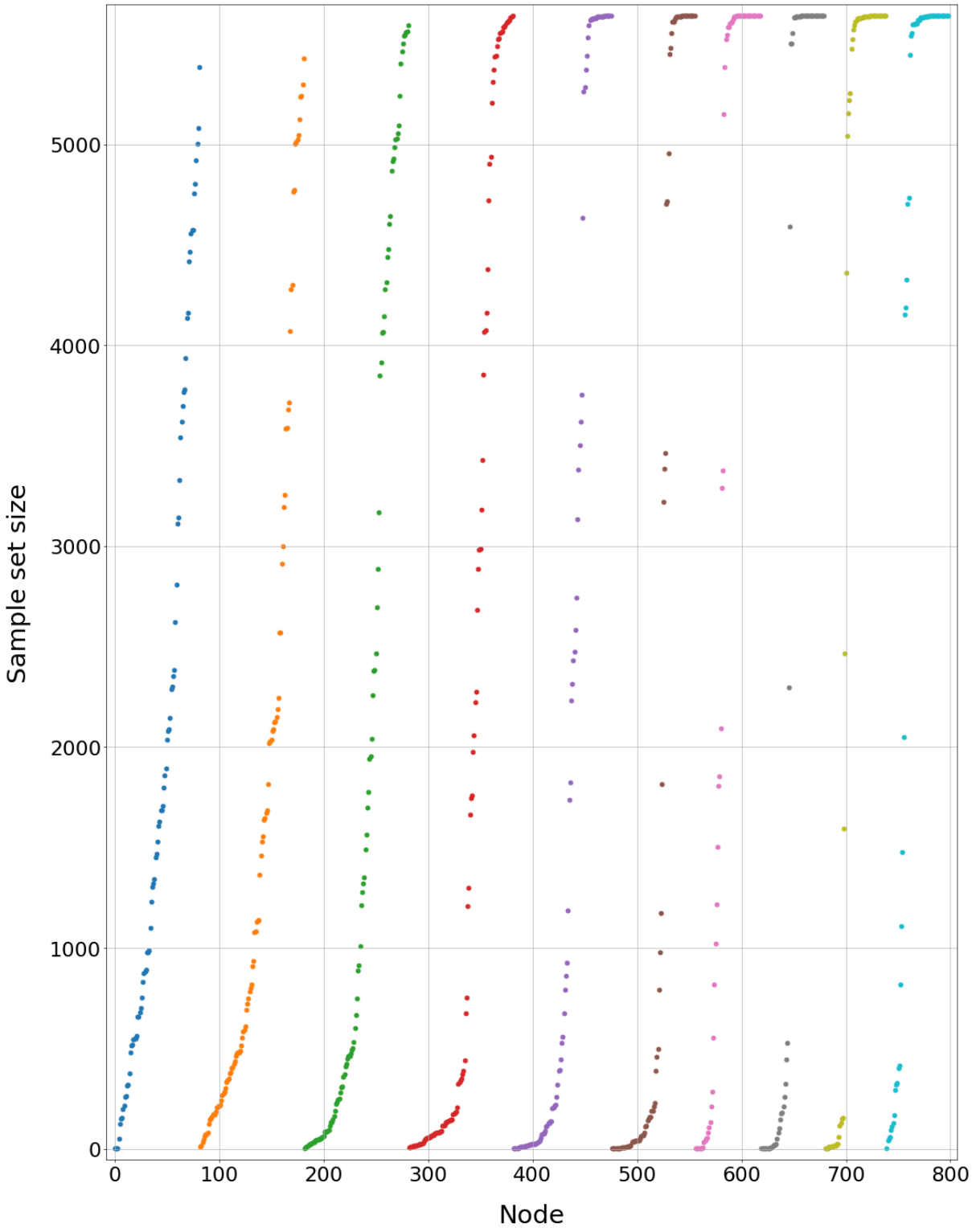


Figure A.16: Sample set size for every node in the network for MNIST class six. Nodes are ordered in the same way as in Figure A.15.

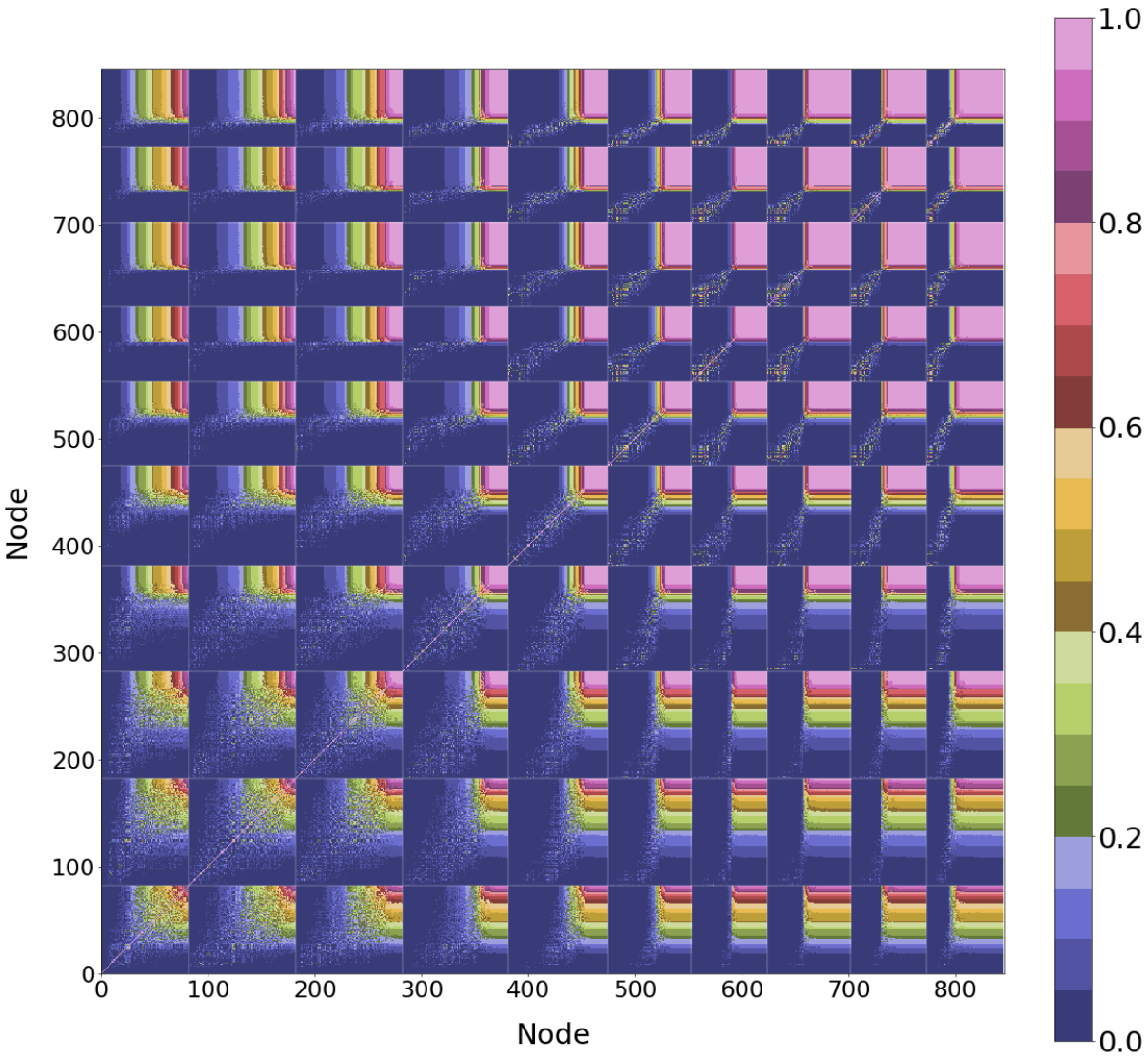


Figure A.17: Jaccard similarity matrix for every possible node pair in the network for MNIST class nine. Nodes are ordered first according to their layer (first to last) and secondly according to their sample set size (small to large) for each layer.

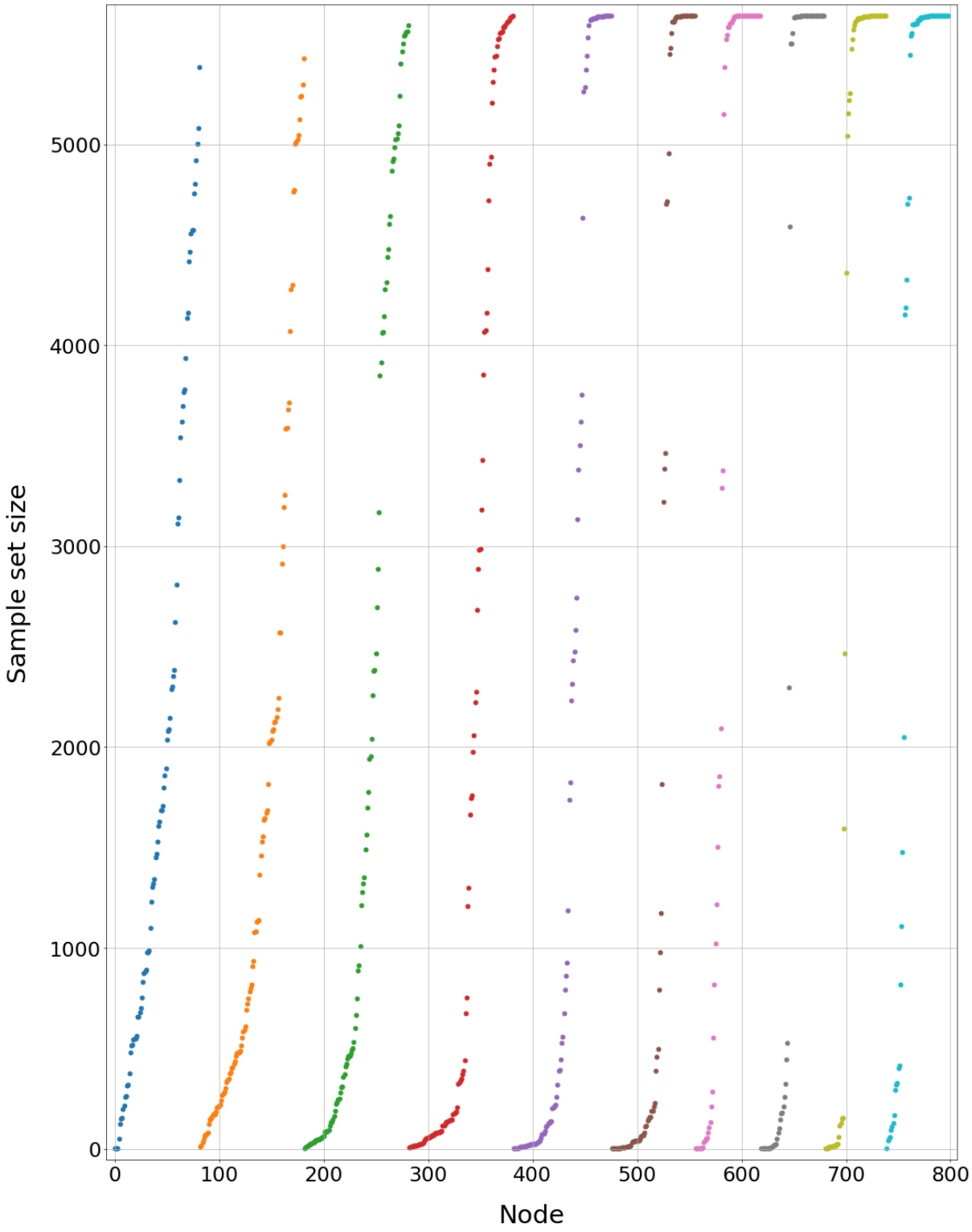


Figure A.18: Sample set size for every node in the network for MNIST class nine. Nodes are ordered in the same way as in Figure A.17.

Appendix B

Supplemental content: Chapter 5

In this chapter we provide supplementary results for Chapter 5.

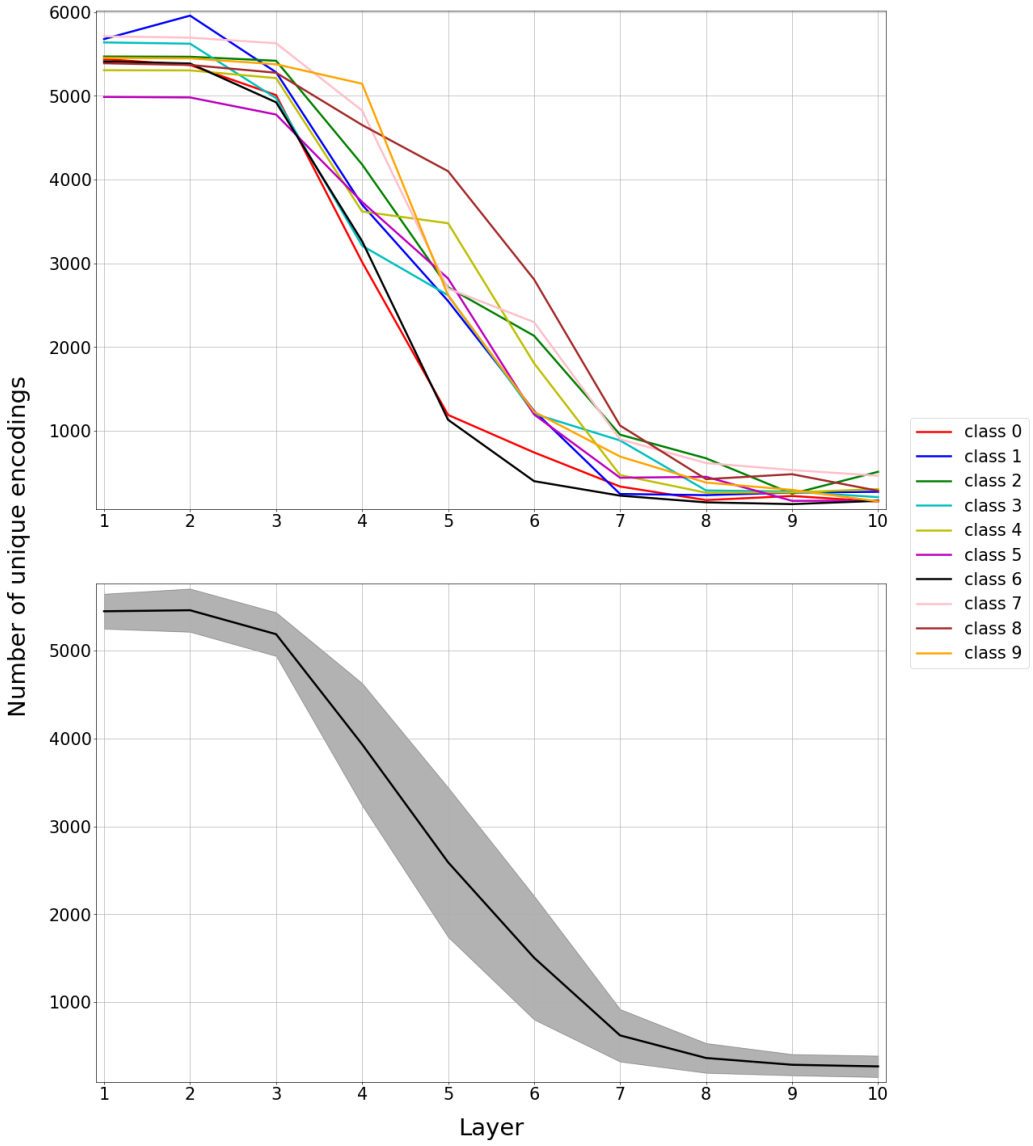


Figure B.1: Number of unique encodings per hidden layer for every class (top) and averaged over all classes (bottom) with the shaded regions indicating the standard deviation between all classes. These are the results for seed three.

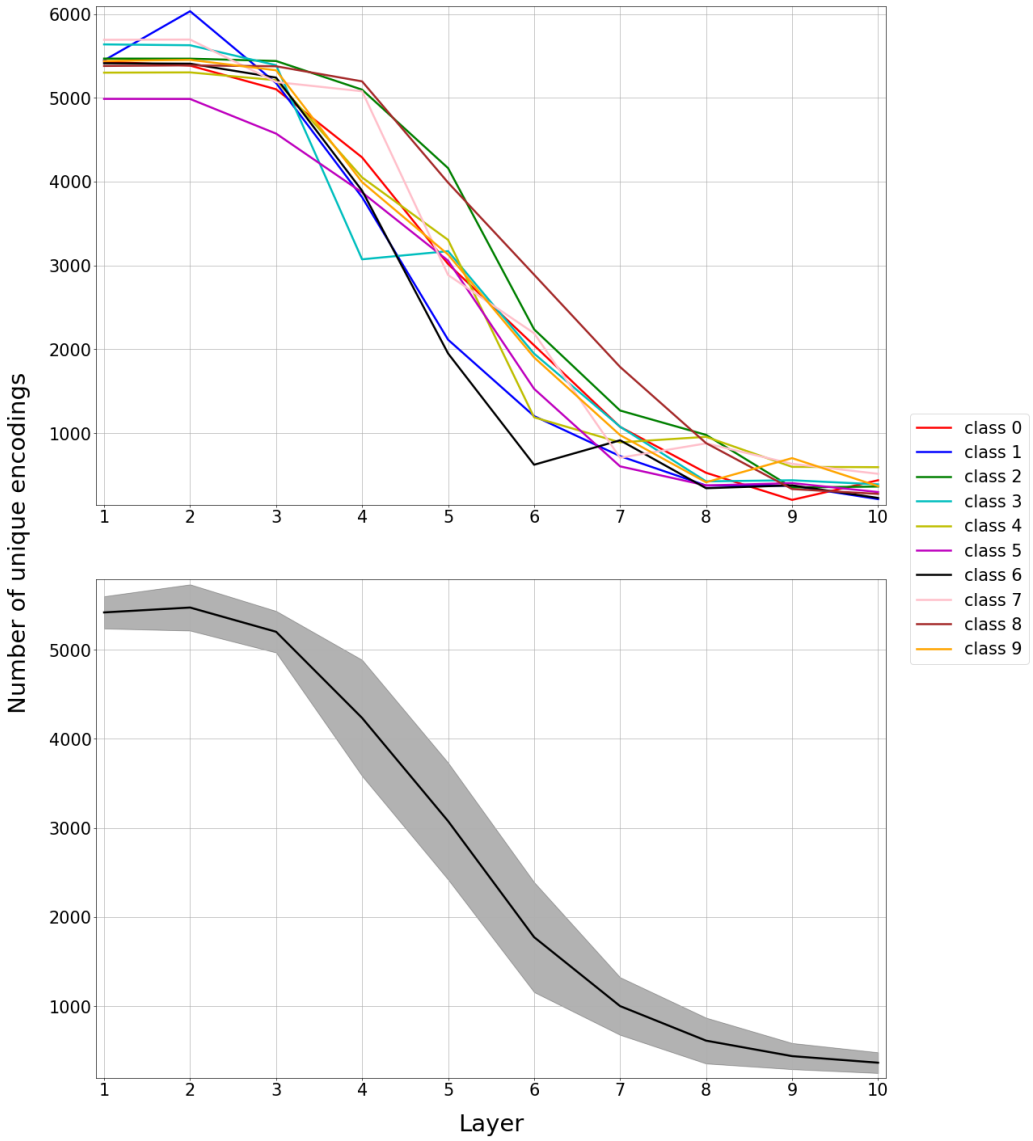


Figure B.2: Number of unique encodings per hidden layer for every class (top) and averaged over all classes (bottom) with the shaded regions indicating the standard deviation between all classes. These are the results for seed six.

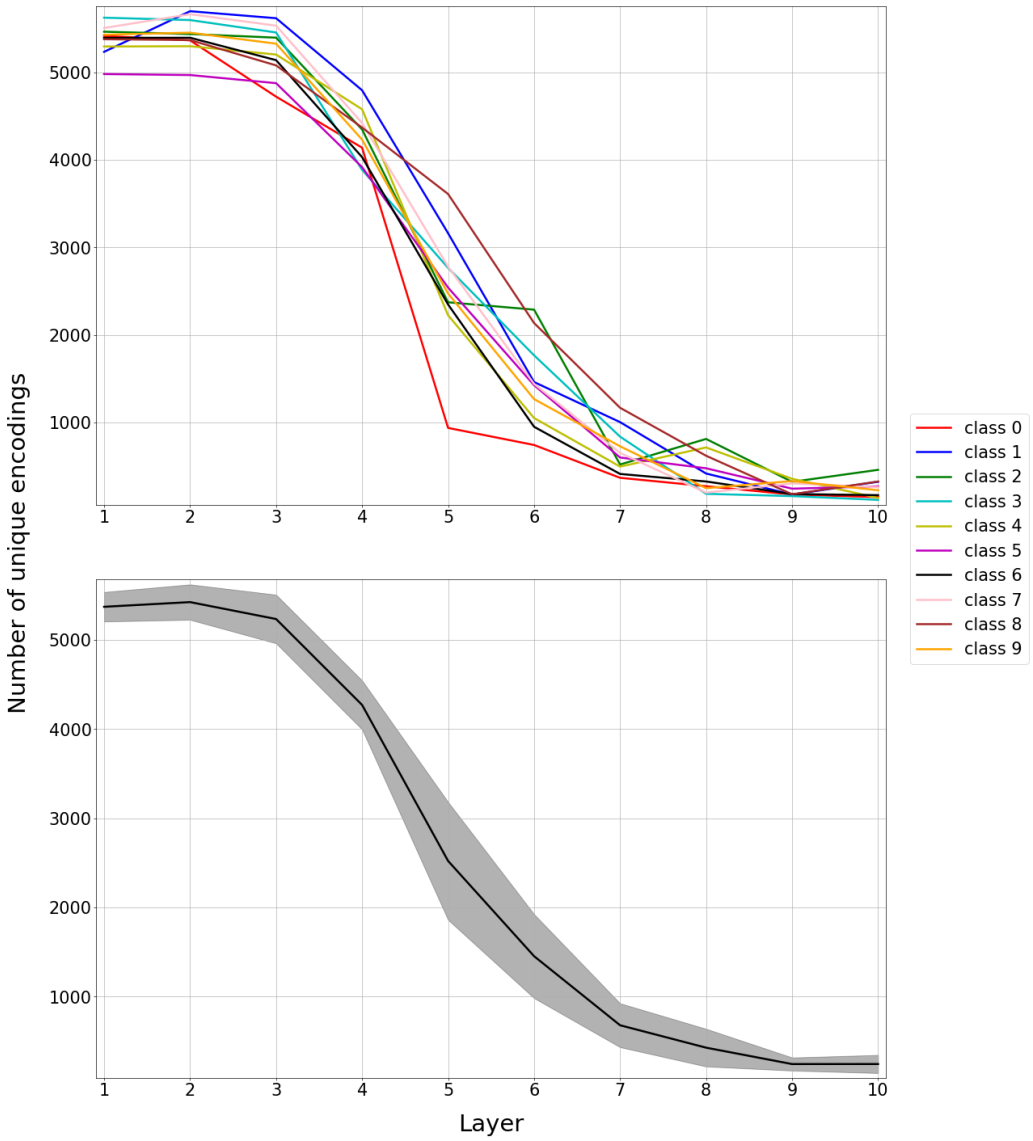


Figure B.3: Number of unique encodings per hidden layer for every class (top) and averaged over all classes (bottom) with the shaded regions indicating the standard deviation between all classes. These are the results for seed nine.

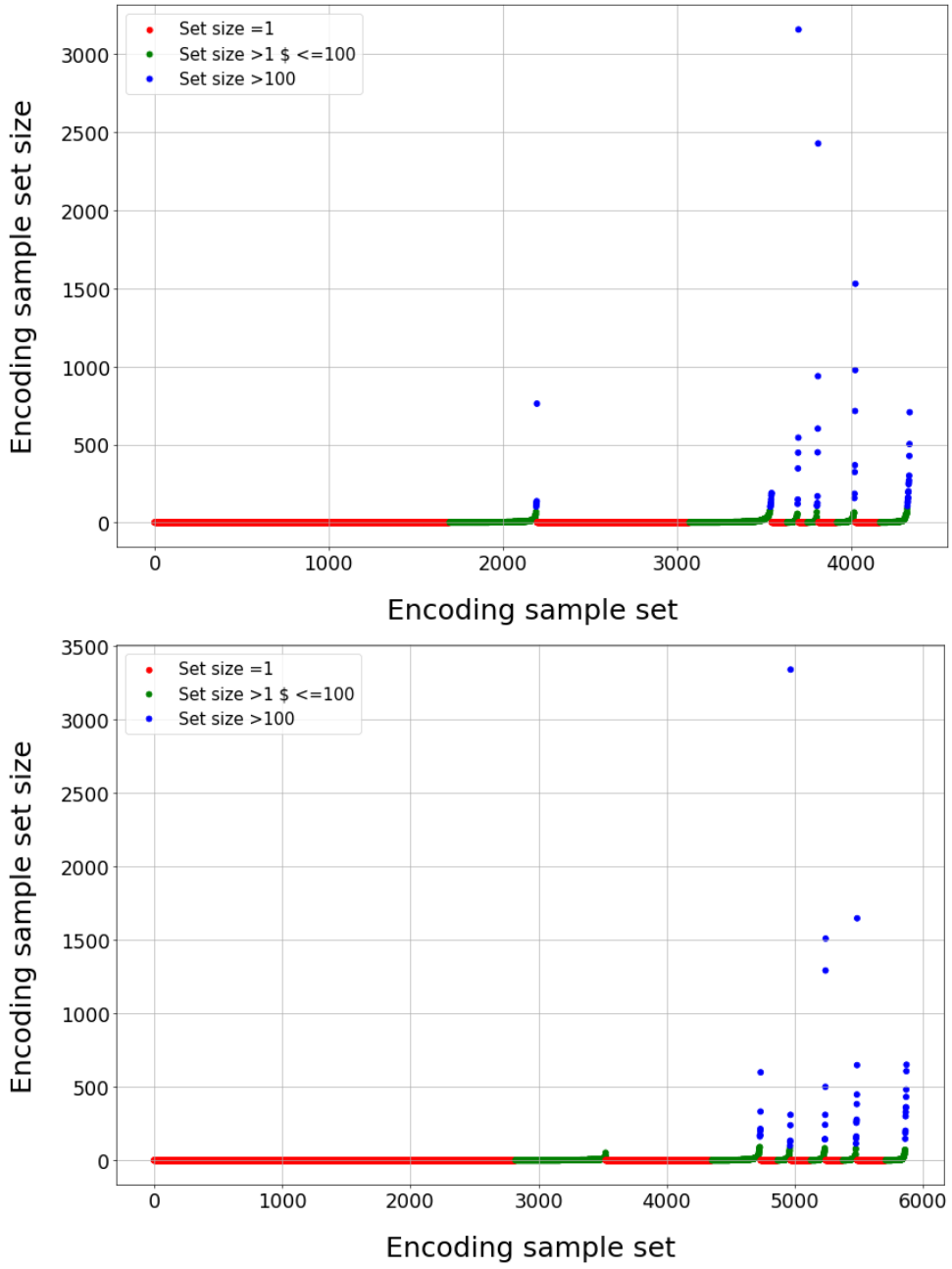


Figure B.4: Encoding sample set sizes for the MNIST classes six (top-image) and nine (bottom-image), between the hidden layers five to ten. Encoding sample sets are first ordered by layer and then by sample set size in ascending order. Encoding sample sets are colour coded as follows: red indicates a size of one, green indicates a size of greater than one but less or equal to 100 and blue indicates a size of greater than 100.

Table B.1: Encoding sample set sizes for the different encoding sample set groups as shown in Figure B.4.

Layer	**Num. of red dots**	**Num. of green dots**	**Num. of blue dots**
		Class 0	
5	1696	497	5
6	877	463	9
7	88	60	6
8	52	53	8
9	108	99	7
10	142	156	14
		Class 3	
5	2822	782	0
6	830	373	7
7	133	95	6
8	158	100	10
9	97	90	8
10	224	150	11

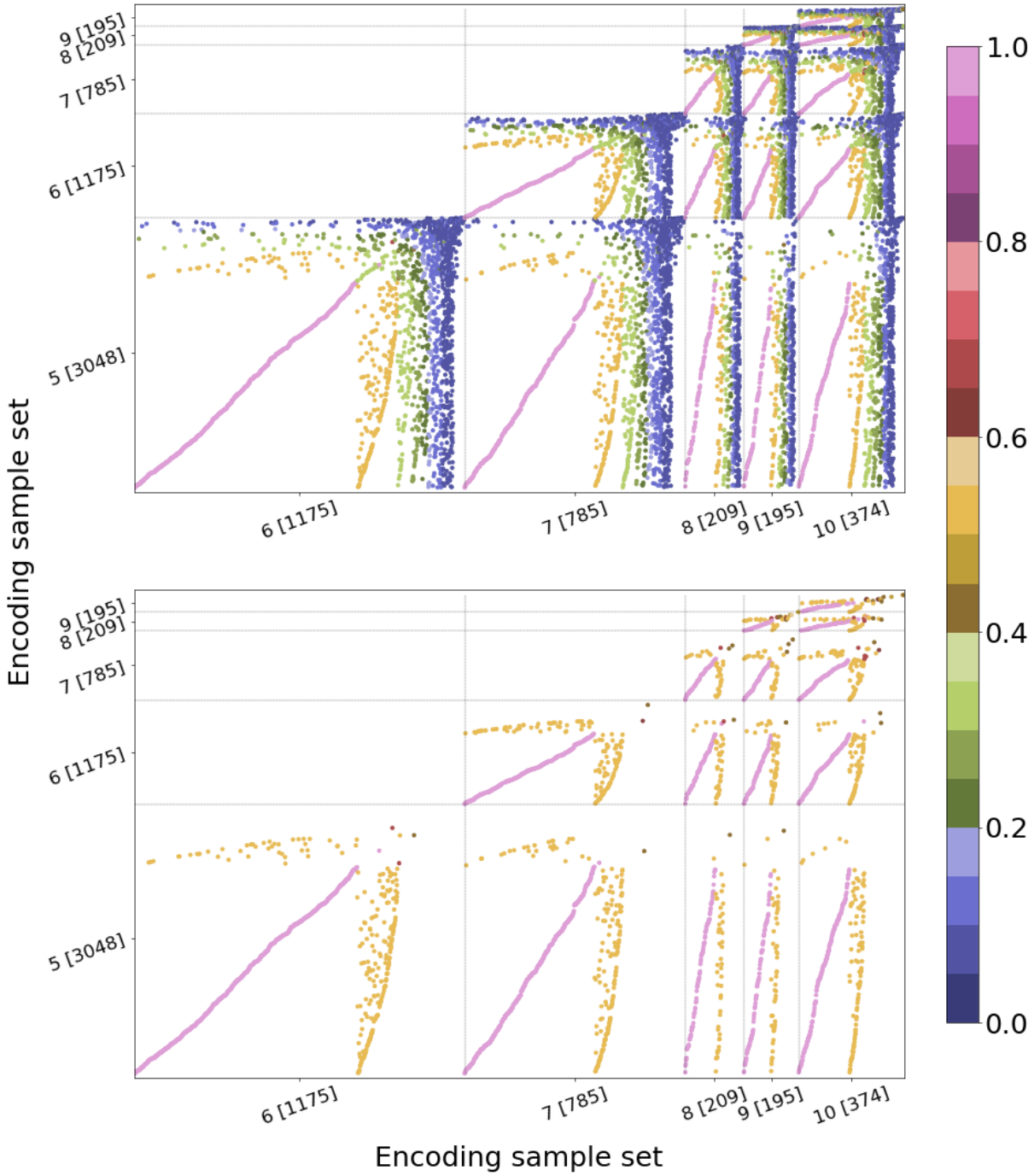


Figure B.5: Jaccard similarity between every pair of encoding sample sets in the deeper layers of the network for MNIST class three. The plots show the same graph, but only for similarities higher than 0.05 (top) and 0.6 (bottom) respectively. Each tick represents the layer and the number of encoding sample sets they contain in brackets, while the black lines are used to separate layers. Encoding sample sets are first ordered by layer and then by sample set size, both in ascending order.

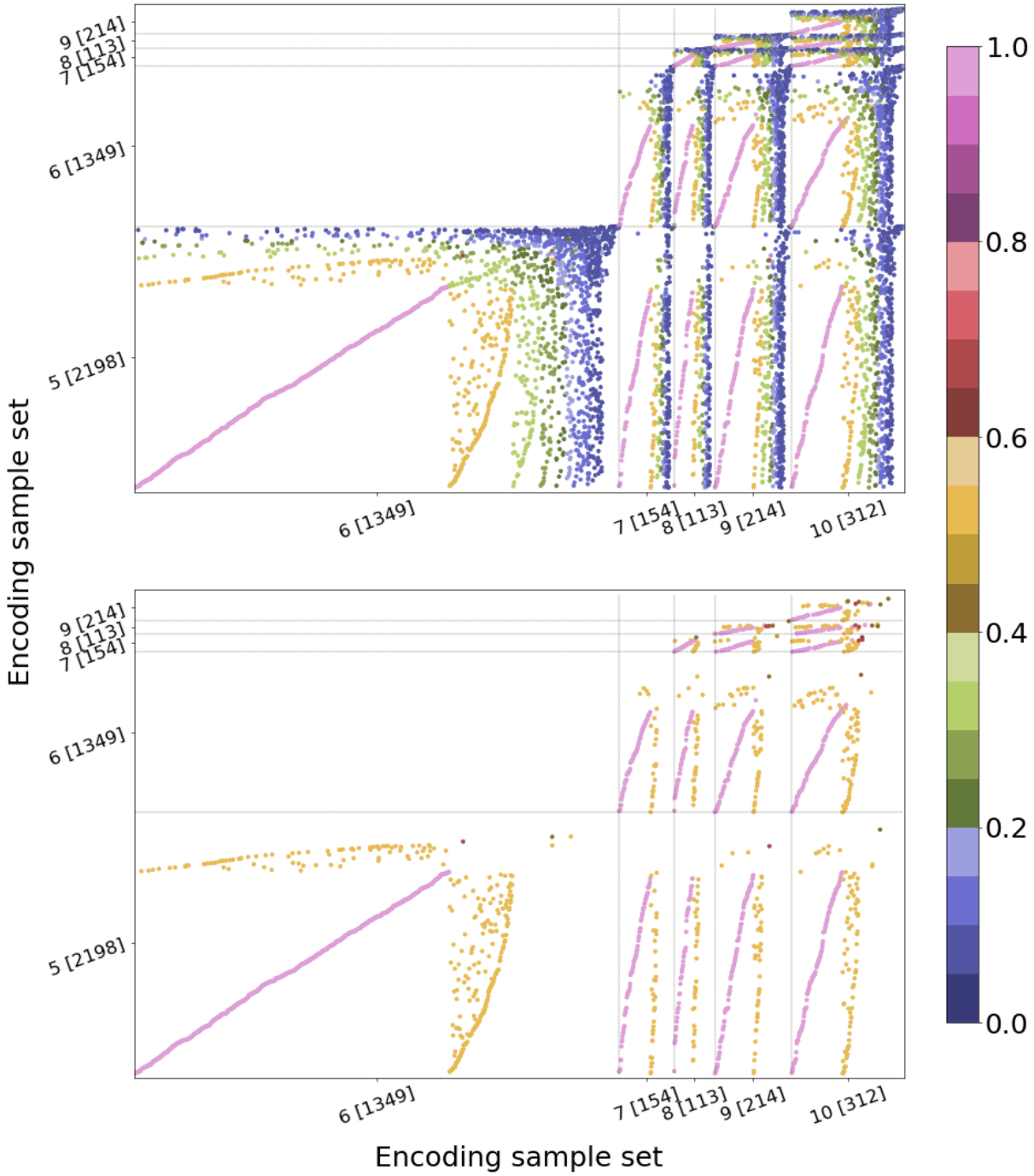


Figure B.6: Jaccard similarity between every pair of encoding sample sets in the deeper layers of the network for MNIST class six. The plots show the same graph, but only for similarities higher than 0.05 (top) and 0.6 (bottom) respectively. Each tick represents the layer and the number of encoding sample sets they contain in brackets, while the black lines are used to separate layers. Encoding sample sets are first ordered by layer and then by sample set size, both in ascending order.

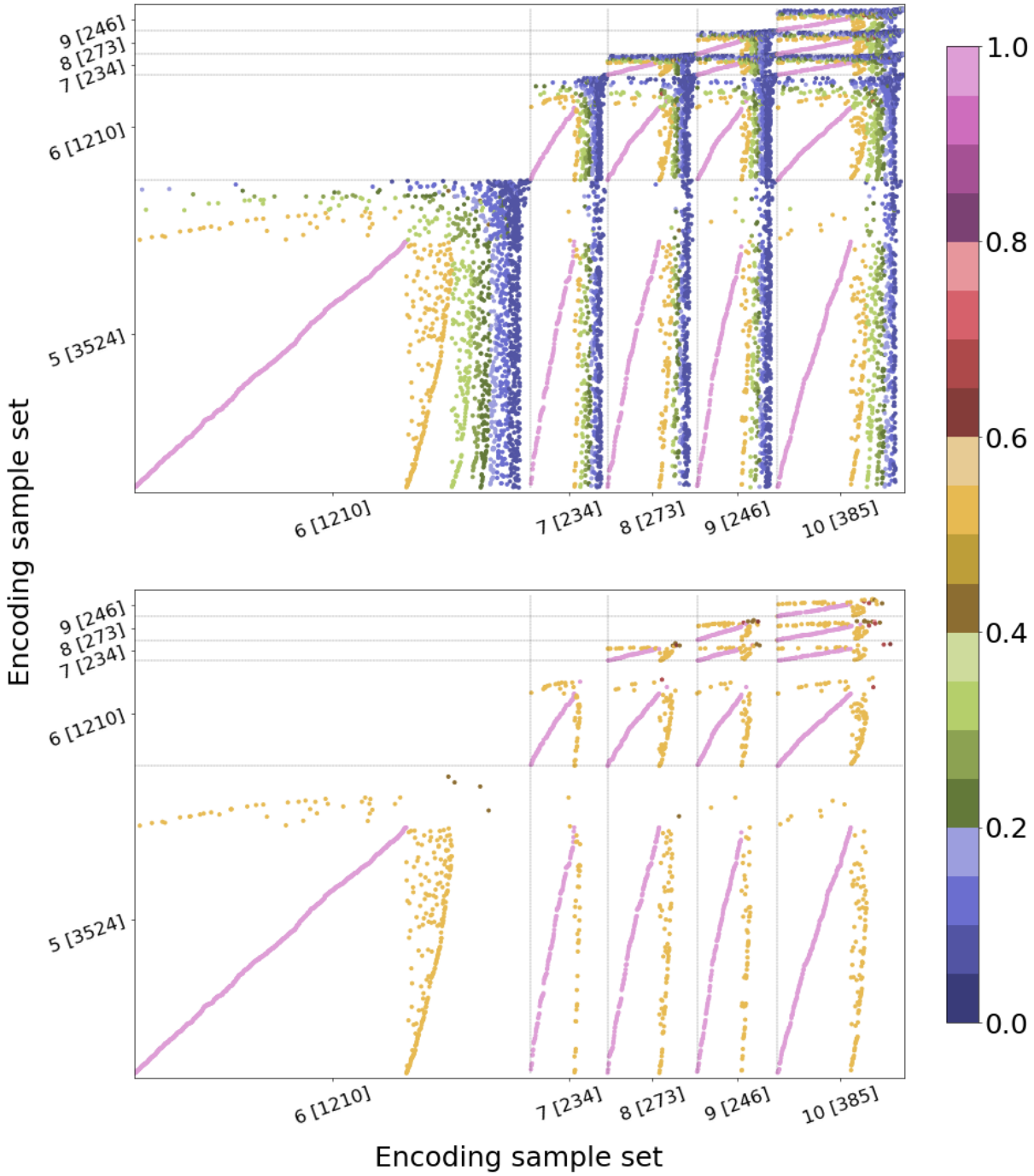


Figure B.7: Jaccard similarity between every pair of encoding sample sets in the deeper layers of the network for MNIST class nine. The plots show the same graph, but only for similarities higher than 0.05 (top) and 0.6 (bottom) respectively. Each tick represents the layer and the number of encoding sample sets they contain in brackets, while the black lines are used to separate layers. Encoding sample sets are first ordered by layer and then by sample set size, both in ascending order.

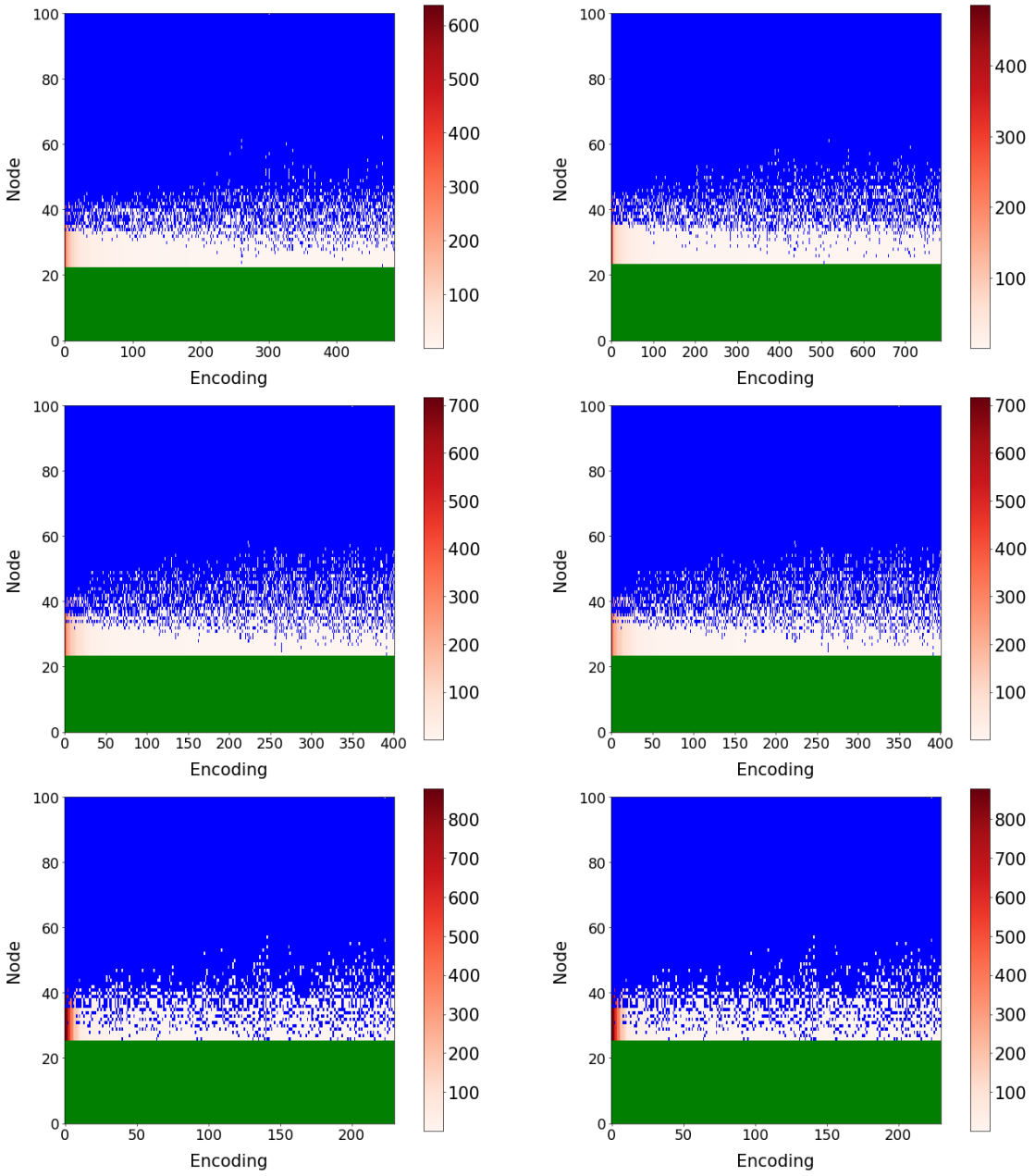


Figure B.8: Visualization of the sample encodings for every encoding sample set. The results are shown for MNIST class zero (left) and class three (right). From top to bottom the results are given for hidden layers seven, eight and nine respectively. Nodes are ordered according to largest number of encoding activations and encodings are ordered according to their sample set size from large to small. Positive node activations are coloured according to the size of encoding sample set, while no activations are blue, and core nodes are shown in green.

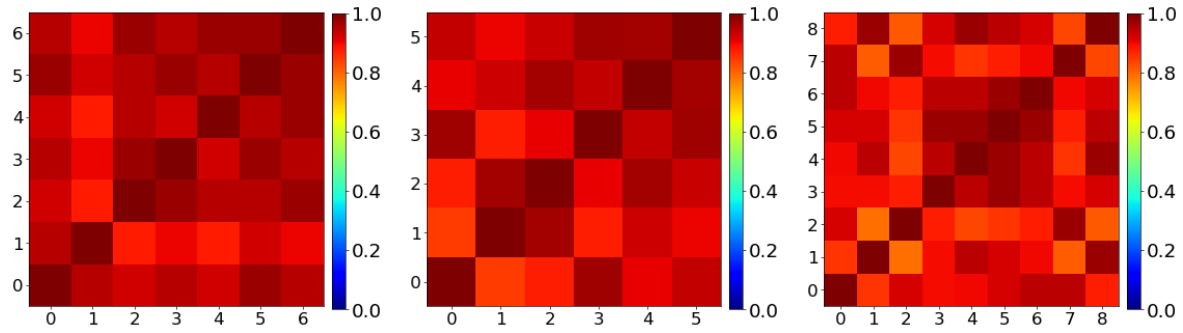


Figure B.9: The Jaccard similarity index of the active nodes between each encoding pair for all class zero (left), three (middle) and six (right) encoding sample sets in layer six of the MNIST network, where all encoding sample sets have a size greater than 100.

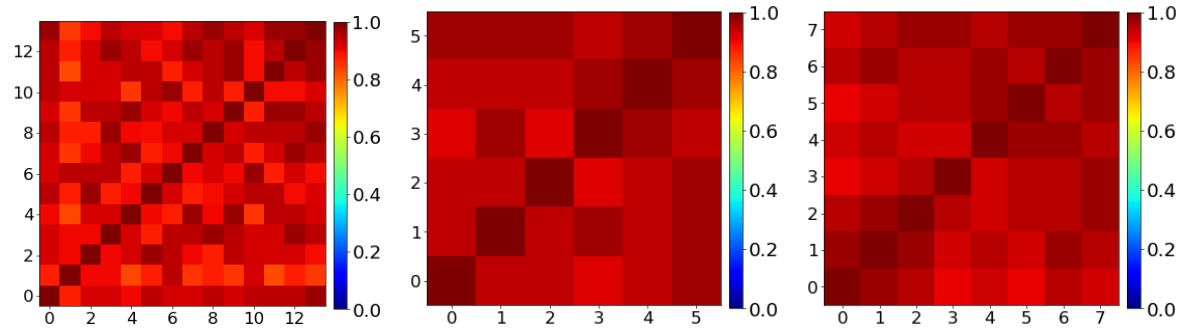


Figure B.10: The Jaccard similarity index of the active nodes between each encoding pair for all class zero (left), three (middle) and six (right) encoding sample sets in layer eight of the MNIST network, where all encoding sample sets have a size greater than 100.

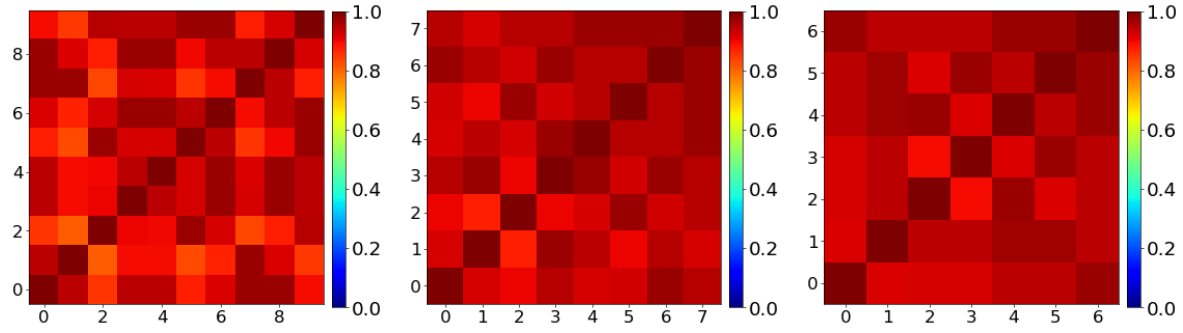


Figure B.11: The Jaccard similarity index of the active nodes between each encoding pair for all class zero (left), three (middle) and six (right) encoding sample sets in layer nine of the MNIST network, where all encoding sample sets have a size greater than 100.

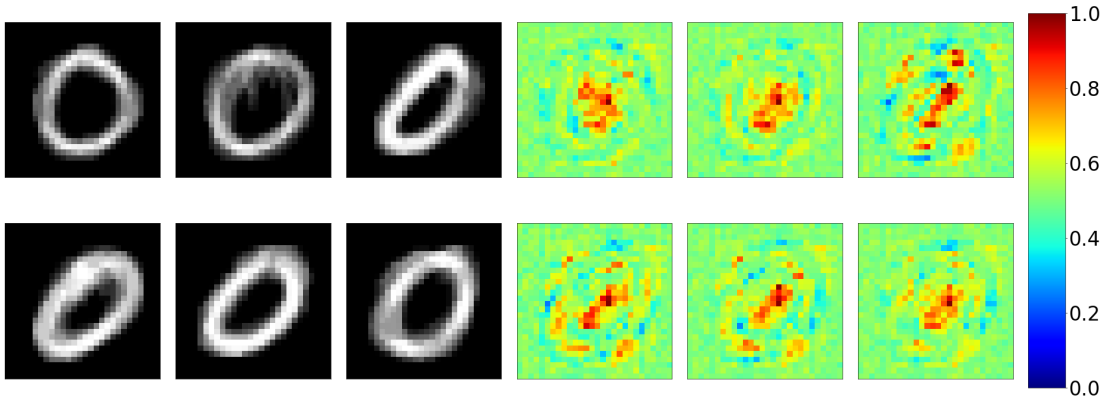


Figure B.12: Visual depiction of between the averaged input features (left) and LRP interpretations (right) for all encoding sample sets in layer six for MNIST class zero with a sample set size greater than 100.

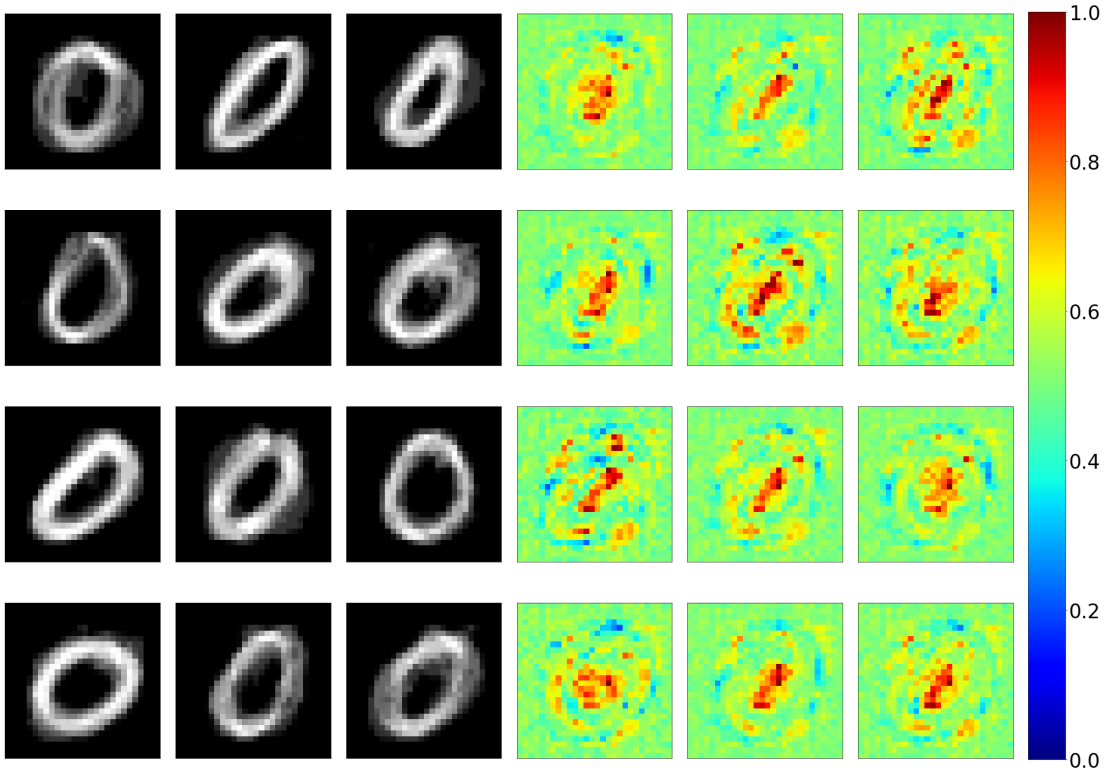


Figure B.13: Visual depiction of between the averaged input features (left) and LRP interpretations (right) for all encoding sample sets in layer eight for MNIST class zero with a sample set size greater than 100.

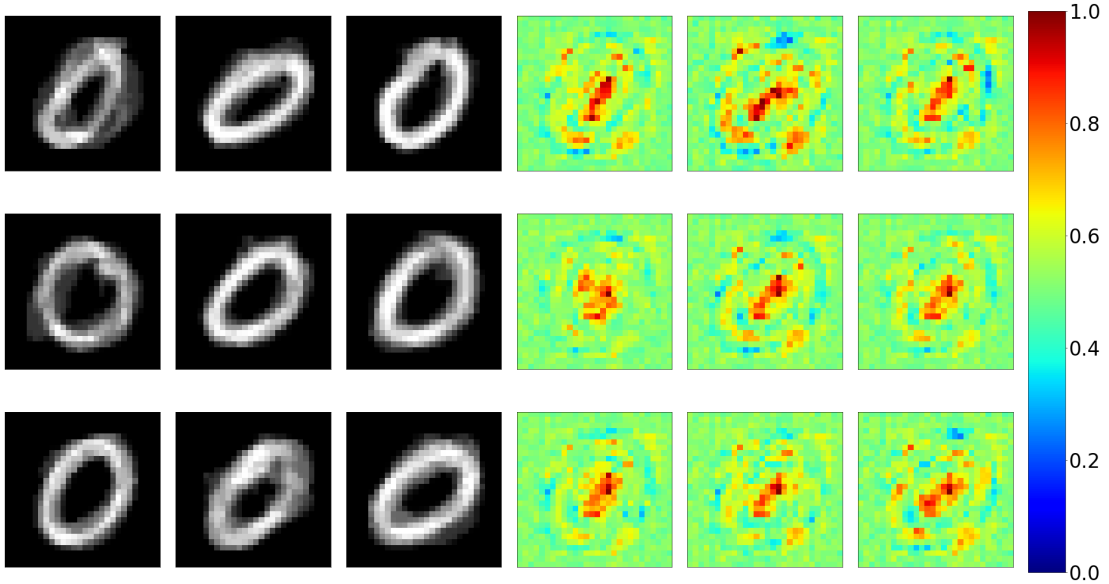


Figure B.14: Visual depiction of between the averaged input features (left) and LRP interpretations (right) for all encoding sample sets in layer nine for MNIST class zero with a sample set size greater than 100.

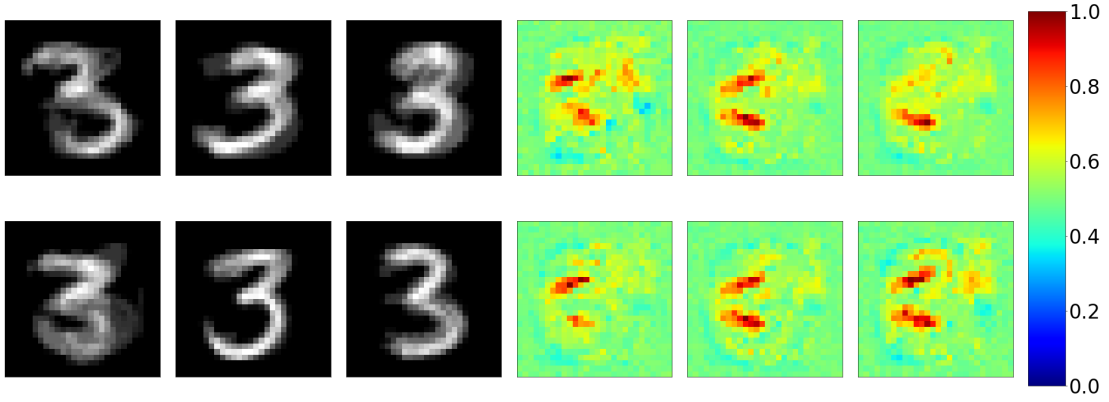


Figure B.15: Visual depiction of between the averaged input features (left) and LRP interpretations (right) for all encoding sample sets in layer six for MNIST class three with a sample set size greater than 100.

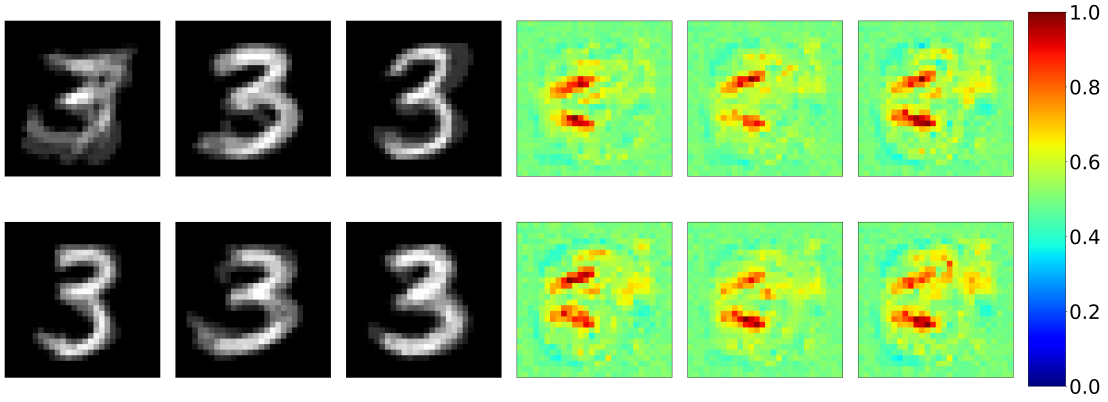


Figure B.16: Visual depiction of between the averaged input features (left) and LRP interpretations (right) for all encoding sample sets in layer seven for MNIST class three with a sample set size greater than 100.

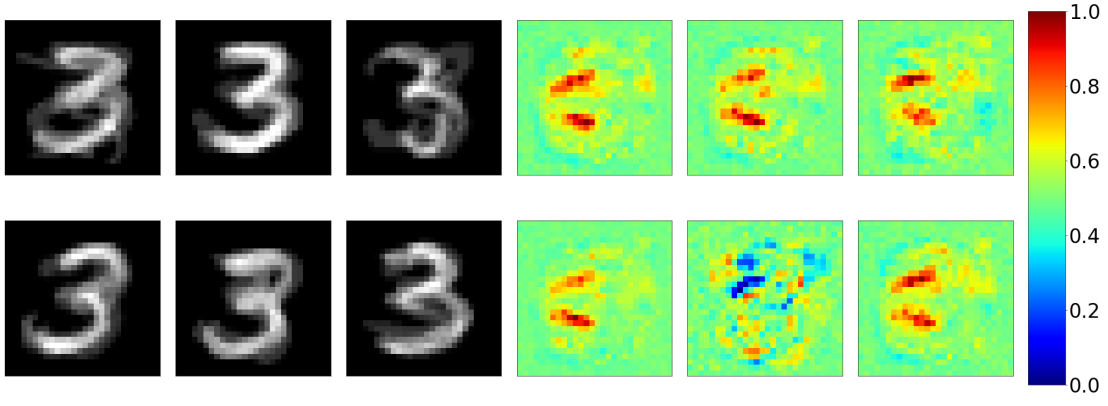


Figure B.17: Visual depiction of between the averaged input features (left) and LRP interpretations (right) for all encoding sample sets in layer eight for MNIST class three with a sample set size greater than 100.

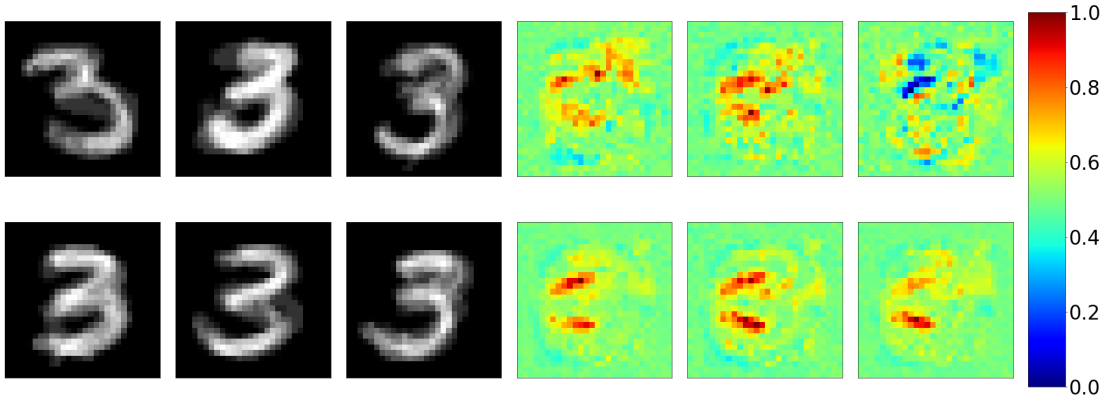


Figure B.18: Visual depiction of between the averaged input features (left) and LRP interpretations (right) for all encoding sample sets in layer nine for MNIST class three with a sample set size greater than 100.

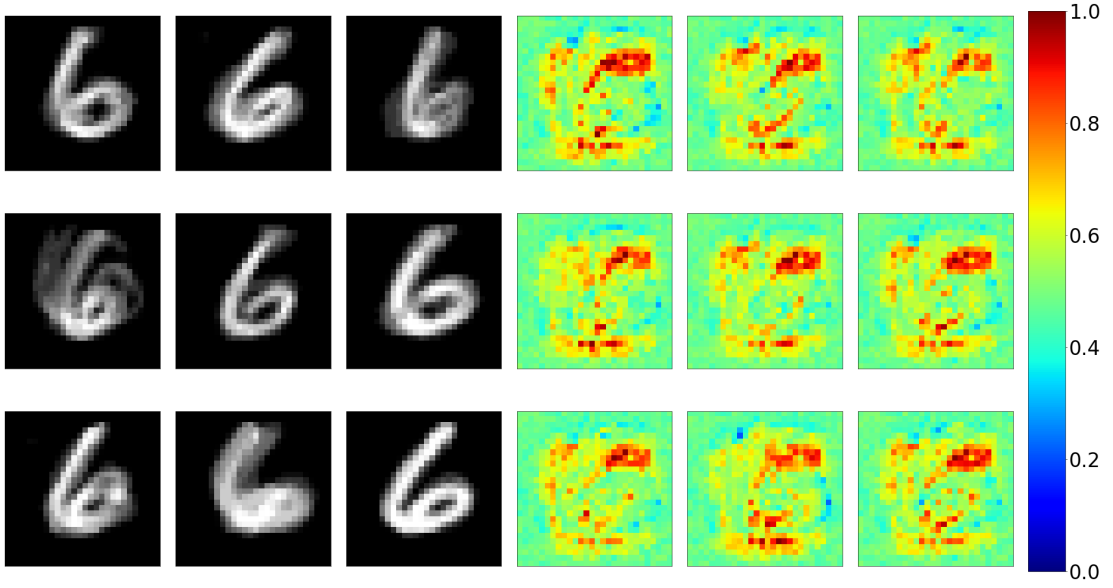


Figure B.19: Visual depiction of between the averaged input features (left) and LRP interpretations (right) for all encoding sample sets in layer six for MNIST class six with a sample set size greater than 100.

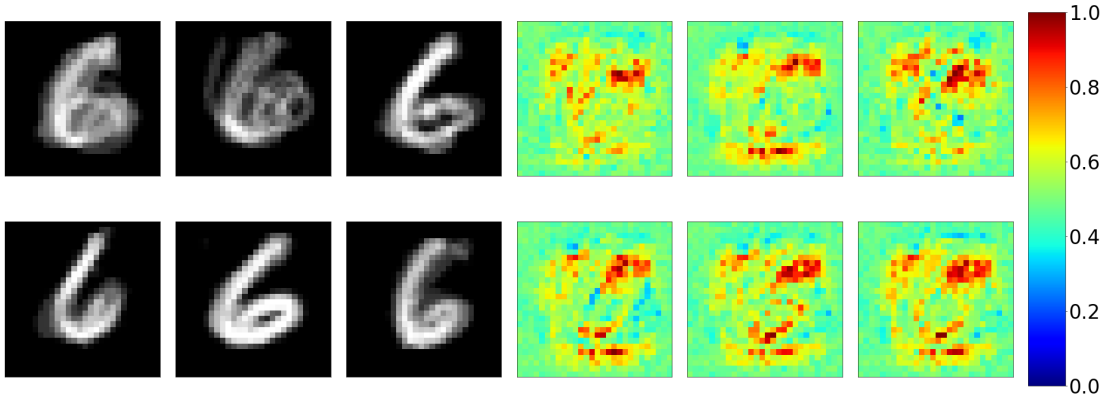


Figure B.20: Visual depiction of between the averaged input features (left) and LRP interpretations (right) for all encoding sample sets in layer seven MNIST class six with a sample set size greater than 100.

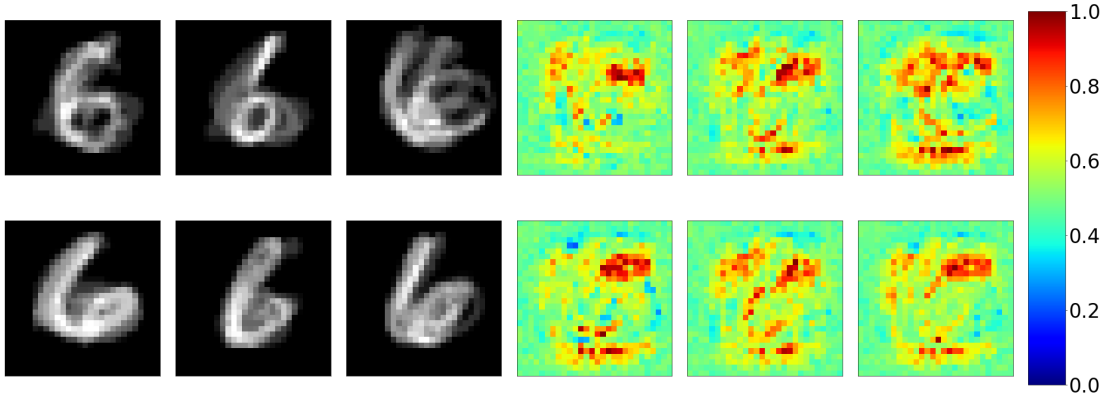


Figure B.21: Visual depiction of between the averaged input features (left) and LRP interpretations (right) for all encoding sample sets in layer eight for MNIST class six with a sample set size greater than 100.

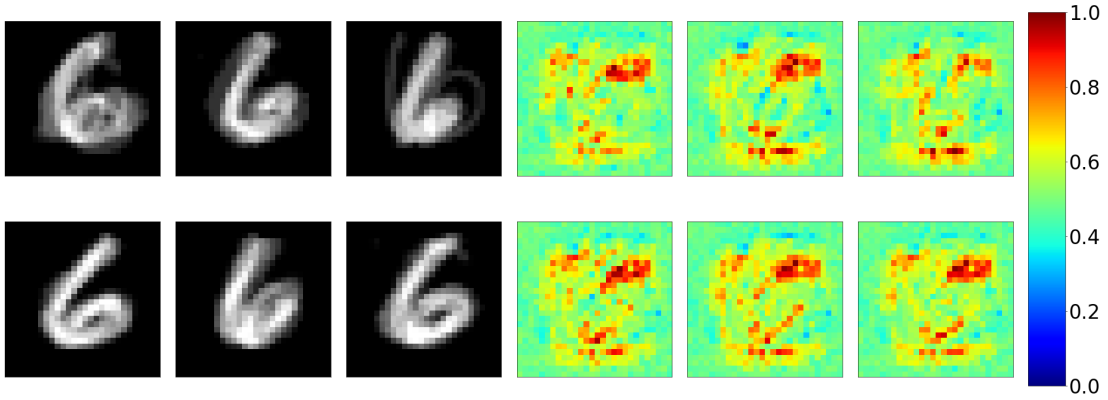
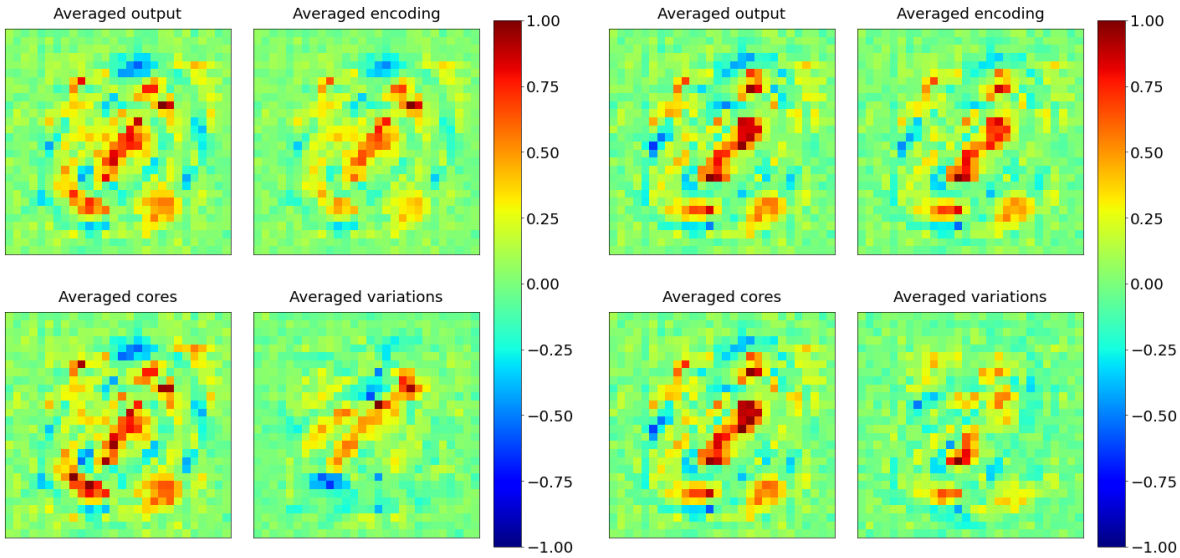
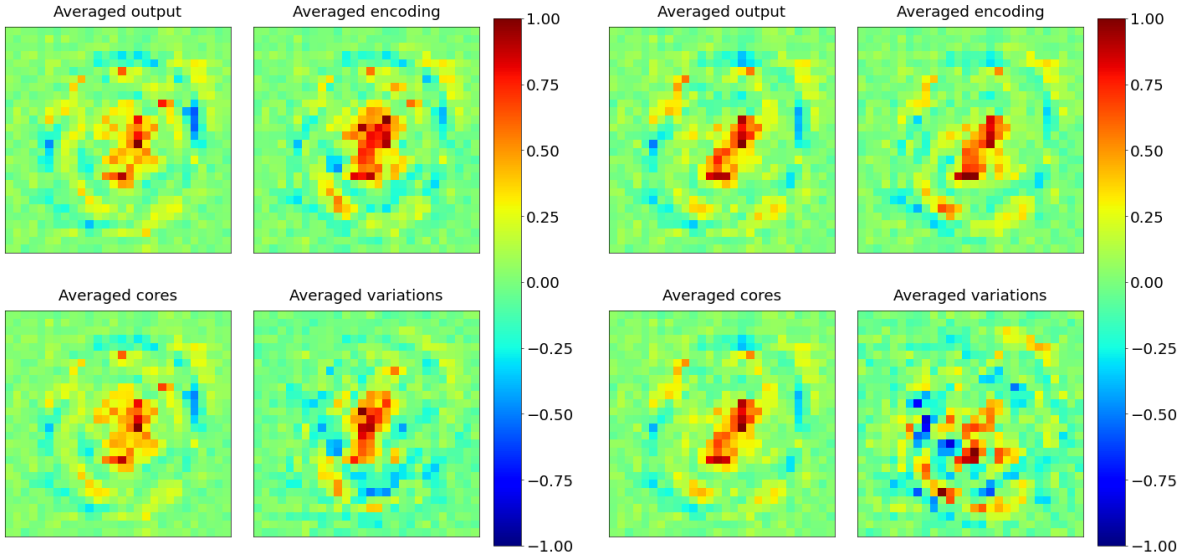


Figure B.22: Visual depiction of between the averaged input features (left) and LRP interpretations (right) for all encoding sample sets in layer nine for MNIST class six with a sample set size greater than 100.



(a) Set size: 136 samples

(b) Set size: 145 samples



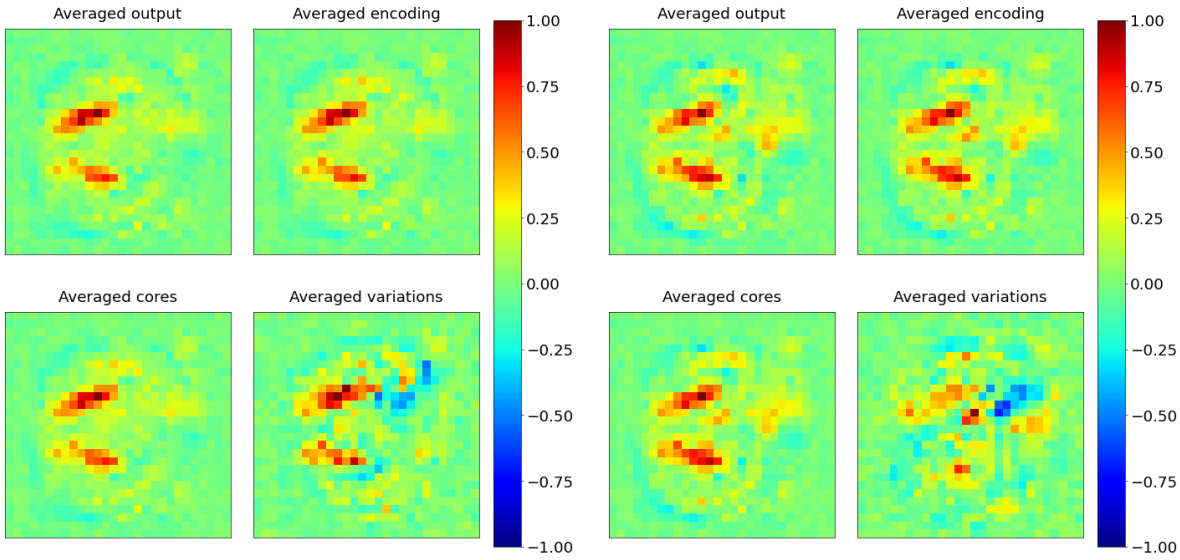
(c) Set size: 198 samples

(d) Set size: 228 samples

Figure B.23: Visual comparisons of the four different interpretation types for four different encodings found in layer eight for MNIST class zero. From left to right, top to bottom, each plot corresponds to an encoding sample set with sizes 136, 145, 198 and 228.

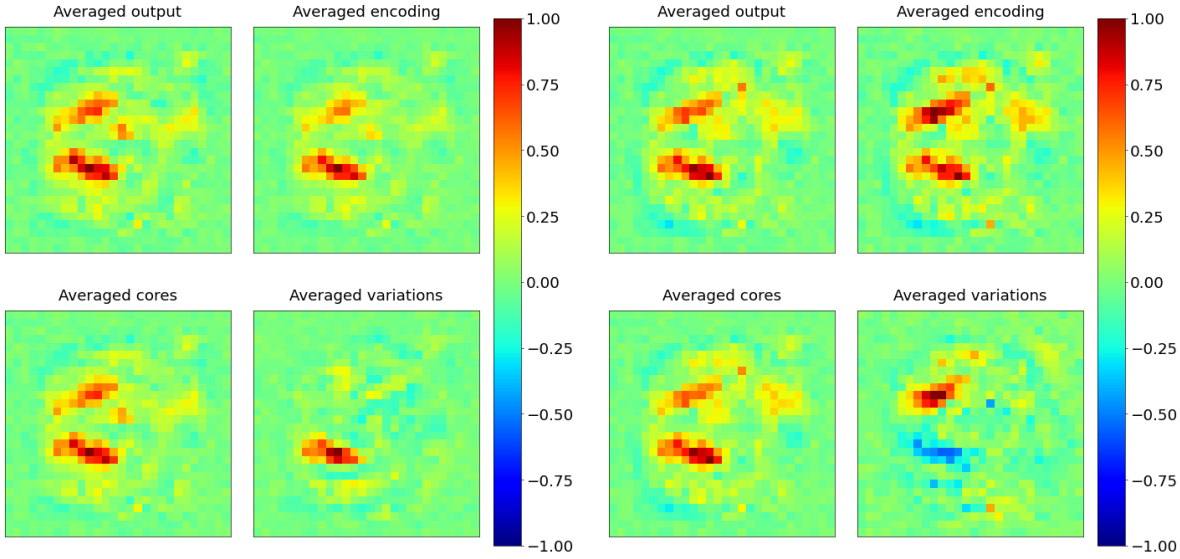
Table B.2: Cosine similarity comparisons between the different LRP interpretations shown in Figure B.23. Encoding sample sets are represented according to their set sizes. The 'output', 'encoding', 'core' and 'variation' scores correspond to the different interpretations types presented in Figure B.23.

	Set size 136	Set size 145	Set size 198	Set size 228
output vs **encoding:**	0.97617	0.97610	0.85066	0.90434
output vs **cores:**	0.97609	0.99382	0.98165	0.99026
output vs **variations:**	0.32245	0.67809	0.41239	0.40125
encoding vs **cores:**	0.93885	0.98466	0.90696	0.92977
encoding vs **variations:**	0.48539	0.80782	0.82025	0.74373
cores vs **variations:**	0.15467	0.69258	0.50300	0.44541



(a) Set size: 150 samples

(b) Set size: 177 samples



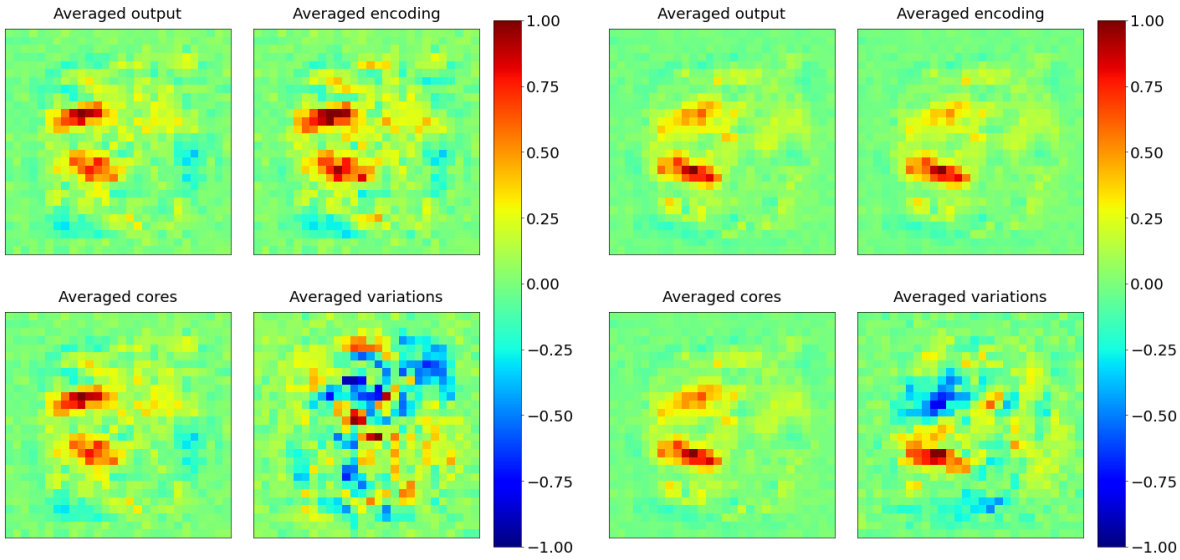
(c) Set size: 351 samples

(d) Set size: 486 samples

Figure B.24: Visual comparisons of the four different interpretation types for four different encodings found in layer seven for MNIST class three. From left to right, top to bottom, each plot corresponds to an encoding sample set with sizes 150, 177, 351 and 486.

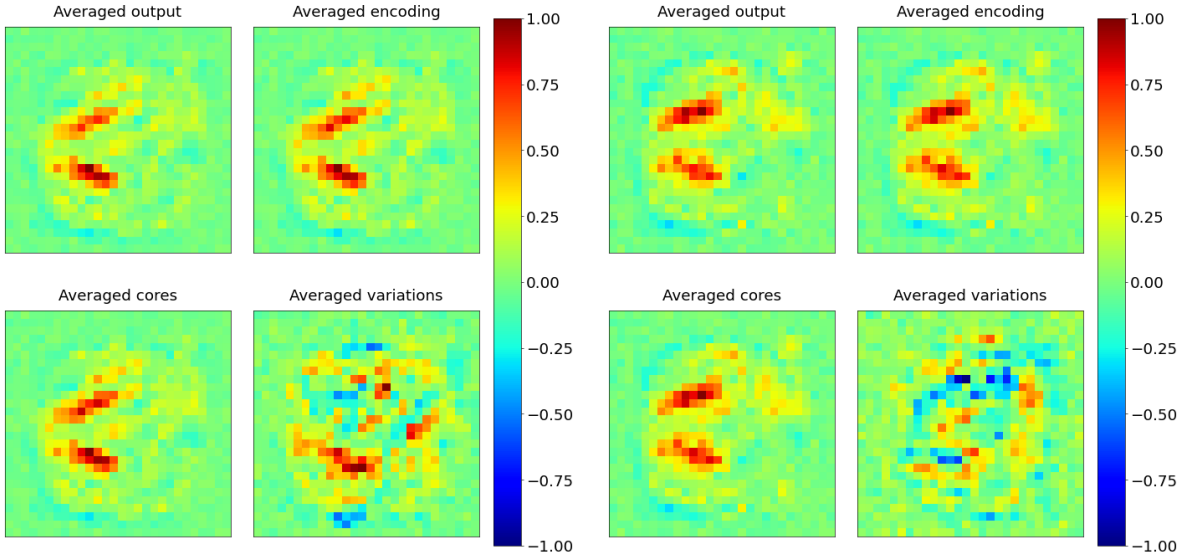
Table B.3: Cosine similarity comparisons between the different LRP interpretations shown in Figure B.24. Encoding sample sets are represented according to their set sizes. The 'output', 'encoding', 'core' and 'variation' scores correspond to the different interpretations types presented in Figure B.24.

	Set size 150	Set size 177	Set size 351	Set size 486
output vs **encoding:**	0.98879	0.99276	0.97627	0.97233
output vs **cores:**	0.99351	0.99410	0.99117	0.99308
output vs **variations:**	0.67620	0.36248	0.63800	0.16962
encoding vs **cores:**	0.99613	0.99844	0.98726	0.97246
encoding vs **variations:**	0.74903	0.41258	0.76655	0.36935
cores vs **variations:**	0.68793	0.36114	0.65460	0.14257



(a) Set size: 252 samples

(b) Set size: 552 samples



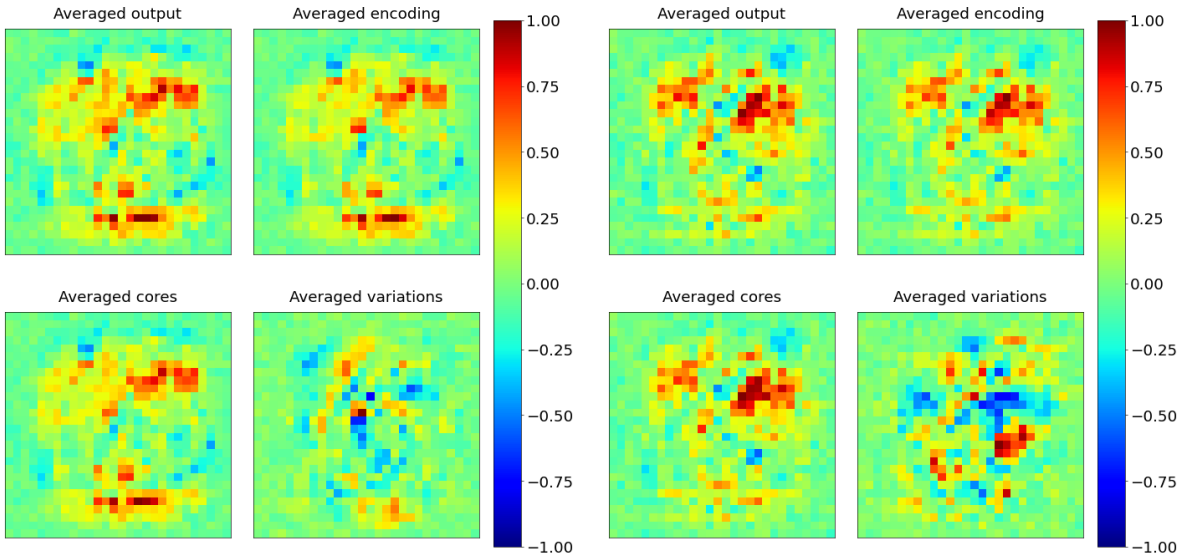
(c) Set size: 1377 samples

(d) Set size: 2222 samples

Figure B.25: Visual comparisons of the four different interpretation types for four different encodings found in layer eight for MNIST class three. From left to right, top to bottom, each plot corresponds to an encoding sample set with sizes 252, 552, 1377 and 2222.

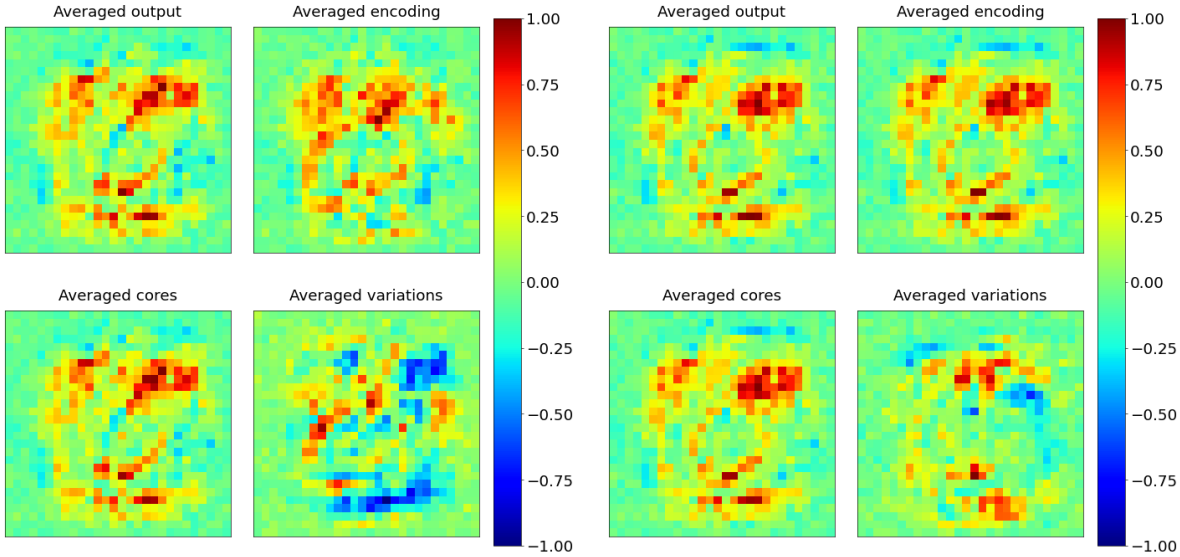
Table B.4: Cosine similarity comparisons between the different LRP interpretations shown in Figure B.25. Encoding sample sets are represented according to their set sizes. The 'output', 'encoding', 'core' and 'variation' scores correspond to the different interpretations types presented in Figure B.25.

	Set size 252	Set size 552	Set size 1377	Set size 2222
output vs **encoding:**	0.94804	0.99046	0.99412	0.98976
output vs **cores:**	0.99290	0.99525	0.99353	0.99345
output vs **variations:**	-0.25833	0.30971	0.52681	-0.13087
encoding vs **cores:**	0.95259	0.98981	0.99851	0.99823
encoding vs **variations:**	0.03863	0.40897	0.54370	-0.03982
cores vs **variations:**	-0.26722	0.27483	0.49714	-0.09921



(a) Set size: 148 samples

(b) Set size: 347 samples



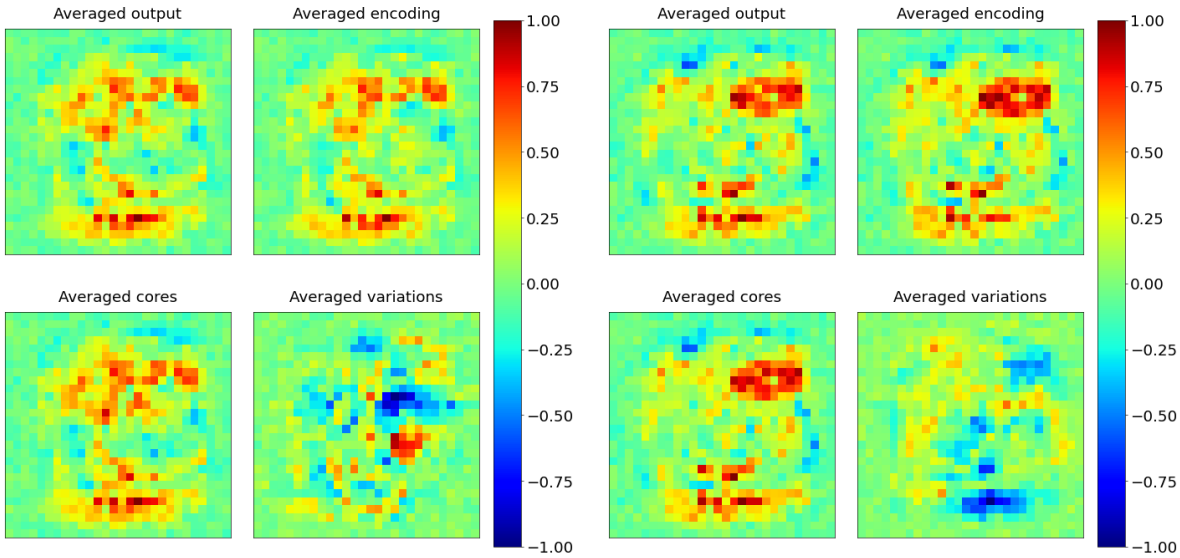
(c) Set size: 448 samples

(d) Set size: 3158 samples

Figure B.26: Visual comparisons of the four different interpretation types for four different encodings found in layer seven for MNIST class six. From left to right, top to bottom, each plot corresponds to an encoding sample set with sizes 148, 347, 448 and 3158.

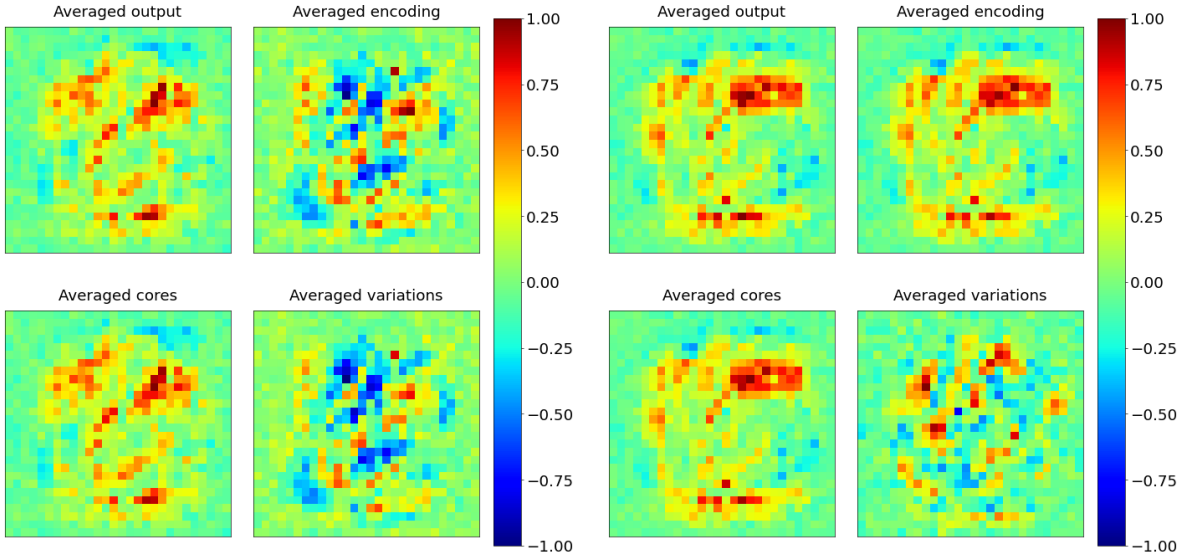
Table B.5: Cosine similarity comparisons between the different LRP interpretations shown in Figure B.26. Encoding sample sets are represented according to their set sizes. The 'output', 'encoding', 'core' and 'variation' scores correspond to the different interpretations types presented in Figure B.26.

	Set size 148	Set size 347	Set size 448	Set size 3158
output vs **encoding:**	0.97948	0.98536	0.79449	0.99473
output vs **cores:**	0.99128	0.98854	0.99210	0.99676
output vs **variations:**	0.15431	-0.07612	-0.22995	0.46427
encoding vs **cores:**	0.98191	0.98550	0.79135	0.99756
encoding vs **variations:**	0.30894	0.02719	0.39717	0.52146
cores vs **variations:**	0.12327	-0.14280	0.24677	0.46057



(a) Set size: 125 samples

(b) Set size: 169 samples



(c) Set size: 450 samples

(d) Set size: 602 samples

Figure B.27: Visual comparisons of the four different interpretation types for four different encodings found in layer eight for MNIST class six. From left to right, top to bottom, each plot corresponds to an encoding sample set with sizes 125, 169, 450 and 602.

Table B.6: Cosine similarity comparisons between the different LRP interpretations shown in Figure B.27. Encoding sample sets are represented according to their set sizes. The 'output', 'encoding', 'core' and 'variation' scores correspond to the different interpretations types presented in Figure B.27.

	Set size 125	Set size 169	Set size 450	Set size 602
output vs **encoding:**	0.94693	0.95724	0.22424	0.98762
output vs **cores:**	0.99251	0.98791	0.99623	0.99462
output vs **variations:**	-0.03382	-0.46940	-0.00953	0.17800
encoding vs **cores:**	0.92877	0.98205	0.21779	0.99241
encoding vs **variations:**	0.27400	-0.23264	0.97214	0.29429
cores vs **variations:**	-0.10198	-0.41190	-0.01704	0.17457

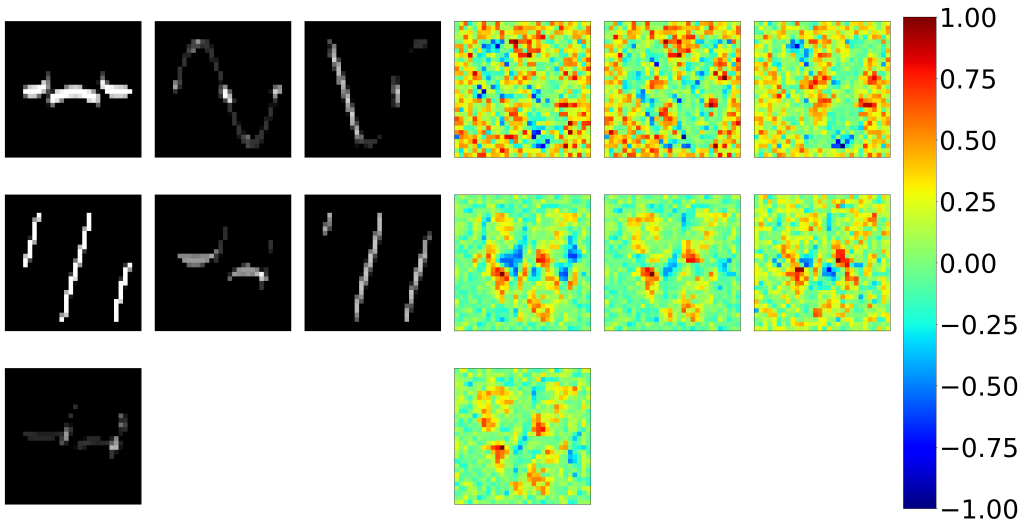


Figure B.28: Visual depiction of the averaged input features (left) over five samples and their respective LRP interpretations (right) for all encoding sample sets in the second-to-last layer with a sample set size greater than 100 from the modified synthetic network's validation set. From left to right, the results of all the different classes are shown as follows: the first two images indicate the samples of class one, the second two images the samples of class two and the rest show the samples of class three.

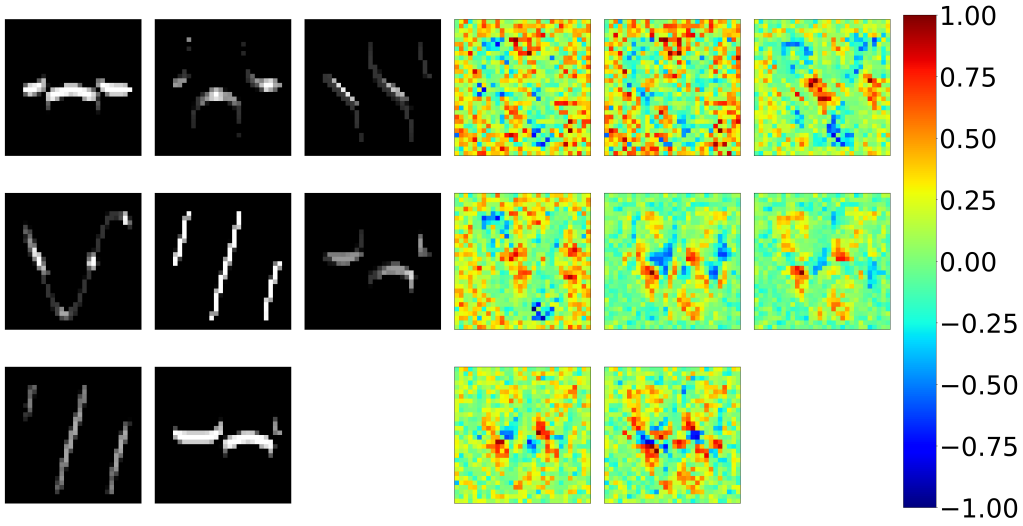


Figure B.29: Visual depiction of the averaged input features (left) over five samples and their respective LRP interpretations (right) for all encoding sample sets in the second-to-last layer with a sample set size greater than 100 from the modified synthetic network's evaluation set. From left to right, the results of all the different classes are shown as follows: the first two images indicate the samples of class one, the second two images the samples of class two and the rest show the samples of class three.

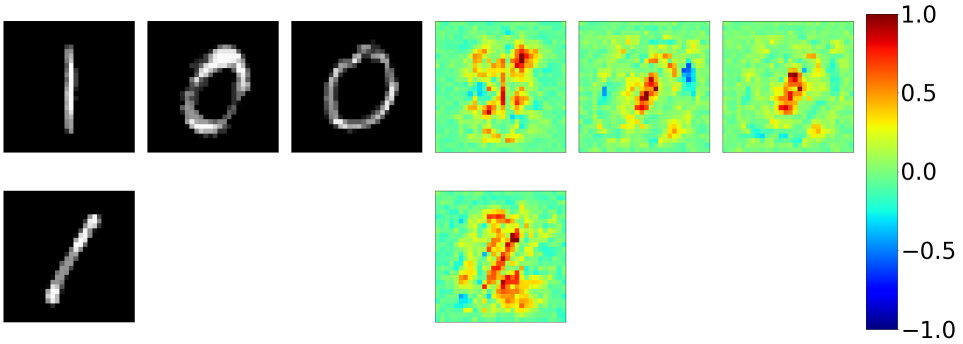


Figure B.30: Visual depiction of the averaged input features (left) over five samples and their respective LRP interpretations (right) for all class zero encoding sample sets in the last layer with a sample set size greater than 100 from the modified MNIST network's validation set.

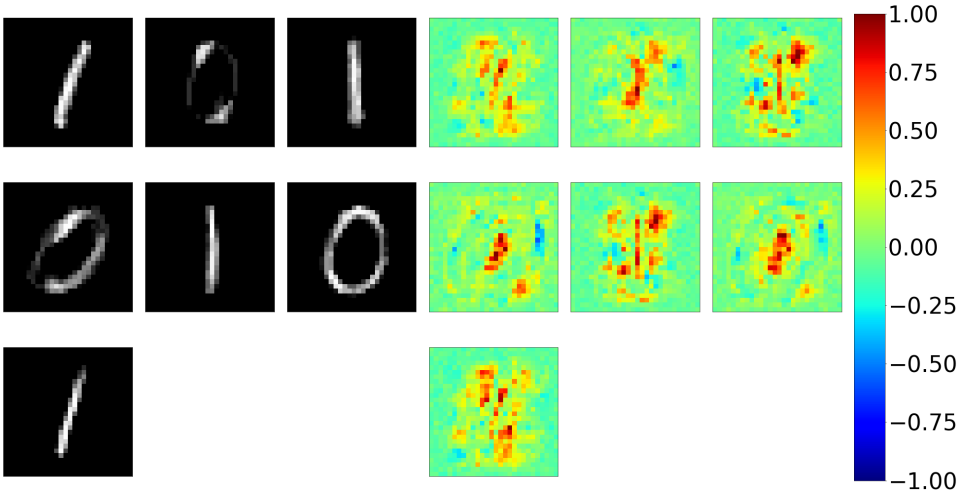


Figure B.31: Visual depiction of the averaged input features (left) over five samples and their respective LRP interpretations (right) for all class zero encoding sample sets in the last layer with a sample set size greater than 100 from the modified MNIST network's evaluation set.