



Exploring the effect of clustered hidden representations on generalisation in DNNs

HL Potgieter



[orcid.org/ 0009-0005-0536-5265](https://orcid.org/0009-0005-0536-5265)

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 C Mouton

Graduation: June 2025

Declaration

I, Hermanus Lambertus Potgieter, hereby declare that the dissertation entitled “Exploring the effect of clustered hidden representations on generalisation in DNNs ”is my own original work and has not already been submitted to any other university or institution for examination.



H.L. Potgieter

Student number: 30057698

Signed on the 27th day of November 2024 at Potchefstroom.

Acknowledgements

Firstly, I would like to thank my supervisors, Prof Marelle H Davel and Dr Coenraad Mouton, for their guidance and their critique of my work, even though it is always difficult to read/hear, it helped me become a better researcher and was instrumental to this work, for which I am grateful. Also, thank you for putting up with my occasional silliness.

Secondly, I would like to thank my parents, Manie and Annalise Potgieter, for their continuous support, without which I would never have come this far. Thank you for always believing in me and teaching me to always dream big. Thank you for keeping me motivated even in the darkest of times.

Thank you to each of the MUST members who played a role in the writing of this dissertation, whether it was technical help, proofreading, a word of encouragement, or a lifelong friendship. Due to the word count limits each of you can not be named here but we will try to include it in the next one.

Lastly, to each person not mentioned here but who still played a role (big or small) you are the unsung heroes, and I thank you.

Abstract

Different approaches have been used to better understand the generalisation ability of deep neural networks (DNNs), that is, their ability to perform well on unseen data. An approach that has only been studied to a limited extent to date, is the way in which hidden representations form implicit clusters within DNNs. Each such cluster is either formed by representations that have similar values, or by some features for some samples that have similar representation values.

We build upon previous work related to clustering in DNNs by exploring the characteristics of representation clusters. Specifically, we investigate the forming of sample-feature clusters between individual and combinations of nodes throughout the network layers. We aim to find a relationship between generalisation and the characteristics of the clusters formed. We investigate these factors on DNNs with different generalisation abilities in order to support or disprove our theoretical intuition on the role of cluster forming in DNNs.

Using existing clustering methods such as k-means, we find that models with better generalisation tend to form purer clusters than those with poorer generalisation. We explore biclustering as a means of identifying sample-feature clusters and find that biclustered representations are also predictive of generalisation. This time, however, models with purer *biclusters* at the convolutional layers tend to generalise more *poorly*. We mitigate the complexity introduced by the high dimensionality of representations by using autoencoders to compress the representations. We find that these compressed representations tend to be predictive of generalisation, with purer clusters again predictive of better generalisation. Combined, these findings provide a novel perspective on the effect of clustered representations on the generalisation ability of DNNs.

Keywords: *DNNs, generalisation, clustering, hidden representations*

Contents

List of Figures	xi
List of Tables	xiv
1 Introduction	1
1.1 Overview	1
1.2 Machine learning	2
1.3 Generalisation	2
1.3.1 Generalisation in ML	3
1.3.2 Generalisation in DNNs	4
1.4 A clustering perspective	5
1.4.1 Hidden representations	5
1.4.2 Studying generalisation from a clustering perspective	7
1.5 Problem statement	8
1.6 Research questions	8
1.7 Objectives	9
1.8 Project scope	10
1.9 Research methodology	10
1.10 Publications	11
1.11 Dissertation overview	11

1.12	Conclusion	12
2	Methods and techniques	13
2.1	Introduction	13
2.2	DNN training	14
2.2.1	DNN model types	14
2.2.2	Optimisation of DNNs	15
2.2.3	Optimisers	16
2.2.3.1	Dadapt	16
2.2.3.2	Prodigy	17
2.2.4	BatchNorm	17
2.3	Hidden representations	18
2.3.1	Definition and notation	19
2.3.2	Sparsity	19
2.4	Dimensionality reduction techniques	20
2.4.1	PCA	20
2.4.2	t-SNE	21
2.4.3	Autoencoder	22
2.5	Clustering	23
2.5.1	Clustering methods	23
2.5.2	Cluster quality metrics	23
2.6	Biclustering	24
2.6.1	Bicluster types	24
2.6.2	Biclustering methods	26
2.6.3	Bicluster quality metrics	28
2.6.3.1	Mean square error	28
2.6.3.2	Scaling mean squared residue	29

2.6.3.3	Virtual error	29
2.6.3.4	Virtual error transposed	30
2.7	Conclusion	30
3	Investigating representational clusters	31
3.1	Introduction	31
3.2	Extracting hidden representations	32
3.3	Case study: PGDL Task 1	32
3.3.1	PGDL challenge	32
3.3.2	Models	33
3.3.3	Activation distributions	36
3.3.4	Low-dimensional perspective	37
3.3.5	Channel-sample activations	39
3.3.6	Clustered representations	42
3.3.6.1	Class-based channel-sample clusters	42
3.3.6.2	Class-agnostic channel-sample clusters	43
3.3.7	Discussion	45
3.4	Case study: batch normalisation	46
3.4.1	Experimental setup	47
3.4.1.1	Models	47
3.4.1.2	Sparsity	49
3.4.1.3	Clustered representations	49
3.4.2	Sparsity analysis and results	50
3.4.2.1	Channel sparsity	50
3.4.2.2	Layer sparsity	52
3.4.3	Clustering analysis and results	53
3.4.3.1	Class-based analysis	53

3.4.3.2	Class-agnostic analysis	55
3.4.4	Discussion	57
3.5	Conclusion	58
4	Biclustering of representations	61
4.1	Introduction	61
4.2	Experimental setup	62
4.2.1	Models	62
4.2.2	Biclustering methods	63
4.2.3	Biclustering quality metrics	64
4.2.3.1	Types of biclusters	64
4.2.3.2	Choosing bicluster quality measures	65
4.2.3.3	QM toolset demonstration	66
4.3	Biclustering of representations	67
4.3.1	Method	68
4.3.2	Result: Bicluster purity	69
4.3.3	Results: Visualising best performing combinations	71
4.4	Predicting generalisation	74
4.4.1	Investigating QM scores at different bicluster combinations	74
4.4.2	Investigating optimal bicluster combinations	76
4.4.3	Biclustering as a measure of generalisation	77
4.5	Discussion	79
4.6	Conclusion	81
5	Autoencoder representations	83
5.1	Introduction	83
5.2	Methods	84

5.2.1	Methodology	84
5.2.2	Autoencoder implementation	85
5.3	Experimental Setup	87
5.3.1	Autoencoder setup	87
5.3.2	Optimal hyperparameter selection	88
5.3.3	Comparing different representations	89
5.3.4	Predicting generalisation using latent dimension representations	90
5.4	Results	91
5.4.1	Model results	91
5.4.2	Class-based clustering analysis	93
5.4.3	Class-agnostic clustering analysis	93
5.4.4	Predicting generalisation with autoencoders	94
5.5	Discussion	95
5.6	Conclusion	97
6	Conclusion	98
6.1	Introduction	98
6.2	Summary	98
6.3	Key findings	99
6.3.1	Representational clustering	99
6.3.2	Representation characteristics	100
6.3.3	Representational key features	102
6.4	Evaluation	102
6.5	Future work	104
6.6	Concluding remarks	105
	References	106

A	Supplemental content	114
A.1	Appendix: Chapter 3	114
A.1.1	Low-dimensional perspective	114
A.2	Appendix: Chapter 4	115
A.2.1	Results: Bicluster purity	115
A.2.2	Visualising best performing clusters	115
A.2.3	Investigating QM scores at different bicluster combinations	116
A.2.4	Predicting generalisation	116
A.3	Appendix: Chapter 5	117
A.3.1	Class-agnostic clustering results	117

List of Figures

1.1	Example: Number of samples per path in the network per class, before and after training. Reproduced from [28], with permission.	6
2.1	Different bicluster structures, (a) Single bicluster, (b) exclusive row and column biclusters, (c) Checkerboard structure, (d) exclusive rows biclusters, (e) exclusive column biclusters, (f) non-overlapping biclusters with tree structure (g) non-overlapping non-exclusive biclusters, (h) overlapping biclusters with hierarchical structure, and (i) non-structured overlapping biclusters. Adapted from [77] . . .	27
3.1	Layer structure of the small and large PGDL Task 1 models. Green blocks represent the convolutional/pooling representations (indicated as ' $Channels \times Height \times Width$ ') while orange blocks represent the fully connected layer representations.	34
3.2	Distribution of feature activation values of each of the small models (BS and GS) for 5 000 samples across the two convolutional layers and the fully connected layer.	36
3.3	Lower dimensional views of BS (a, b and c) and GSs' (d, e and f) hidden representations of each layer reduced by t-SNE to two dimensions. Colours indicate the true class label of each sample point.	38
3.4	The activation frequency per channel and class, for the convolutional layers of BS (a and c) and GS (b and d). The channels are arranged in ascending order from lowest average activation frequency over classes	40
3.5	The activation frequency a channel of the network is active for, for the convolutional layers of BS (a, c) and GS (b and d) for each class. The channels are arranged in ascending order from lowest average activation frequency over classes	41
3.6	Class-based DBI score per layer, for both the good (blue) and bad (orange) models. A lower DBI indicates higher cluster purity.	43
3.7	The optimal number of class-agnostic clusters per layer for both the good (blue) and bad (orange) models, selected by the lowest DBI score.	44

3.8	Class-agnostic DBI score per layer, for both the good (blue) and bad (orange) models. A lower DBI scores indicate higher purity.	44
3.9	Average channel sparsity for BatchNorm (blue) and Non-BatchNorm (orange) models for selected convolutional layers of the standard CNN models trained on MNIST. Error bars show the standard deviation across 4 random initialisation seeds.	51
3.10	Average channel sparsity for BatchNorm (blue) and Non-BatchNorm (orange) models in selected VGG-16 convolutional layers trained on CIFAR10, with error bars showing the standard deviation across 4 random initialisation seeds.	51
3.11	Average layer sparsity for models trained with (blue) and without (orange) Batch-Norm. Error bars indicate the standard deviation across 4 random initialisation seeds.	52
3.12	Average class-based DBI score per layer for BatchNorm (blue) and Non-BatchNorm (orange) models, with error bars showing standard deviation across 4 random initialisation seeds. Lower scores indicate higher purity.	54
3.13	Average optimal number of class-agnostic clusters per layer for BatchNorm (blue) and Non-BatchNorm (orange) models, with error bars showing the standard deviation across 4 random initialisation seeds.	55
3.14	Average class-agnostic DBI score for k-means per layer for BatchNorm (blue) and Non-BatchNorm (orange) models, with error bars indicating standard deviation across 4 random initialisation seeds. Lower scores indicate higher purity.	56
4.1	Average VEt and SMSR for all tested bicluster combinations. Both convolutional layers for both BS and GS are shown. A lower score indicates a purer bicluster average. (Take note of the scale difference between models.)	70
4.2	Average VEt and SMSR for all tested bicluster combinations. The fully connected layer for both BS and GS are shown. A lower score indicates a purer bicluster average. (Take note of the scale difference between models.)	71
4.3	BS and GS Conv_1 representations sorted to the cluster labels generated by the best performing bicluster combinations.	72
4.4	BS and GS Conv_2 Representations sorted to the cluster labels generated by the best performing bicluster combinations.	73
4.5	SMSR for each bicluster combination, for each of the five randomly selected models. The scores are sorted by the index of the sorted mean score of all five models in ascending order. Each colour represents a different model.	75
4.6	Number of sample and channel clusters responsible for each of the five sampled models Conv_2 QM sorted scores as is shown in Figure 4.5. The x-axis denotes the index of the mean scores when sorted in ascending order, while the y-axis denotes the number of clusters for the samples and channels respectively.	76

4.7	SMSR for each model for Conv_2 , measured with each of the proposed methods and plotted against the evaluation accuracy of each model. We also indicate the Kendall's rank of each plot.	80
5.1	Autoencoder implementation for Conv_1 of a small convolutional model. Grey block represents the frozen layers, green blocks indicate convolutional layers of the AE and orange blocks indicates fully connected latent dimensions. Included is a shortened model that uses the AE as input. Representation sizes is included for each layer.	86
5.2	Layer wise class-based DBI score of each of the five AE plotted against the evaluation accuracy of each respective model. We also indicate the Kendall's rank of each plot.	95
A.1	Lower dimensional views of BL (a, b and c) and GL (d, e and f) hidden representations of each layer reduced by t-SNE to two dimensions. Colours indicate the true class label of each sample point.	115
A.2	Lin_1 Representations sorted to the cluster labels generated by the best performing bicluster combinations according to VEt and SMSR for both BS and GS. . .	117
A.3	Average VET for each bicluster combination, for each of the five randomly selected models. The scores are sorted by the cluster index of the mean score over all five models (low to high) for each bicluster combination. Each colour represents a different model.	118
A.4	Average SMSR for each bicluster combination of the three layers, for each of the five randomly selected models. The scores are sorted by the cluster index of the mean score over all five models (low to high) for each bicluster combination. Each colour represents a different model.	119
A.5	Highest correlation VEt for each of the layers, measured with each of the proposed methods plotted against the evaluation accuracy of each model. We also indicate the Kendall's rank of each plot.	120
A.6	Highest correlation SMSR for each of the layers, measured with each of the proposed methods plotted against the evaluation accuracy of each model. We also indicate the Kendall's rank of each plot.	121

List of Tables

3.1	Overview of PGDL challenge tasks	33
3.2	Chosen PGDL Task 1 models performance metrics	35
3.3	Chosen PGDL Task 1 models' corresponding model-specific training hyperparameters.	35
3.4	Train, validation, and evaluation classification accuracy for the BatchNorm (BN) and non-BatchNorm (NBN) models on MNIST and CIFAR10. $\pm$ indicates the standard deviation across 4 random initialisation seeds.	49
4.1	Different types of bicluster examples that we may encounter.	65
4.2	QM purity scores for each of the example biclusters as showed in Table: 4.1. . . .	67
4.3	Summary of the average and best VEt and SMSR from the bicluster combination grid search.	69
4.4	Kendall's rank coefficient between the evaluation accuracy and various QM score for PGDL Task 1 small models	78
5.1	Chosen PGDL Task 1 models performance metrics sorted in ascending order of evaluation accuracy.	90
5.2	Chosen PGDL Task 1 models and corresponding model-specific training hyperparameters.	90
5.3	AE performance metrics for both GS and BS. Included is GS and BS performance metrics for comparison. Also includes the evaluation delta which is the difference between the AE evaluation accuracy and that of the model.	91
5.4	AE performance metrics <code>Conv_1</code> . Also includes the evaluation delta which is the difference between the AE evaluation accuracy and that of the model.	92

5.5	AE performance metrics for <code>Conv_2</code> . Also includes the evaluation delta which is the difference between the AE evaluation accuracy and that of the model.	92
5.6	GS and BS DBI scores for class-based clustering. TRSM as well as the latent dimension of the autoencoder trained on the representation are shown.	93
5.7	GS and BS DBI scores for class-agnostic clustering. TRSM and the latent dimension of the autoencoder trained on the representation are shown. Representations are clustered with k-means between 2 and 25 optimal clusters with the DBI score of the best-performing number of clusters being shown.	94
A.1	Summary of the average and best VET and SMSR from the bicluster combination grid search for GL and BL.	116
A.2	P-values associated with the Kendall's rank correlation using different QM scores for PGDL Task 1 small models. All results are significant at $p < 5e-4$, except the bolded results that are only significant at $p < 1e-4$	119
A.3	Kendall's rank coefficient between the evaluation accuracy and various QM score for PGDL Task 1 large models	120
A.4	P-values associated with the Kendall's rank correlation using different QM scores for PGDL Task 1 large models. All results are significant at $p < 1$, except the bolded results that are only significant at $p < 1e-2$	121
A.5	GS and BS optimal number of clusters for class-agnostic clustering. TRSM and the latent dimension of the AE trained on the representation are shown. Representations are clustered with k-means between 2 and 25 optimal clusters with the optimal cluster being selected by the lowest DBI score	122

Chapter 1

Introduction

“Well, I mean, yes idealism, yes the dignity of pure research, yes the pursuit of truth in all its forms, but there comes a point I’m afraid where you begin to suspect that if there’s any real truth, it’s that the entire multidimensional infinity of the Universe is almost certainly being run by a bunch of maniacs. And if it comes to a choice between spending yet another ten million years finding that out, and on the other hand just taking the money and running, then I for one could do with the exercise.”

- Douglas Adams, The Ultimate Hitchhiker’s Guide to the Galaxy Page 132 [1]

1.1 Overview

Deep learning (DL) has become a particularly important set of machine learning techniques. At the same time, many open questions remain with regard to the ability of these models to generalise well, that is, their ability to perform well on unseen data. This chapter introduces DL, with a focus on the generalisation ability of such models. It also introduces how DL models produce hidden representations and why a better understanding of the characteristics of these hidden representations could contribute to a better understanding of the generalisation ability of deep neural networks (DNNs).

Once the context has been sketched, we introduce the problem statement we aim to address (Section 1.5). Subsequently, we outline the main research questions (Section 1.6) and research objectives of the study (Section 1.7). We restrict the project scope (Section 1.8) and describe the research methodology followed (Section 1.9). Finally, we briefly mention publications resulting from the study (Section 1.10) before presenting an overview of the rest of this dissertation (Section 1.11).

1.2 Machine learning

Machine learning (ML) can be described as algorithms that learn from experience to perform a specified task with respect to a performance metric, with the classification as ML depending on whether their performance on the task improves [2, p. 97]. ML is used to solve complex tasks that might be difficult for humans or to automate a process. Two of the main categories of learning problems in ML are supervised and unsupervised learning. Supervised learning refers to a task in which the data set contains both the features and the labels, and the problem is to predict the labels given the features as input. Unsupervised learning refers to the task of extracting information from the dataset without the use of labels.

DL is a subset of ML and it encompasses learning techniques where models contain multiple layers of non-linear processing units. DL is a field that is still growing and is actively studied. DNNs are used on complex tasks such as image classification [3], natural language processing [4], and speech recognition [5] [6, ch. 1]. This is due to their ability to process large amounts of data and their ability to perform well on high-dimensional data as well as unseen data [6, ch. 1]. For these reasons, they are excellent tools to use in solving large-scale real-world problems.

1.3 Generalisation

We first discuss generalisation in the context of general ML models before looking more closely at generalisation of DNNs, in particular. We also discuss why we are particularly interested in this.

1.3.1 Generalisation in ML

Generalisation in ML is described as the ability to accurately predict an output given some unseen input, by finding patterns in the seen data [6, ch. 3.6]. The fundamental problem in ML is to discover patterns that generalise beyond the training data. In classic ML, one common approach to enhancing the generalisation of a model is to minimise the gap between the test error and the training error. It serves as an indicator of how well the model fits the seen data and its ability to make accurate predictions on unseen data.

Two challenges that are present in ML when trying to achieve a small training error and good generalisation are overfitting and underfitting. Underfitting occurs when the model cannot fit the training data sufficiently and results in a large training error. Overfitting occurs when the model fits the training data but performs poorly on the test data, resulting in a large gap between the training error and the test error. This usually indicates that the model memorised the training set and not a general pattern [2, p. 110]. The ability of the model to fit a wide range of functions is referred to as its capacity. By altering the capacity of the model, we can control whether it overfits or underfits. Although a model can fit a wide range of functions given sufficient representational capacity, its effective capacity is constrained by the optimisation algorithm [7, p. 134]. We want to use an effective capacity where the training error is reduced, not necessarily a capacity where the training error is zero.

One way of combatting overfitting is through a regularisation method. Regularisation techniques are modifications to the learning algorithm that encourage the model to have less complex solutions. This is done by introducing a penalty term to the loss function to reduce the generalisation error, reducing the complexity of the model class, or constraining the set of values that the parameters can take [6, ch. 5.5]. Different regularisation methods are used, such as early stopping and L1 or L2 regularisation, each of which is used for specific tasks, including the data and type of model used [2, p. 116],[6, ch. 3.7].

Two types of error that characterise generalisation in ML are bias and variance. Bias is the average difference between the predicted and ground truth, whereas variance is the variability in the model's prediction when resampling from the true underlying data distribution. An increase in the bias leads to the model underfitting, whereas an increase in the variance results in the model overfitting. The trade-off in complexity is the key to balance between bias and variance

and is critical for achieving good generalisation results by minimising the total error [2, p. 127].

Generalisation is well understood in traditional machine learning under statistical learning theory (SLT). However, the same cannot be said for the generalisation of DNNs, which we elaborate on in the following section.

1.3.2 Generalisation in DNNs

Our understanding of why DL works as well as it does is still limited. Zhang et al. [8] showed that DNNs can reach levels of generalisation which is not explainable with SLT. Specifically, DNNs with enough capacity are able to memorise random noise while still being able to generalise well. These findings challenge the notion that bias and variance should be balanced in DNNs to improve generalisation [9]. Following this, the classic ' U '-shaped bias-variance curve is not applicable to DNNs, but rather the double descent (DD) phenomenon [9], [10]. It shows that the performance of a model gets worse as the capacity increases until the effective capacity is reached, where the model interpolates the data, after which the model performance *increases* again with a further increase in capacity. The reasons behind these phenomena remain unclear [9], [10].

Irrespective of the generalisation ability of ML models, the generalisation ability of DNNs is still being studied from many perspectives, such as: the geometry of the loss landscape [11], statistical measures of stability and robustness [12], size of margins (distance to the decision boundary between classes) [13], information-theoretic techniques [14], etc. Nevertheless, we are still unable to predict which model will generalise better when comparing two models given a set of training data.

Many aspects of DNNs and their optimisation are well understood. For instance, their representational ability is known: given sufficient size, even a multilayer perceptron (MLP) can model (almost) any function [7, p. 29]. Similarly, the process for training DNNs, stochastic gradient descent (SGD), is a straightforward optimisation process [3], [15]. However, how well these models are expected to generalise remains an open question and an active area of research.

A promising research direction is to study the characteristics of the internal data representations formed by DNNs. Aspects of these representations that have been studied include the size of margins in the representation space [16]–[18]; representation sparsity, that is, the number of

non-zero elements in a data representation [19]; and the ‘quality’ of representations, evaluated using the consistency of class-specific representations and their robustness when combined [18]. We discuss this last approach in more detail in the next section.

1.4 A clustering perspective

This section introduces hidden representations, how they can be clustered, and how this can be used to study generalisation in DNNs.

1.4.1 Hidden representations

Hidden representations are internal representations of the data that a model generates as it makes predictions [20]. These representations are created through the processing of the input data by the hidden layers in the model, i.e. the activation values at each layer. The hidden representations are updated iteratively during the training process and capture intricate patterns or features present in the data. These representations are not directly interpretable, but techniques can be used to provide insight into the information encoded in the hidden representations and improve our understanding of the models’ learning process [21], [22].

The activation values can be represented as continuous (real values) or binary (1 or 0) values. Continuous activation values are generated by activation functions such as rectified linear unit (ReLU) [23] or sigmoid, while binary activation values are created by unit step activation functions [6, ch. 5.1]. Binary activation values consist of two values that are set depending on whether the total input is greater than or less than a set threshold. Continuous activation values are the total input mapped to a certain value depending on the activation function used. Activation functions such as ReLU incorporate both these methods, where negative values are set as zero, and input values larger than zero are calculated by a linear function. In addition, a ReLU activation function can be analysed as a binary activation function if it is recorded as ‘1’ whenever it produces an activation value and ‘0’ otherwise.

Activation values can be analysed in different ways with regard to the type of architecture used. When working with an MLP, there is a single activation value for each node. Convolutional neural networks (CNNs) produce a feature map consisting of multiple activation values for

each channel [24]. This can result in different ways of clustering the activation values, such as spatial or sample clustering [15]. Spatial clustering clusters ‘pixels’ together that form similar parts in different samples. Sample clustering clusters different samples together that have some corresponding features or characteristics. Other methods such as co-clustering combines spatial and sample clustering.

Whether obtained from a CNN or MLP, a set of hidden representations associated with a batch of samples can be viewed as a ‘hidden representation matrix’. This matrix would then consist of activation values, with a single value per sample and feature. The feature can either be associated with a node (as in the case of MLPs), with a value inside a feature map (for CNNs), or for a summary value associated with a feature map, such as an average.

If a single ReLU-activated node is observed for multiple samples, some samples are effectively grouped together, since the node will only produce an output for some samples. A group of samples that are clustered together is referred to as the ‘sample set’ of that node [25], [26]. Different clusters will form for each node, and each node becomes a feature generator during training that activates when certain features are present in the sample [25]–[27]. With image data, in the deeper layers, the nodes cluster larger textural structures but can also replicate textures already captured in the previous layers [27]. This is related to the observation that the nodes in the first layers of an MLP are not specific to a certain class, while the deeper layers become more class-specific [25].

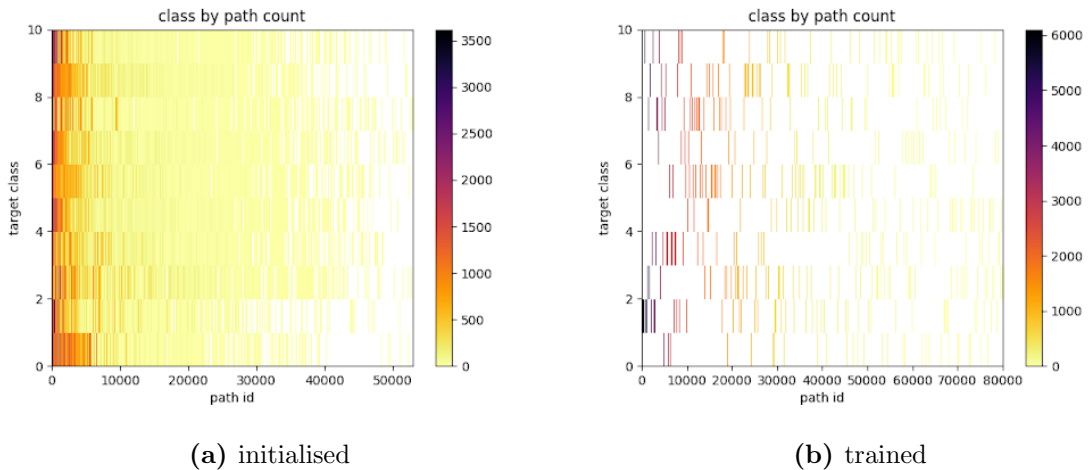


Figure 1.1: Example: Number of samples per path in the network per class, before and after training. Reproduced from [28], with permission.

This organising effect is illustrated in Figure 1.1, where the network uses specific paths for each class of samples. The path mentioned is the specific nodes in each layer that connect the input layer to the output for nodes that have non-zero values. Figure 1.1a shows the number of paths used by an untrained network for each class of a dataset, indicating that many different paths are used for each classification. Meanwhile, in Figure 1.1b, which is of the same network after training on the MNIST data set, the network uses fewer paths for each class. This indicates that the network clusters similar samples during training and uses specific nodes to achieve this.

1.4.2 Studying generalisation from a clustering perspective

There are limited studies that explore the link between representation clustering and generalisation. The two main papers that make a direct link are those of Natekar et al. [18] and Liao et al. [15].

Natekar et al. proposed three measures of representation quality to predict generalisation in a post-hoc diagnostic setting, where the quality is measured after training [18]. The first metric they use is consistency, which is relevant as it is used as an indication of cluster purity of activations. Each of the samples are clustered according to its class label. To measure the consistency of the clusters, the Davies-Bouldin index (DBI) [29] is calculated on a dimensionally reduced version of the activation values of a layer. This produces a measure of how consistent the representations are within a cluster as well as how different they are from other clusters. Dimensionality reduction is performed with principal component analysis (PCA) and Max-Pooling (discussed in more detail in Section 2.4). The main observations from their study were that simple measures based on the quality of internal representations are somewhat predictive of the generalisation gap, as well as that the consistency in clustered representations of objects in internal layers aid in generalisation. This indicates that the model inherently clusters the data according to the class to which it belongs, and that these clusters become more pure as the model generalisation ability increases.

Liao et al. [15] use the process of clustering representations as a regulariser and demonstrate some performance gains in the process. They looked at three different clustering methods, namely: sample clustering, spatial clustering, and the co-clustering of both samples and channels. They implemented the technique by adding a weighted regulariser term to the standard loss.

The regularisation term consists of the distance from the element to the mean of the cluster. The cluster centres are initialised and updated with k-means and then resampled and updated through the k-means++ procedure. Their results seem to be fairly similar, with sample and spatial clustering outperforming co-clustering.

Clustering in the unsupervised learning setting has gained increasing popularity among researchers due to its ability to discover hidden patterns in unlabelled data. In ‘Deep clustering’ by Caron et al. [27], clustering of internal representations has been used to train CNNs. The training process consists of an alternating process of clustering internal representations (using standard k-means) and updating the parameters until convergence. Once the clustering is complete, the cluster tags are used as pseudo-labels for the corresponding hidden representation vectors. The pseudo-labels are used to train a classifier network on top of the CNN to predict the pseudo-labels for each hidden representation vector. It is a form of self-supervised learning. The idea was further developed by Zhan et al. [30] who introduced a batch-aware version that addresses the instability caused by samples changing their batch membership. The same idea was used by Li et al. [31] to force the training process to create representations that are close to the cluster prototypes (cluster centroids).

1.5 Problem statement

Many different approaches have been taken to better understand the generalisation ability of DNNs. An approach that has not been well studied in this context is the forming of data clusters by networks as they implicitly sub-divide the training data during modelling. The hidden representation matrix can either be clustered per representation (each representation is an element), or sample-feature biclusters can be studied. It is not known what effect the characteristics of such sample clusters or sample-feature biclusters have on a network’s generalisation ability.

1.6 Research questions

The following research questions are identified:

1. What different types of sample clusters (such as per node, per layer, per channel, or com-

bbinations of these) formed within the hidden representations of a network are potentially useful to investigate?

2. What are appropriate ways in which to analyse clusters formed within the hidden representations?
3. What cluster characteristics show a difference in behaviour between models that generalise well and those that generalise poorly?

1.7 Objectives

The main aim of this study is to track the forming of sample-feature clusters between individual elements as well as combinations of network elements in models. This is done with the aim of better understanding the relationship between the generalisation ability of the network and the characteristics of the clusters formed.

In order to achieve this aim and address the main research questions, the following objectives are set:

- Determine whether clear representation clusters can be observed in CNNs.
- Determine whether clear sample-feature biclusters can be observed in CNNs.
- Determine what the implications are of using different hidden representation summary methods at different layers of a model. What effect does this have on the clusters observed?
- Determine whether hidden representations cluster according to their class labels.
- Define quantitative measures of cluster characteristics, such as cluster purity or number of clusters that form, that can be used to probe the differences between models.
- Determine whether the clustering characteristics measured show a difference in behaviour between models that generalise well and those that generalise poorly.

1.8 Project scope

Given the problem statement (Section 1.5), we restrict the scope of the research to limited data sets and architectures, with the main objective being image classification.

- **Datasets:** The data sets that will be used are MNIST [32] for prototyping, while using the larger set CIFAR10 [33] for analysis. These data sets have varying complexity, which is desirable when analysing DNNs.
- **Architecture:** CNN architectures such as VGG-like networks [34] and standard CNNs [10] will be used.
- **Activation functions** We will only consider the ReLU [23] activation function, both for its popularity and as it can be analysed by producing continuous or binary representations.

1.9 Research methodology

This study consists of exploratory research that builds on previous work on clustering in DNNs. The research will be carried out through empirical experiments and quantitative analysis thereof. The following aspects form part of this study:

- **Literature review:** A literature review will be conducted to investigate the fundamentals of DNNs, as well as their training procedures and optimisation techniques. Furthermore, different methods to extract hidden representations from networks, as well as dimensionality reduction techniques will be analysed. Relevant clustering methods will be studied, along with ways in which to measure the purity of the clusters.
- **Codebase development:** mustnet (an internal codebase for the experimentation and analysis of neural networks) will be extended with the necessary components to extract and analyse hidden representations. This is done in the Python programming language and utilising frameworks such as PyTorch [35] and libraries such as Scikit-learn [36].
- **Experimental development:** A series of representational clustering experiments will be conducted on both custom-developed and trained models, as well as pre-trained models

that possess known characteristics. In addition, analysis techniques will be developed that measure the characteristics of clusters. The measurements taken in controlled experimental setups will be reported on.

- **Discussion of experimental findings:** Once experimentation is concluded, results will be interpreted and discussed in detail.

1.10 Publications

Some of the research related to this dissertation has been published during the course of the study:

- “Impact of batch normalization on convolutional network representations” [37]. This paper overlaps with some of the content of Chapter 3, specifically the case study in Section 3.4. This publication was mainly the efforts of H.L. Potgieter. C. Mouton and M.H. Davel performed an advisory role. H.L. Potgieter carried out the experimental setup, generated the results, and performed the analysis.
- “Is network fragmentation a useful complexity measure?” [38]. This work was collaborative. Although it is related to this study, as it also studies generalisation in DNNs, the material from this paper is not included in the dissertation.

1.11 Dissertation overview

This dissertation has the following structure:

- **Chapter 2 Methods and techniques** In this chapter, we provide essential background information on the techniques and methods that are used throughout this study.
- **Chapter 3 Investigating representational clusters** We begin to investigate representational clusters through various probing experiments to gain insight into the characteristics of representations and their formed clusters.

-
- **Chapter 4 Biclustering of representations.** Finding that standard clustering techniques are too restrictive in their handling of representations as a single unit, this chapter investigates biclustered representations. We also investigate whether the representational bicluster characteristics are able to predict generalisation.
 - **Chapter 5 Autoencoder representations** In the previous chapters we identified that the dimensionality of representations are excessively high, complicating our ability to extract meaningful insights from the data. This chapter introduces a novel approach to use an autoencoder (AE) to reduce the dimensions of the representations prior to clustering. We again investigate whether the autoencoder latent dimension is predictive of generalisation through the capturing of key features essential to the models' classification abilities.
 - **Chapter 6 Conclusion** This chapter concludes the study by summarising the observations and key findings. We also identify future study directions.

1.12 Conclusion

The generalisation ability of DNN has been studied from many perspectives (Section 1.3.2) including limited studies on the effects that representational clustering has on generalisation (Section 1.4). We aim to investigate the implicit forming of clusters in the representations of DNNs and the effect that these clusters have on the generalisation ability of the network. To our knowledge, this has not been investigated yet.

Chapter 2

Methods and techniques

“One of the things Ford Prefect had always found hardest to understand about humans was their habit of continually stating and repeating the very very obvious.”

- Douglas Adams, The Ultimate Hitchhiker’s Guide to the Galaxy Page 35 [1]

2.1 Introduction

In this chapter, we provide background information on the methods and techniques used throughout this study. We begin with an overview of the specific types of DNNs employed, along with details on their training processes (Section 2.2). We then introduce our approach for extracting hidden representations from these networks (Section 2.3).

Furthermore, we include a review on representational dimensionality reduction (Section 2.4) since representations are typically high dimensional, followed by an overview of clustering methods and quality metrics relevant to this work (Section 2.5). Finally, we offer similar information on biclustering, covering both the techniques and evaluation metrics used in the remainder of this study (Section 2.6).

2.2 DNN training

In this section, we cover the essential components of our training approach. Specifically, we review different types of DNNs that will be essential to this study along with non-standard optimisation techniques used in the training of the DNNs. Additionally, we discuss the application of batch normalisation (BatchNorm) which plays a key role in one of our experimental settings.

2.2.1 DNN model types

The most simple form of DNNs is the multilayer perceptron (MLP) [6, ch. 5.1] consisting of an input layer, one or many hidden layers, and an output layer. Each layer consists of neurons that are fully connected to those in the layers above and below by a weight vector followed by an activation function which we discuss in Section 2.2.2. A bias node, which is equivalent to a feature of value one, is also added to each layer except for the output layer. The bias offers several benefits not addressed in this study but are detailed in [2], [7], [39].

Another popular architecture is the convolutional neural network (CNN), which is composed of multiple convolutional layers [7, ch. 10], [6, ch. 7]. Each convolutional layer applies a set of kernels across the input with a specified stride, capturing local patterns. In contrast to MLPs, the weights of a CNN is captured in the kernels, which is updated during the training process. Multiple convolutions can occur in parallel, generating multiple sets of individual hidden representations known as channels or feature maps. These feature maps provide a refined hierarchical understanding of the image content. Padding can be added to the channel before convolution to increase the spatial dimensions of the subsequent channel. However, to reduce spatial dimensions while maintaining critical features, pooling layers can be added throughout the network. Combinations of kernel size, padding, and stride can be employed to produce a channel of specified dimensions.

A CNN can function independently or with fully connected layers added after the last convolutional layer, which when followed by a softmax function can output class probabilities. Although CNNs are commonly used for image classification, they are also well suited for tasks such as object detection and image segmentation [7, ch. 10].

2.2.2 Optimisation of DNNs

In order to reduce the loss of a DNN, adjustments are made to the models' parameters through the use of optimisation techniques. During optimisation, the loss is measured using a loss function that quantifies the distance between the predicted target and the real target [6, ch. 3.1]. Popular loss functions include: Mean squared error (MSE) used for regression tasks, and cross-entropy which is often used in classification tasks [7, ch. 5]. During the training process, gradient descent is used to adjust the networks parameters by minimising the loss function iteratively to produce an output that matches the labelled examples. We make use of stochastic gradient descent (SGD) and Adam [40] as optimisation techniques. Due to their widespread use, we do not review them but information on them can be found in Prince [7, ch. 6]. We present some less conventional gradient descent-based optimisation techniques in Section 2.2.3, which we also use in this study.

To fully harness the potential of a model, a non-linear activation function is applied to each node to transform the output. The most popular activation function used in DNNs is the rectified linear unit (ReLU) [23] that can be defined as:

$$a(\tilde{x}) = ReLU(\tilde{x}) = \begin{cases} 0 & \tilde{x} < 0 \\ \tilde{x} & \tilde{x} \geq 0 \end{cases} \quad (2.1)$$

where $\tilde{x}$ is the pre-activation output of the node. Other activation functions can be found in Prince [6, ch. 5.1.2].

Each model consists of hyperparameters that need to be set before the model is trained. These hyperparameters can encompass any variable that needs to be set prior to training, such as the depth of a model, the optimisation technique, loss function, activation function, etc. None of the hyperparameters are updated during the training loop. Since these hyperparameters affect the performance of the model, they can be tuned through a hyperparameter search process to find the best hyperparameters [7, ch. 8.5]. Usually, the hyperparameters are evaluated based on the model's performance on a validation set which the model has not been trained on.

Regularisation techniques are one method used to improve the generalisation ability of a model [7, ch. 9] and typically describe any approach that achieves this objective. One of these methods is early stopping, where the training process is stopped once a predefined criterion is satisfied, preventing the model from overfitting. Another method which has a regularising effect is batch

normalisation, although it is primarily used as a way to stabilise training. We discuss Batch-Norm in more depth in Section 2.2.4. Other regularisation methods can be found in Prince [7, ch. 9].

2.2.3 Optimisers

This section presents optimisation methods that are less known in deep learning, highlighting techniques that deviate from traditional approaches to enhance model performance and convergence.

2.2.3.1 Dadapt

The choice of learning rate has a significant impact on how efficiently a model converges to the minimum point in the loss landscape. Selecting the optimal values can be a laborious task, particularly without reference points. D-Adaption (Dadapt) [41] is a hyperparameter free approach to automatically set the learning rate to achieve the optimal convergence rate for Lipschitz functions. This is achieved without using backtracking or line searches and without incorporating an additional function value or gradient assessments per epoch. The method adapts the step size of AdaGrad [42] within a weighted dual averaging framework, combined with a technique by the authors to maintain a lower bound on the distance between the initial point and any minimiser of the function. At each epoch, a new lower bound is constructed using empirical qualities and is used in the subsequent epoch if it is lower than the current lowest bound. The algorithm starts with a lower limit defined by the user, which prevents it from becoming negative.

Dadapt can improve the performance of both Adam [43] and SGD [44] optimisers. Some initial method adjustments allow it to integrate with these optimisers. It was tested on various machine learning problems, using an initial learning rate of 1.0, and was shown to be effective [41].

2.2.3.2 Prodigy

Prodigy [45] is an adaptation of Dadapt that aims to improve convergence rates by dynamically adjusting step sizes in an AdaGrad-like manner. This adaptive approach enables Prodigy to take larger step sizes when gradients allow, while still controlling the error term similarly to Dadapt. These adaptive step sizes contribute to Prodigy’s faster convergence compared to Dadapt. Additionally, Prodigy can be seen as a weighted implementation of Dadapt. Mishchenko et al. [45] claim that Prodigy adapts more quickly to the correct learning rate than other methods.

2.2.4 BatchNorm

Batch normalisation (BatchNorm) [46] is a popular technique used to normalise hidden activations when training DNNs. The goal of BatchNorm is to normalise each batch of activation values for some layer to a learnt mean and standard deviation during both training and inference. More precisely, for the activation values $\mathbf{x}^l$ in the layer l for a sample $\mathbf{x}$, the BatchNorm operation is defined as:

$$\text{BN}(\mathbf{x}^l) = \gamma \odot \frac{\mathbf{x}^l - \hat{\boldsymbol{\mu}}_{\mathcal{B}}^l}{\hat{\boldsymbol{\sigma}}_{\mathcal{B}}^l} + \boldsymbol{\beta}. \quad (2.2)$$

where $\hat{\boldsymbol{\mu}}_{\mathcal{B}}^l$ and $\hat{\boldsymbol{\sigma}}_{\mathcal{B}}^l$ is the *per feature* mean and standard deviation vectors calculated over a batch of samples $\mathbf{X}_{\mathcal{B}}$. The vectors $\boldsymbol{\gamma}$ and $\boldsymbol{\beta}$ are learnable parameters of the same dimensionality as $\mathbf{x}^l$.

In the case of CNNs, each channel is considered a ‘feature’, and the mean and standard deviation are calculated independently per channel. More precisely, the activation values for a convolution layer l for a batch of samples can be denoted as a 4D tensor such that $\mathbf{X}_{\mathcal{B}}^l \in \mathbb{R}^{N \times C \times H \times W}$. Here, N, C, H , and W correspond to the number of samples in the batch, the number of channels, and the height and width of the representation, respectively. The per-channel mean and standard deviation are calculated in the same way, but for a channel instead of a node.

The per-channel mean is then calculated for a specific batch $\mathcal{B}$ as:

$$\hat{\mu}_{\mathcal{B},c}^l = \frac{1}{N \cdot H \cdot W} \sum_{n=1}^N \sum_{h=1}^H \sum_{w=1}^W \mathbf{x}_{\mathcal{B},n,c,h,w}^l \quad (2.3)$$

where $\hat{\mu}_{\mathcal{B},c}^l$ is the mean for channel c at layer l over the batch $\mathcal{B}$. Similarly, the per channel

standard deviation ($\hat{\sigma}_{\mathcal{B},c}^l$) for channel c at layer l over the batch $\mathcal{B}$) is calculated as:

$$\hat{\sigma}_{\mathcal{B},c}^l = \sqrt{\frac{1}{N \cdot H \cdot W} \sum_{n=1}^N \sum_{h=1}^H \sum_{w=1}^W \left(\mathbf{x}_{\mathcal{B},n,c,h,w}^l - \hat{\mu}_{\mathcal{B},c}^l \right)^2 + \epsilon} \quad (2.4)$$

where ϵ is a small constant added for numerical stability.

Networks trained with BatchNorm show desirable properties such as faster convergence and better generalisation ability [47], [48]. Despite the success and widespread adoption of BatchNorm, the exact mechanisms by which BatchNorm achieves its performance remain unclear.

Originally, the success of BatchNorm was motivated by its reduction of ‘internal covariance shift’ [46], meaning that BatchNorm reduces the distributional differences between the inputs to different layers in the network. However, Santurkar et al. [48] show that even when the internal distributions are artificially shifted (by injecting noise after BatchNorm layers) there is no noticeable degradation in performance. Instead, they argue from an optimisation perspective, and conclude that BatchNorm drastically smooths the loss landscape.

From a similar perspective, Bjorck et al. [49] show that without BatchNorm, a small subset of the activations in the deeper layers grow uncontrollably large during training. They further show that BatchNorm prevents this phenomenon and allows for the use of larger learning rates that result in speedier convergence. Similarly, Li and Arora [47] show that an exponential learning rate schedule is made possible by BatchNorm.

Although the optimisation perspective is the dominant view, several other authors have also investigated the benefits of BatchNorm from alternative perspectives. These include the effect of BatchNorm on adversarial robustness [50], its effect on the density of linear regions in the input space [51], and the orthogonality of internal representations [52].

2.3 Hidden representations

In this section, we focus on the extraction of hidden representations and their properties, with particular attention to sparsity. We explore how hidden representations are obtained from the network and discuss the role of sparsity in these representations, highlighting its significance and impact on model efficiency and interpretability.

2.3.1 Definition and notation

As introduced in Section 1.4.1, hidden representations (sample activation values) are transformed versions of the input particular to that layer. To extract sample representations for experimentation, we pass each sample through the network and record the activation values of each layer after the ReLU activation function. This produces a representation $\mathbf{x}^l \in [0, \infty]$ for each sample in each layer of the network. In the case of an MLP this results in a 2D matrix $\mathbf{X}_{\mathcal{B}}^l \in \mathbb{R}^{N \times h}$ for a batch of samples, where N is the number of samples and h the number of nodes. In the case of a CNN, it results in a 4D tensor $\mathbf{X}_{\mathcal{B}}^l \in \mathbb{R}^{N \times C \times H \times W}$ for a batch of samples, where N is the number of samples, C the number of channels and H and W the height and width of the channel, respectively.

To analyse a certain dimension, the other dimensions (such as the layer or channels of the network) are flattened. Specifically, the tensor is unfolded into a 2D matrix where one dimension is the size of the element analysed, while the other dimension consists of the product of the remaining dimensions. For example, analysing the sample dimension gives a matrix of $\mathbf{X}_{\mathcal{B}_{\text{flat}}}^l \in \mathbb{R}^{N \times CHW}$. We refer to the individual points that a channel consists of as features of the channel ($\mathbf{X}_1^l \in \mathbb{R}^{C \times HW}$), when considering the representation of a single sample.

Hidden representations are considered to be ‘distributed’, or ‘entangled’, meaning that different elements of a single representation are used in combination to represent a complex concept. This is in contrast to the ‘grandmother cell hypothesis’. This hypothesis from neuroscience suggests that single human neurons can encode a complex concept and trigger when exposed to stimuli that represents a well-known concept, such as your own grandmother.

2.3.2 Sparsity

Representational sparsity refers to the extent to which a representation consists of values that are zero or close to zero [2, p. 251]. Glorot et al. [19] approach sparsity from a biological perspective, arguing the use of rectified neurons over sigmoidal neurons is inspired by neuroscience. In addition to the use of rectified neurons, they propose the use of L_1 regularisation on the activation values to promote sparsity as a mechanism for better performance. Sparsity has been attributed to benefits such as information disentanglement, efficient representation of

variable sizes, and representations being more linearly separable [19]. Representational sparsity are typically considered desirable in CNNs [53].

Pretorius et al. [54] investigated the effect of activation functions on MLPs by analysing network characteristics such as network sparsity. They found that networks trained with ReLU activation are more sparse on average than networks trained on sigmoidal activations. They also investigated the effect of BatchNorm on representation sparsity and found that BatchNorm-trained models tend to have *less* sparse representations. It was also found that the less sparse BatchNorm models tend to have a better generalisation performance, and it was suggested that sparsity is not necessarily an indication of good generalisation. This apparent contradiction is returned to in Section 3.4.

In a ReLU-activated network, any negative feature is assigned a value of zero, meaning the neuron does not ‘activate’ for that feature. This non-activation for negative features effectively induces sparsity, allowing the network to focus on positive activations that contribute to the output. For our purposes, we interpret the sparsity threshold to be zero, and use this throughout all our experiments.

2.4 Dimensionality reduction techniques

One of the advantages of working with DNNs is the ability to work with high dimensional data [2, ch. 1], but the results are not always interpretable due to the high dimensionality. Different tools can be used to reduce the dimensionality, which could allow for better interpretability as well as practical uses such as being less computationally intensive.

2.4.1 PCA

Principal Component Analysis (PCA) [55] is a dimensionality reduction technique that linearly transforms data to a new coordinate system by identifying the directions of maximum variance. This process involves computing the covariance matrix, which captures the relationships and variances between the variables, and then determining its eigenvectors. The principal components can then be ranked by the amount of variance captured by them. By selecting only the top n principal components, the variation of the data can be described with fewer dimensions.

The method retains most of the information from the transformed dimensions, making it easier to visualise and analyse while minimising information loss. PCA is widely used in data preprocessing, feature extraction, and noise reduction across various fields.

The only hyperparameter is the number of principal components that are selected. For visualisation purposes, this is often two or three. Otherwise, the number of principal components that retain the most variance, often 95% of the variance [56], are used. Other selection methods include the Kaiser-Guttman method [57]–[59] where components are retained if eigenvalues are larger than one, as well as the scree plot method [56], [60] where the eigenvalues are plotted against the number of components, the ‘elbow’ of the plot indicates the suitable components.

2.4.2 t-SNE

Another dimensionality reduction technique is t-distributed stochastic neighbour embedding (t-SNE) [61] which is a way to illustrate high-dimensional data in a lower-dimensional space while minimising information loss. Unlike other techniques, t-SNE emphasises local relationships, prioritising the grouping of similar points to reveal clusters and patterns within the data. To achieve this, t-SNE calculates a similarity score for each point based on its distance to all other points. These distances are then converted into similarity scores using a t-distribution, creating scaled scores by normalising each score relative to the total. The resulting matrix of similarity scores allows t-SNE to reveal clusters and patterns in the data, helping to group similar points closely in the lower-dimensional space making it especially effective for exploring complex patterns and similarities. Because of its ability to illustrate local structure clearly, t-SNE has become a popular tool in fields like machine learning and bioinformatics to visualise high-dimensional datasets.

t-SNE has a number of hyperparameters that need to be set beforehand. These include the number of components, the perplexity (number of nearest neighbours used in the calculations), the learning rate, metric by which the distance is calculated, and other initialisation and stopping criteria hyperparameters. Although t-SNE is useful as a low-dimensional visualisation tool, it does have some limitations. First, not all data can be dimensionally reduced with t-SNE [61], [62]. It is also computationally expensive, which only worsens when it is used for multiple epochs which is required when the data is high dimensional, in order to ensure convergence.

Furthermore, convergence is not guaranteed as it can get stuck in a local minimum [61]. It is also sensitive to hyperparameter changes, and the results are not reproduceable due to the random initialisation upon each usage.

2.4.3 Autoencoder

An autoencoder (AE) [63], [64] is an unsupervised learning technique [65] that efficiently compresses an input to its essential features and then reconstructs the original input from compression. This is done by encoding the input into a lower-dimensional representation before reconstructing the input from the representation. An autoencoder consists of three different parts: An encoder, bottleneck, and decoder. The encoder compresses the input to remove any noise in order to generate the latent dimension (bottleneck). This is done by multiple hidden layers, which progressively reduces the dimensionality. The latent dimension is usually smaller than the input. The decoder recreates the input using the latent dimension encoding. The decoder consists of hidden layers, which progressively increases the dimensionality of the encoding until it reaches the dimensions of the input.

Several different types of AEs exist for different tasks such as: dimensionality reduction (Undercomplete, sparse, and contractive), denoising data (denoising), and data generation (variational AEs). For this study, we only focus on undercomplete AEs. Undercomplete AEs constrain the number of nodes in each hidden layer, limiting the number of information that can flow through the network. This allows the model to only learn the most important attributes of the input data and how to reconstruct the original input from the encoding. Due to the non-linear nature of the model through the use of non-linear activation functions, it is able to capture non-linear relationships in the data [66]. An AE can also incorporate convolutional hidden layers, while the latent dimension only consists of nodes.

The encoder is a function h that maps an input $\mathbf{x}$ to a latent representation $\mathbf{z} = h(\mathbf{x})$. The decoder consists of a function g which maps the latent representation $\mathbf{z}$ back to a reconstruction $\mathbf{y} = g(\mathbf{z})$ of the input [65].

2.5 Clustering

Clustering is an unsupervised learning method that aims to form groups of similar elements based on different metrics. Methods can range from forming clusters by minimising or maximising a quality metric [67], using hierarchical methods that build a dendrogram to iteratively merge or split clusters based on a distance metric [68], or using linear algebra directly (eigenvalues and eigenvectors) to cluster [69].

2.5.1 Clustering methods

One of the most popular clustering methods is the k-means algorithm [67]. The method partitions the samples into k clusters placing each sample in the cluster whose centre is closest according to a distance measure. After each iteration, each cluster centre is repositioned to the mean of the points assigned to it. The process is iterated for several steps. Different variations of the method exist, such as k-means++ [70], Fuzzy C-means clustering [71], and k-medians clustering [72], all of which use the same principle of k-means clustering.

The k-means++ algorithm improves upon k-means with a more informed selection of the initial centroids. Fuzzy C-means allow data points to belong to multiple clusters, assigned via the similarity to the centroid. It focuses more on the membership relationship and the uncertainty of the data points. k-medians minimise the sum of absolute differences between the centroids and data points, which is a more robust approach. The number of clusters must be specified in advance for the k-means method, which often necessitates a hyperparameter search to determine the optimal number of clusters [67].

2.5.2 Cluster quality metrics

It is not always clear how well a clustering method performed or whether the optimal cluster configuration was reached. Cluster quality indexes have been introduced to measure how well the method conforms to some desirable properties and to be able to compare different models. A quality index can be measured internally and externally, where internal indexes measure compactness and separation between clusters, and external indexes rely on the true and optimal division of data into clusters.

The Davies-Bouldin index (DBI) [29] is a popular metric for evaluating cluster quality. It evaluates the average similarity between each cluster and its most similar cluster, offering an indication of how well-separated and cohesive the clusters are. It is calculated by comparing the average distance between the clusters and the size of the clusters. This method can be used to compare the relative appropriateness of different data partitions. Importantly, the index is not sensitive to the number of clusters or the specific partitioning of the data. The DBI is defined as:

$$DBI = \frac{1}{k} \sum_{i=1}^k \max_{\substack{j \in \{1, \dots, k\} \\ j \neq i}} \left\{ \frac{d(x_i) + d(x_j)}{d(c_i, c_j)} \right\} \quad (2.5)$$

where k denotes the number of clusters, i, j are cluster labels, $d(x_i)$ and $d(x_j)$ are the mean intra-cluster distance for the clusters i and j , respectively, and $d(c_i, c_j)$ is the distance between the cluster centroids. For each cluster i , its most overlapping cluster j is found and these ‘worst-case’ values are averaged across clusters to produce the final metric. The DBI can be used to measure the purity of the representational clusters based on the class to which the representation belongs. It is also the method used by Natekar et al. [18] (Section 1.4) in the Predicting Generalization in Deep Learning (PGDL) challenge [73] (Section 3.3.1) to predict generalisation.

2.6 Biclustering

Biclustering was introduced to allow the clustering of both rows and columns simultaneously [74] for the direct interpretation of patterns within a data matrix. It is especially popular in genomic research [75], [76] as it is able to cluster unlabelled genes and conditions at the same time. Given a matrix $\mathbf{A} \in \mathbb{R}^{X \times Y}$ where X is the number of rows and Y the number of columns, a bicluster is a submatrix $\mathbf{B} \in \mathbb{R}^{|I| \times |J|}$ consisting of the intersection of rows $I \subseteq \{1, \dots, X\}$ and columns $J \subseteq \{1, \dots, Y\}$.

2.6.1 Bicluster types

In this section, we investigate different types of biclusters that have been identified within datasets to identify the types that we might encounter within hidden representations. Identifying the specific types allows us to choose appropriate quality metrics with which to measure the characteristics of the biclusters.

Madeira et al. [77] identified different types of biclusters in four main categories: constant values, constant rows/columns, coherent values, and coherent evolutions. The first three categories, constant values, constant rows/columns, and coherent values, focus on specific numerical patterns found in the data matrix and can be found in the rows, columns, or both. The last category, coherent evolutions, aims to find consistent patterns of behaviour in the data over time. This is not applicable to our data, and we will focus only on the first three categories.

We start by defining a perfect bicluster as the submatrix $\mathbf{B}_{\text{perfect}}(I, J)$ where all values are constant values π . However, in practical applications, we do not expect the data to reflect a uniform value π . Instead, we expect variability within the data that leads to the emergence of two patterns, scaling and shifting [78]. A perfectly shifted (constant rows/columns) bicluster $\mathbf{B}_{\text{shifted}}(I, J)$ satisfies the properties:

$$b_{ij} = \pi + \alpha_i \quad (2.6)$$

for a constant row bicluster, and the properties:

$$b_{ij} = \pi + \beta_j \quad (2.7)$$

for a constant column bicluster, where π is the constant value in the bicluster, α_i is an adjustment for $1 \leq i \leq |I|$, and β_j is a adjustment for $1 \leq j \leq |J|$. Furthermore, the values of a perfectly constant scaled value bicluster $\mathbf{B}_{\text{scaled}}(I, J)$ satisfies the properties:

$$b_{ij} = \pi \times \alpha_i \quad (2.8)$$

for a scaled row bicluster, and the properties:

$$b_{ij} = \pi \times \beta_j \quad (2.9)$$

for a scaled column bicluster. Lastly a perfect coherent bicluster can be defined as a combination of rows and columns with the following properties:

$$b_{ij} = \pi + \alpha_i + \beta_j \quad (2.10)$$

$$b_{ij} = \pi \times \alpha_i \times \beta_j \quad (2.11)$$

These two patterns can occur on any of the three main bicluster categories we identified, allowing us to define the biclusters types related to our work: ‘Perfect’, ‘Shifted’, ‘Scaled’, ‘Row’, and ‘Column’ biclusters. The row and column biclusters are added as their own type, as they consist of coherent values with properties shown in Equations 2.10 and 2.11. An example of each of the bicluster types we use is presented in Section 4.2.3.1.

2.6.2 Biclustering methods

The primary objective of a biclustering algorithm is to identify constant values by reordering both rows and columns, thereby grouping similar rows and columns together. As mentioned in the previous section, there are numerous types of biclusters, each with their own characteristics. Each bicluster algorithm cluster different characteristics and in various ways, making it difficult to identify a single algorithm that can effectively capture all variations of biclusters. In this section, we investigate these various biclustering algorithms with the aim of selecting the most suitable one for our representations.

Biclustering algorithms fall into different categories as presented by Figure 2.1: Single bicluster (2.1a), exclusive row and column biclusters (2.1b), non-overlapping biclusters with chequerboard structure (2.1c), exclusive rows biclusters (2.1d), exclusive column biclusters (2.1e), non-overlapping biclusters with tree structure (2.1f), non-overlapping non-exclusive biclusters (2.1g), overlapping biclusters with hierarchical structure (2.1h) and lastly arbitrarily positioned overlapping biclusters (2.1i) [77]. All of the algorithms, except (2.1a), produce k biclusters where k is either found or defined beforehand. In this study, we focus only on the chequerboard type (c) where k is defined beforehand and each element is assigned to a single exclusive bicluster.

Kluger et al. [76] developed a biclustering algorithm that identifies the underlying chequerboard patterns (if they exist) in cancer datasets by exploiting relations between genes (rows) and conditions (columns) through partitioning algorithms that find latent classes. The algorithm they developed is based on two assumptions:

1. That two rows have similar feature values which may be difficult to observe due to noise, but averaging over different columns can help reduce this noise and reveal the similarity more clearly.
2. The patterns of row features for any two similar columns are expected to be correlated, and this correlation becomes more evident when the features are averaged over sets of features with similar values.

Initially, the algorithm normalises the data to eliminate systematic biases in the feature values. Specifically, the bistochastic normalisation method is used to iteratively normalise the rows and columns of the input matrix $A \in \mathbb{R}^{M \times N}$ until the sum of each row and column equals the same

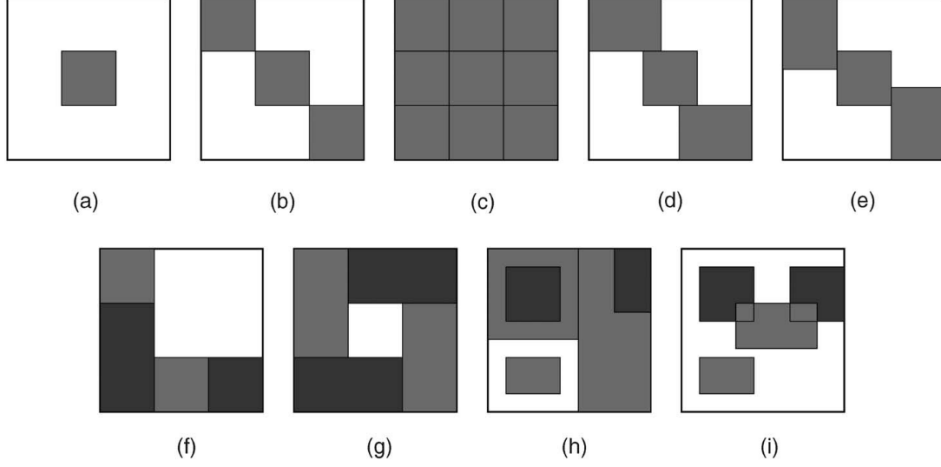


Figure 2.1: Different bicluster structures, (a) Single bicluster, (b) exclusive row and column biclusters, (c) Checkerboard structure, (d) exclusive rows biclusters, (e) exclusive column biclusters, (f) non-overlapping biclusters with tree structure (g) non-overlapping non-exclusive biclusters, (h) overlapping biclusters with hierarchical structure, and (i) non-structured overlapping biclusters.

Adapted from [77]

constant. Subsequently, the singular vectors of the normalised matrix A_n are computed using a method known as singular value decomposition (SVD). The SVD of A_n is expressed as:

$$A_n = U \Sigma V^T \quad (2.12)$$

where U is the orthogonal matrix with the left singular vectors of A_n , Σ is the matrix with the singular values of A_n on the main diagonal, and V^T is the orthogonal matrix of the right singular values of A_n .

Next, the first p singular vectors (where p is a hyperparameter) are ranked according to the Euclidean distance between the singular vector and its approximation. The singular vector approximation is found by clustering the elements of the singular vector using one-dimensional k-means and replacing each element value with the average cluster value. Of these p singular vectors, the best q right singular vectors (where q is a hyperparameter) are used to project the rows of A_n to a q -dimensional space. Similarly, the best q left singular vectors are used to project the columns of A_n to a q -dimensional space. Lastly, k-means clustering is used to obtain both the row and the column labels respectively [36], [76].

2.6.3 Bicluster quality metrics

As mentioned in Section 2.6.2, a bicluster is defined as $\mathbf{B} \in \mathbb{R}^{|I| \times |J|}$. Each element in the bicluster is represented by b_{ij} where $1 \leq i \leq |I|$ and $1 \leq j \leq |J|$. Therefore, b_{ij} is an element in a bicluster at the i^{th} row and j^{th} column. Subsequently, b_{iJ} refers to the mean element value of the row i and b_{Ij} refers to the mean element value of the column j .

As mentioned in the previous sections, there are many different types of biclusters with different characteristics. To measure the purity of the biclusters, different metrics can be used that account for some of these characteristics. We discuss the workings of the most prominent of these in this section, along with the types of biclusters for which they account. We specifically investigate mean square error (MSE), scaling mean squared error (SMSR), virtual error (VE), and virtual error transposed (VEt). We discuss selecting these metrics in Section 4.2.3.2.

2.6.3.1 Mean square error

MSR is one of the earliest used bicluster quality metrics [75] [79] and it evaluates the coherence of biclusters, by evaluating the distance of bicluster elements from different bicluster means, of the rows and the columns of a bicluster and can be defined as:

$$\text{MSR}(\mathbf{B}) = \frac{1}{|I| \cdot |J|} \sum_{i=1}^{|I|} \sum_{j=1}^{|J|} (b_{ij} - b_{iJ} - b_{Ij} + b_{IJ})^2 \quad (2.13)$$

The lower the MSR score the higher the coherence of the bicluster. A final MSR score can be calculated by taking the mean of all the MSR scores for each bicluster.

Due to the formulation of MSR, it favours more biclusters with fewer data points in each bicluster. It is dependent on the variance of the scaling coefficient (α_i) but not the variance of the shifting coefficient (β_j) within the bicluster. Therefore, shifting patterns will not affect the MSR score, while scaling patterns will [79].

2.6.3.2 Scaling mean squared residue

SMSR is similar to MSR but with the added benefit of recognising scaling patterns [80] [79]. It can be defined as:

$$\text{SMSR}(\mathbf{B}) = \frac{1}{|I| \cdot |J|} \sum_{i=1}^{|I|} \sum_{j=1}^{|J|} \frac{(b_{iJ} \times b_{Ij} - b_{ij} \times b_{IJ})^2}{b_{iJ}^2 \times b_{Ij}^2} \quad (2.14)$$

SMSR is not able to identify shifting patterns like MSR. A final SMSR score can be calculated by taking the mean of all SMSR scores for each bicluster.

2.6.3.3 Virtual error

VE [81] [79] creates a pattern from each bicluster to represent the general trends within it and measures the distance between each element of a bicluster and its pattern. The quality score depends on the way the tendency pattern is built. The pattern (P_i) defined as a collection of $|I|$ elements is given by the row means (b_{iJ}). The data must then be standardised to examine the degree of the similarity between columns. The standardised elements are defined by:

$$\acute{b}_{ij} = \frac{b_{ij} - b_{Ij}}{\sigma_{Ij}} \quad (2.15)$$

where σ_{Ij} is the standard deviation of column j . The pattern is also standardised to create a virtual pattern:

$$\acute{P}_i = \frac{P_i - \bar{P}}{\sigma_P} \quad (2.16)$$

where $\bar{P}$ indicates the mean of the pattern and σ_P denotes the standard deviation of the pattern. VE can be defined as the mean of the difference between the rows and the pattern values denoted by:

$$\text{VE}(\mathbf{B}) = \frac{1}{|I| \cdot |J|} \sum_{i=1}^{|I|} \sum_{j=1}^{|J|} (\acute{b}_{ij} - \acute{P}_i) \quad (2.17)$$

The lower the VE, the better the bicluster fit to the data. A VE score is calculated for each bicluster. To calculate the VE score for the entire data set, the average of all VE scores can be taken for a final score.

2.6.3.4 Virtual error transposed

The VEt is very similar to the VE score, but uses the transpose of the bicluster [82] [79]. The main difference to VE is that the virtual pattern (P_j) is given by the column means (b_{Ij}). The pattern is also standardised to create a virtual pattern:

$$\acute{P}_j = \frac{P_j - \bar{P}}{\sigma_P} \quad (2.18)$$

and the VEt is then denoted as:

$$\text{VEt}(\mathbf{B}) = \frac{1}{|I| \cdot |J|} \sum_{i=1}^{|I|} \sum_{j=1}^{|J|} (b'_{ij} - \acute{P}_j) \quad (2.19)$$

The closer the score to zero the better the fit of the bicluster pattern. VEt can detect both scaling and shifting patterns independently or a combination of the two. To calculate the VEt score of the whole data set, the average of the VEt scores can be taken.

2.7 Conclusion

In this chapter, we introduced the methods and techniques associated with this study. We discussed foundational topics such as the types of DNN models used as well as the methods employed for training these models. The process of extracting hidden representations was also covered along with representation characteristics such as sparsity. In addition, we examined various clustering techniques, including both single-axis clustering and biclustering, alongside methods for evaluating the quality of the resulting clusters.

All of the methods and techniques outlined here are used in the following chapters, starting with initial exploration of basic clustering in hidden representations in Chapter 3.

Chapter 3

Investigating representational clusters

“So once you do know what the question actually is, you’ll know what the answer means.”
- Douglas Adams, *The Ultimate Hitchhiker’s Guide to the Galaxy* Page 121 [1]

3.1 Introduction

In this chapter, we introduce hidden representations and investigate the characteristics of hidden representations obtained from models exhibiting different generalisation gaps. We identify two case studies where such generalisation gaps occur and analyse the representations from a clustering perspective. We first describe the process used to extract hidden representations (Section 3.2) before investigating the two case studies: one using selected models from the PGDL challenge (Section 3.3) and one comparing models developed with and without batch normalisation (Section 3.4). This chapter serves as a point of departure into the rabbit hole that is hidden representation clustering.

3.2 Extracting hidden representations

To extract the representations, we make use of the method introduced in Section 2.3.1. If not stated otherwise, 5 000 randomly selected samples from the training dataset is used. As other studies have used small sample sets when using PGDL models [13], [18], we consider this set of samples to be sufficient and representative of the dataset used. Furthermore, we use the same data seed for the models that are compared so that the same samples are selected and a fair comparison is made between the models. Specific layers are referred to by the type of layer, followed by the layer number. For example, the convolutional layer 1 will be referenced as `Conv_1` and the fully connected layer 1 as `Lin_1`.

3.3 Case study: PGDL Task 1

In this section, we explore the characteristics of hidden representations through basic probing experiments. This is an initial probing step to obtain insight into the representations and how they differ in models with different generalisation abilities.

3.3.1 PGDL challenge

The *Predicting Generalization in Deep Learning* (PGDL) challenge was a competition that was hosted at the international *Conference on Neural Information Processing Systems* (NeurIPS) in 2020 [73], [83]. The objective of the competition was for contestants to design robust and general complexity measures to rank models based on their generalisation ability. Participants were provided with various sets of trained models, which include all their parameters and training data, and they had to rank each of the sets individually according to their generalisation gap. Each metric was evaluated based on the rankings of a held-out set of models that the participants had no prior knowledge of.

The total set of PGDL models consisted of 550 trained models, divided into eight different tasks, each of which corresponds to a different architecture and dataset on which the models for that task were trained on. Six different image classification datasets were used, along with three different model architectures. The different architectures used were: a modified VGG [84] CNN

network, referred to as ‘VGG-like’, variants of Network in Network [85], and fully convolutional neural networks. Tasks 1 – 5 were given to the participants for development and Tasks 6 – 9 were held out as the testing set. An overview of each task is presented in Table 3.1.

Table 3.1: Overview of PGDL challenge tasks. Tasks 1 to 5 form the development set, and Tasks 6 to 9 form the held-out test set. There is no Task 3.

Task	Num models	Architecture	Dataset	Training samples	Test accuracy variation (%)
1	96	VGG-like CNN	CIFAR-10 [86]	50 000	66.83 - 86.34
2	54	Network in Network	SVHN [87]	73 257	70.32 - 95.05
4	96	Fully Convolutional	CINIC-10 [88]	36 000	53.85 - 67.87
5	64	Fully Convolutional	CINIC-10 [88]	36 000	25.33 - 66.71
6	96	Network in Network	Oxford-flowers [89]	2 040	58.43 - 73.93
7	48	Network in Network	Oxford-pets [90]	3 680	44.81 - 51.81
8	64	VGG-like CNN	FMNIST [91]	60 000	90.38 - 93.58
9	32	Network in Network	CIFAR-10 [86] (augmented)	50 000	84.36 - 93.06

Different hyperparameters were used for each tasks which resulted in the generalisation gaps presented in Table 3.1. Some of these hyperparameters include, but are not limited to: batch size, learning rate, architectural variation, weight decay, and dropout probability. All of the models were trained until they reached a specified training loss threshold, which halted the training process, leading to each model attaining a training accuracy of at least 99%. This allows for comparisons between models that have both fitted the training set but obtain varying scores on the validation set. This enables the exploration of model characteristics that enables the models to achieve improved performance on previously unseen data.

3.3.2 Models

We select trained models from the PGDL Task 1 to use in our experimentation process. Since the PGDL challenge was used as a benchmark for generalisation prediction specifically to evaluate state-of-the-art complexity measures (recall Section 3.3.1), it provides us with models with different architectures and different generalisation gaps. We specifically choose Task 1, as it consists of commonly used architectures (VGG-like CNNs) and dataset (CIFAR10) while having large differences in generalisation gaps between the models. In particular, the Task consists of two different architectures, one small and one large when referring to the number of convolutional layers in each. This allows us to analyse different depths of models, trained on a dataset which consists of more natural images with overlapping classes.

The architecture of the small models consists of 2 convolutional layers with [256, 256] or [512, 512] channels, respectively. There is also a 2×2 max pooling layer after the 2nd convolutional layer, with a stride of 2. The models also have one or two fully connected layers before the output layer of 10 nodes, which consists of 128 nodes. The large model architecture consists of 6 convolutional layers with [128, 128, 256, 256, 512, 512] or [64, 64, 128, 128, 256, 256] channels, respectively. There are 2×2 max pooling layer after the 2nd, 4th, and 6th convolutional layers. The architecture also consists of one or two fully connected layers with 128 nodes before the output layer. We provide a visual representation of the largest and smallest architectural variations in Figure 3.1.

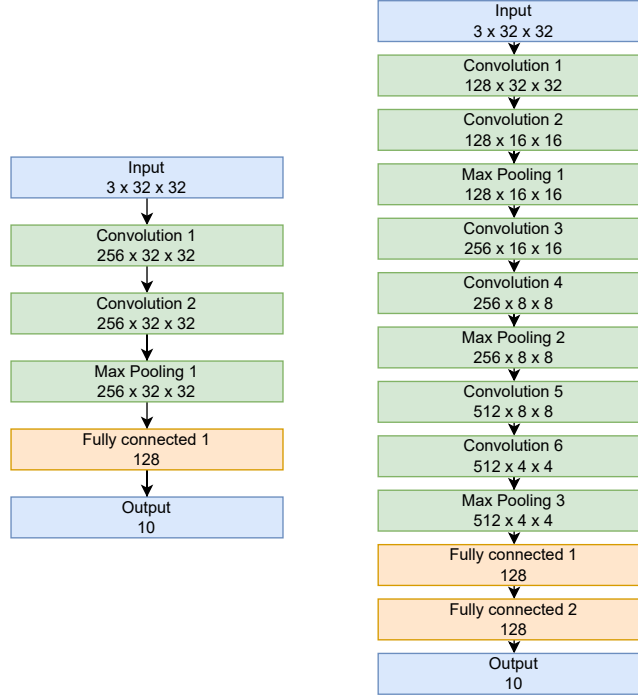


Figure 3.1: Layer structure of the small and large PGDL Task 1 models. Green blocks represent the convolutional/pooling representations (indicated as ‘ $Channels \times Height \times Width$ ’) while orange blocks represent the fully connected layer representations.

Due to the PGDL challenge not having a validation dataset, our choice of models relies on their performance on the evaluation set. (The models have fitted all of the training data.) For both architectures, we select a model with a high evaluation accuracy and a model with a low evaluation accuracy within the task, which should provide us with a sufficient generalisation gap. We choose four different models, two with a small architecture, ‘Good Small’ (GS) and ‘Bad Small’ (BS), and two with a large architecture, ‘Good Large’ (GL) and ‘Bad Large’ (BL).

Of these two architectures, one model that generalises well (GS, GL) and one that generalises poorly (BS, BL) on the test data is selected.

For the small architecture, we select two models with the exact same setup (number of channels and fully connected layers) with a large generalisation gap difference. This implies that they are not necessarily the best and worst performing models across all small architecture models, but two that are easily comparable and still maintain a large difference in performance. Similarly, for the large architecture we select two models with the exact same setup, again with a large generalisation gap difference. The selected models along with their respective accuracies are presented in Table 3.2. It is worth noting that despite GS being the best of the smaller architectures and BL being the worst of the larger architectures, both exhibit relatively similar evaluation accuracies, with GS slightly outperforming BL.

The models make use of the following shared hyperparameters: A ReLU activation function that is applied after each hidden layer, cross-entropy as the loss function, and SGD as the optimiser for both architectures. A learning rate of 0.001 and a Nesterov momentum of 0.9 are used. The variations in training hyperparameters that are specific to each model are provided in Table 3.3. This includes each models' batch sizes, weight decay, and dropout probability. The models were trained for 1 200 epochs or until interpolation is reached on the training data (reaches 100% training accuracy). Not all experiments will include both model architectures as the small model is sufficient in some probing experiments, and the analysis of the large models is computationally expensive.

Table 3.2: Chosen PGDL Task 1 models performance metrics

Model	Train Accuracy (%)	Evaluation Accuracy (%)	Generalisation
Model 20	99.97	73.30	GS (Good small)
Model 90	100.00	66.83	BS (Bad small)
Model 667	99.94	69.28	BL (Bad Large)
Model 669	99.99	85.97	GL (Good Large)

Table 3.3: Chosen PGDL Task 1 models' corresponding model-specific training hyperparameters.

Model	Reference	Layers	Batch Size	Drop probability	Weight Decay
Model 20	GS	2	32	0.5	0.001
Model 90	BS	2	512	0.0	0.000
Model 667	BL	6	512	0.0	0.001
Model 669	GL	6	8	0.5	0.001

3.3.3 Activation distributions

Before using any clustering methods, we start by observing the distribution of the activation values for each layer. For this, the selected PGDL models are used to see if there are differences in the activation values between a good and a bad generalising model. The hidden representations ($\mathbf{X}_B^l$) of each model are extracted as described in Section 2.3.1. For each of the network layers, we flatten the activation tensors of each sample to bin the individual features of the channels. This results in a 2D matrix of our representation with dimensions of $\mathbf{X}_{\text{flat}}^l \in \mathbb{R}^{N \times CHW}$. Each feature is placed in one of the 25 bins that span the entire range of activation values. The bins are automatically assigned according to the distribution of the activation values, except for the first bin. The first bin contains all of the zero-valued activations since they signify negative-valued activations in a ReLU activated network, and are therefore analysed separately. The distribution of the feature values is presented in Figure 3.2.

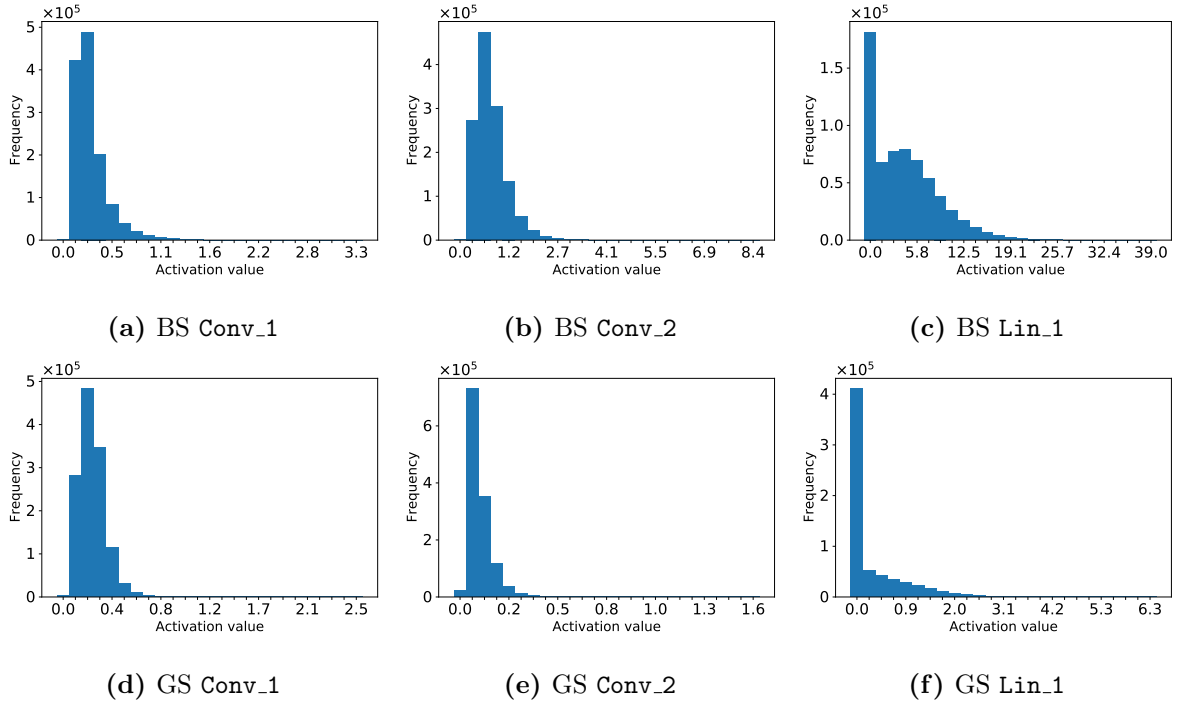


Figure 3.2: Distribution of feature activation values of each of the small models (BS and GS) for 5 000 samples across the two convolutional layers and the fully connected layer.

For both GS and BS, the median activation values are close to 0, indicating that most activation values are very small in the convolution layers and 0 in the fully connected layers, except in the case of BS Lin_1 where the mean activation values are much larger compared to the other

layers. We also note a difference in the spread, as well as the scale of the activation values, when comparing BS with GS. Specifically, in both the convolutional layers of GS and BS the distribution of activation values is closer to zero whereas the activation values in `Lin_1` of BS are much more spread out compared to GS. Additionally, it is noted that within the BS model, the activation values become much larger as they advance through each layer. However, the GS model maintains a much more consistent scale for all its activation values across the different layers. The GS model also produces significantly more 0 valued feature activations than the BS model, especially in `Conv_2` and `Lin_1`.

Examining Figure 3.2, we can speculate that since GS has more sparse activation values, few features contribute more to the output of the model, whereas the BS model has more features that contribute to the output. We revisit this observation in Section 3.4.4.

3.3.4 Low-dimensional perspective

From the observations in Section 3.3.3, there appear to be some differences between GS and BS in the hidden representations. A common method used in data science for data visualisation is using t-SNE which is a dimension reduction tool (Section 2.4.2). Due to the non-linear reduction of the dimensions, the pairwise similarities between data points are preserved in the lower-dimensional space, providing an indication of the relationships among the high-dimensional representations. The goal of this experiment is to see if any explicit clusters form in the underlying data and to compare the lower-dimensional representations of both GS and BS models. We make use of the t-SNE implementation provided in the Scikit-Learn library [36].

The flattened activations ($\mathbf{X}_{\text{flat}}^l \in \mathbb{R}^{N \times CHW}$) as used in Section 3.3.3, are repurposed. We perform a hyperparameter sweep on the perplexity and the number of iterations to obtain the best low-dimensional representation of the activations by visual inspection. This is done for each of the models separately, applying each of the values to all layers of the model. All other hyperparameters are left at their default value as they do not affect the results as much as the perplexity and iterations. For our large hyperparameter sweep, we use 5 000 iterations for each of the perplexity values to reduce the computational cost. This is deemed as sufficient iterations to be able to choose three potential perplexity values for each model. Another sweep is performed using the three selected perplexity values for each model but train for 15 000 iterations to ensure

convergence. Finally, we visually assess the results to determine the best performing models.

In order to present the representations on a 2D plane, they are reduced to only two dimensions. This allows us to inspect the inherently formed clusters due to similar points moving together and dissimilar points moving away from each other, as well as the reduction of dimensions. We choose the initial perplexity values as $[2, 5, 8, 10, 15, 20, 30, 40, 50, 75, 100, 200, 250, 300]$. In the second sweep, the BS model uses the values $[200, 250, 300]$, while the GS model uses $[8, 20, 40]$. The resulting plot for each layer is then taken into account when selecting the optimal perplexity values. The final plots shown are generated using a perplexity of 250 for the BS model and 40 for the GS model. Each of the points is assigned a colour according to the class that the sample belongs to. The results are presented in Figure 3.3, while we present those of GL and BL in Appendix A.1.1.

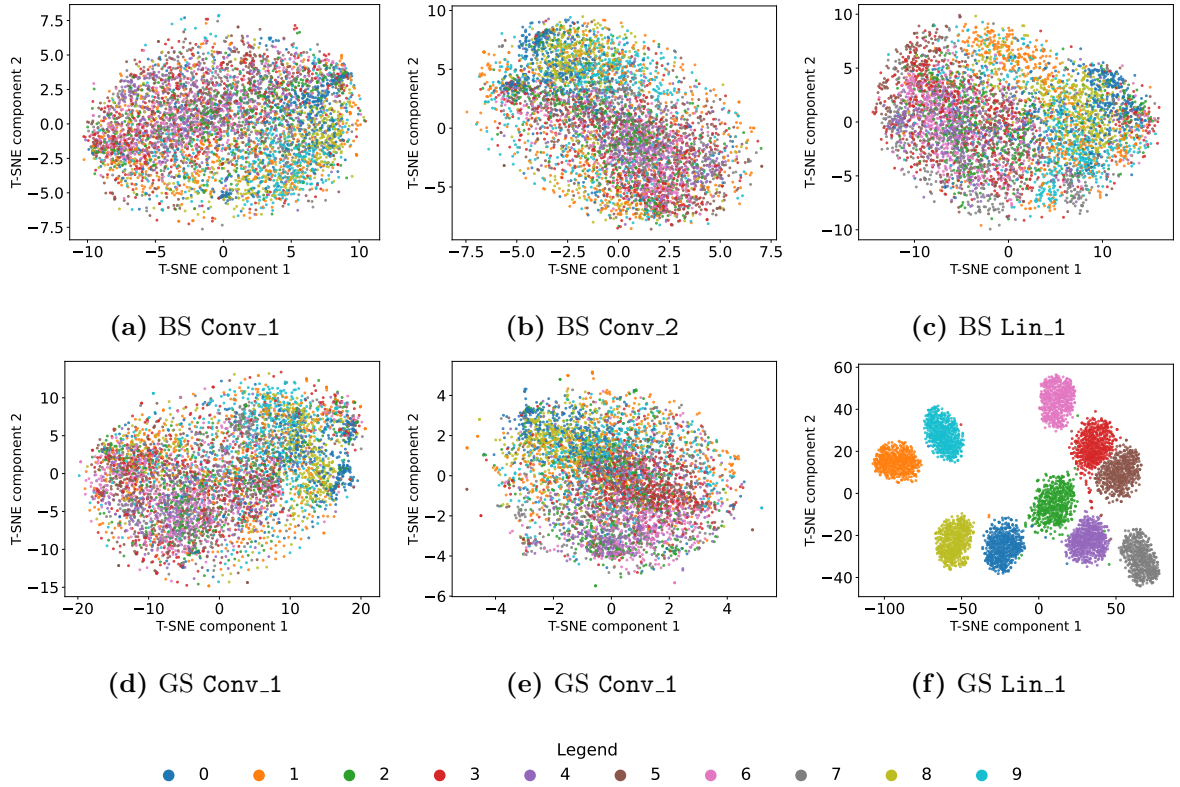


Figure 3.3: Lower dimensional views of BS (a, b and c) and GSs' (d, e and f) hidden representations of each layer reduced by t-SNE to two dimensions. Colours indicate the true class label of each sample point.

Starting with an overview of the convolutional layers, for both GS and BS, there are no clear class clusters that are formed (Figures 3.3a, b, d, and e). However, we do observe that there

are well separated class clusters that form in the linear layer of GS (Figure 3.3f) but not in BS (Figure 3.3c). An interesting observation is that there are some sample points in Figure 3.3f that belong to a different class than the cluster to which it is closest. This may occur if the sample is mislabeled [92], resulting in it being a member of the appropriate cluster but inaccurately coloured.

An important observation is that the representation of GS `Lin_1`, when dimensionally reduced with t-SNE, can easily be separated into the different classes, whereas the `Lin_1` representation of BS could not. It is possible that the features, particularly in the convolutional layers, could possess high-dimensional interrelationships not easily visualised with dimensionality reduction alone. One key takeaway is that these representations are high dimensional, making them challenging to work with. As this was only an initial probing step, the rest of the study will focus only on the convolutional layers as this is our main interest.

3.3.5 Channel-sample activations

While Section 3.3.4 proved useful as a probing step, we now turn our attention back to internal clusters formed by networks within the higher dimensional convolutional layers. The objective of this section is to observe the way in which a model internally clusters samples of different classes based solely on their activation values. We aim to compare models with different generalisation capabilities, regarding their internal clustering ability, to gain insights into performance differences. Intuitively, we expect a difference between the models, evident from the variation in the activation value sizes for the same sample, as observed in Section 3.3.3.

To analyse the internal clusters, the representations are reduced to $\mathbf{X}_{\text{flat}}^l \in \mathbb{R}^{N \times C}$ in order to have a single activation value per channel. This is achieved by taking the average of the elements in the feature map across the height (H) and width (W) for each channel, expressed as:

$$\mathbf{X}_{n,c}^l = \frac{1}{H \cdot W} \sum_{h=1}^H \sum_{w=1}^W \mathbf{X}_{n,c,h,w}^l \quad (3.1)$$

where $\mathbf{X}_{n,c,h,w}^l$ denotes the element (feature) of the tensor $\mathbf{X}^l$ at position (n, c, h, w) , and $\mathbf{X}_{n,c}^l \in \mathbb{R}^{N \times C}$ is the resulting 2D tensor. Each element represents the average value of the feature map over the spatial dimensions (H and W) for each channel (C) and sample (N). For each class, the number of samples for which the channel is ‘active’ for is counted, which is an integer value

greater than or equal to 0. This is referred to as the ‘activation frequency’. We sort the channels in ascending order based on the average activation frequency for all classes for each channel. This is shown in Figures 3.4 and 3.5. Due to the random selection of the samples from a balanced dataset, each class has approximately 500 samples.

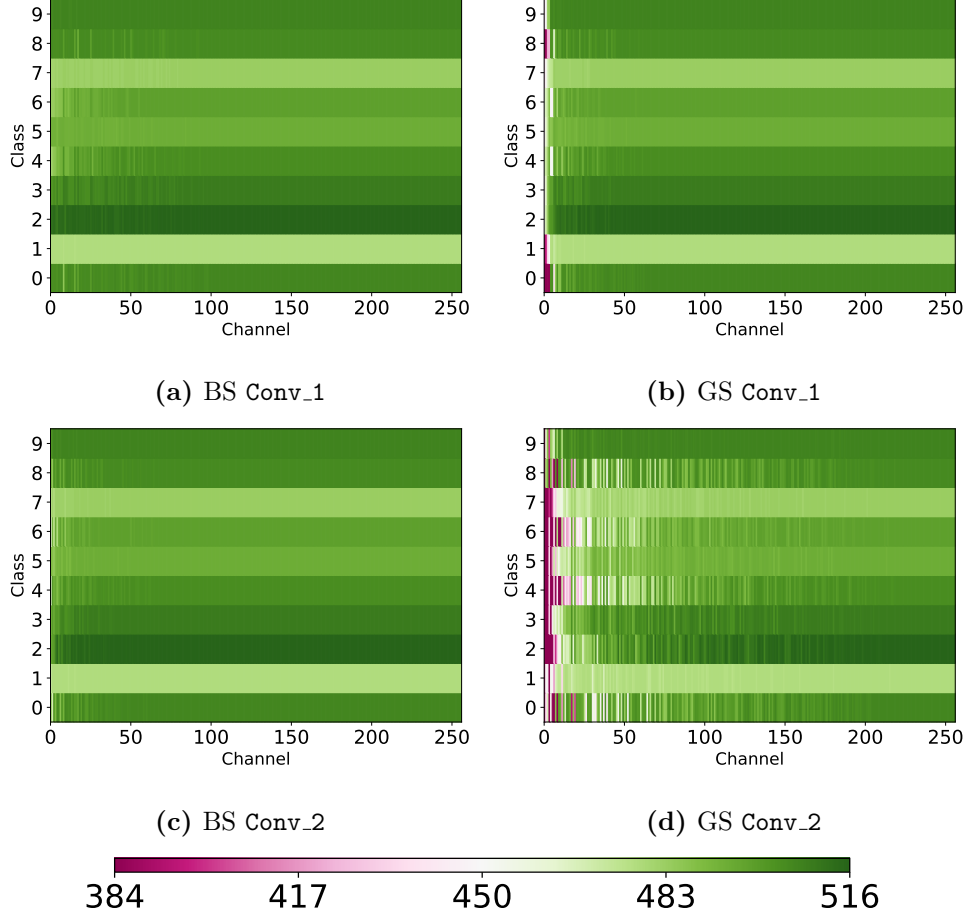


Figure 3.4: The activation frequency per channel and class, for the convolutional layers of BS (a and c) and GS (b and d). The channels are arranged in ascending order from lowest average activation frequency over classes

In Figures 3.4 and 3.5 we show activation frequency per channel and class. Figure 3.4 shows the convolutional layers of BS and GS, while Figure 3.5 shows the fully connected layers. For the convolutional layers, we adjust the scale to differentiate among the upper quartile activation frequencies only ($384 \rightarrow 516$), since few channels are active for less than 284 samples. (These are all indicated with the same colour). This adjustment allows us to make a clear distinction between the GS and BS models, as they closely resemble each other when using the full scale. For the fully connected layers, we use the full scale, as there is a clearer distinction between BS

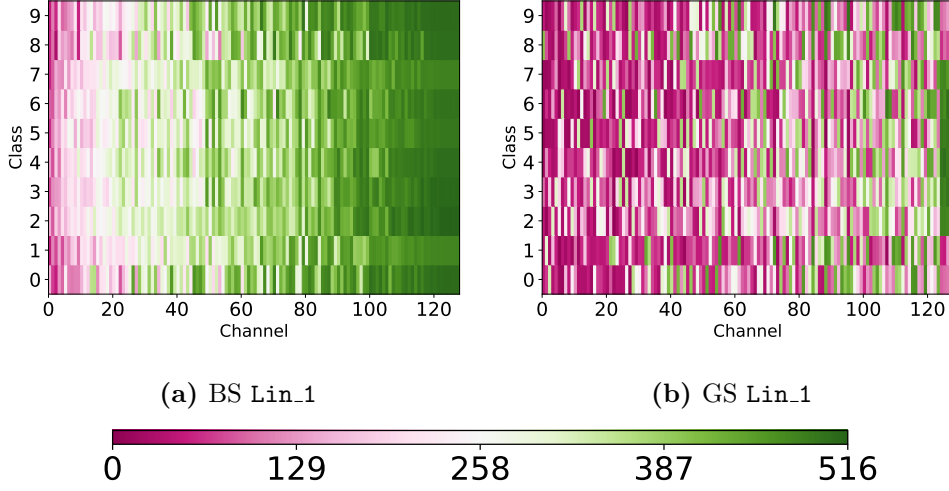


Figure 3.5: The activation frequency a channel of the network is active for, for the convolutional layers of BS (a, c) and GS (b and d) for each class. The channels are arranged in ascending order from lowest average activation frequency over classes

and GS.

We first explore the overview of the channel activations without taking into account the class labels. Following the results observed in Section 3.3.3, it is evident that GS again has slightly more sparse activations than BS when only considering the convolutional layers of Figure 3.4. There is also an increase from `Conv_1` to `Conv_2`, for channels that activate for no samples. It is noticeable that the sparse activations are mainly contained to only a few channels and not distributed, when looking at Figure 3.4d. For the convolutional layers of the BS model, almost all channels are active for all samples, with a few channels less active than the rest.

The differences between BS and GS become clearer when comparing the fully connected layers in Figure 3.5. While BS `Lin_1` exhibits sparser channel activations than the preceding layers, it remains less sparse compared to GS. In Figure 3.5a, the results for the BS model demonstrate a gradual change in the sparsity of channel activation. Although many channels are active for nearly all samples, approximately half of the channels activate only for a few to almost half of the sample set. Observing Figure 3.5b almost all channels are inactive for most samples, with a few selected channels being active for half to almost all samples in the set.

When considering class-specific activations, a distinction is made between activations from different classes. Examining the convolutional layers in Figure 3.4, focusing solely on class-specific

activations, both models seem to activate similarly across various classes, especially for classes 1, 2, 7, and 9. Both class 1 and 7 seem to consistently have fewer active channels, which could indicate that the channels activate for specific object features, potentially absent in some samples but found in others. When considering the fully connected layer in Figure 3.5 there are slight distinctions between the classes, but it is difficult to make clear distinctions in either BS or GS. The channels appear to activate more specifically for certain classes, with some channels activating more for particular classes and less for others.

3.3.6 Clustered representations

We now take the next step and utilise clustering techniques to investigate hidden representations and the internal clusters that potentially form during training. We consider clustering from both a class-based approach (Section 3.3.6.1) and a class-agnostic approach (Section 3.3.6.2).

The same representations ($\mathbf{x}^l$) are used as in Section 3.3.5. Due to the high dimensionality of the representations and the associated computational complexity, we reduce the representations to $\mathbf{X}_{\text{flat}}^l \in \mathbb{R}^{N \times C}$ according to Equation 3.1. After reducing the representations, we normalise them, as a scale difference was previously observed between the representations of the models (Section 3.3.3), which could impact the results when comparing the models. To eliminate this variable, each sample representation is normalised by dividing it by its Euclidean norm to produce the representation $\mathbf{x}^l$ to be $\in [0, 1]$, which will be used from here onwards. This experimental setup is used for both class-based and class-agnostic clustering.

3.3.6.1 Class-based channel-sample clusters

For the class-based approach, no clustering techniques are applied to the representations, instead we analyse the clusters naturally formed by the representations of each labelled class. In this context, the DBI of the layer representations is measured by using the labels of the sample classes as the cluster labels, following the method of Natekar and Sharma [18] (Section 2.5.2). We show these results in Figure 3.6.

Observing the class-based clustering presented in Figure 3.6, it is evident that the GS and GL models consistently have a lower DBI score than the BS and BL models across all layers. In

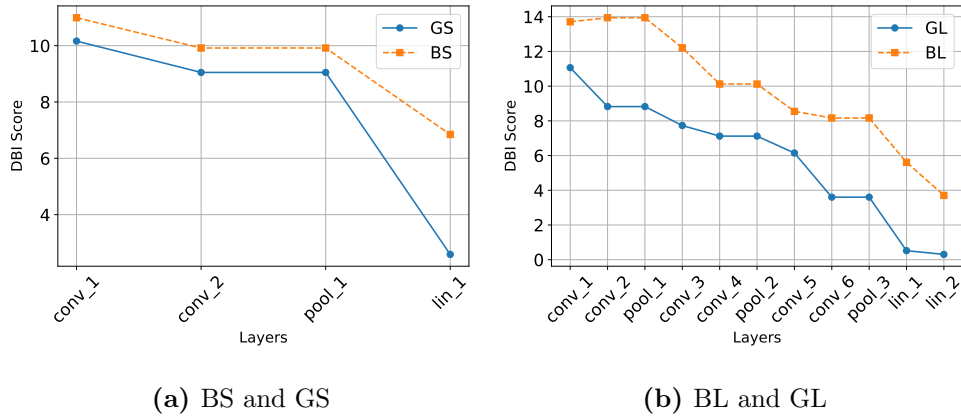


Figure 3.6: Class-based DBI score per layer, for both the good (blue) and bad (orange) models. A lower DBI indicates higher cluster purity.

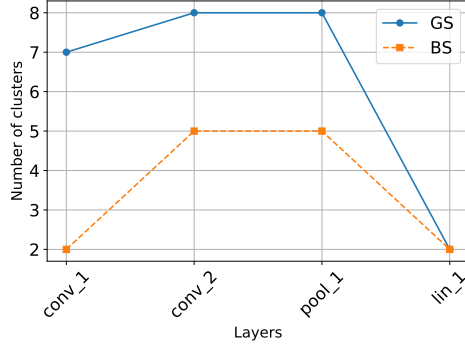
particular, both the GS and GL models demonstrate a significantly lower DBI score specifically in the fully connected layers compared to their counterparts. Each model exhibits a high DBI score in the initial layer, with the DBI progressively decreasing with each subsequent layer. This is indicative of the clusters becoming more pure, whereby the clusters formed in the fully connected layers, especially in GL, become nearly completely separated, suggesting that the clusters are increasingly defined by classes.

We can conclude that measuring the DBI purely on the representations, with the class label of each sample representing the corresponding cluster, shows that the better generalising models produce purer internal clusters than their counterparts. This is observed for both the small and large architectures. Furthermore, it is clear that as the samples traverse the model, the distinction between the label classes becomes more evident and pronounced.

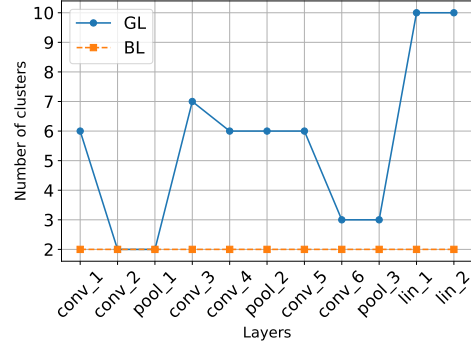
3.3.6.2 Class-agnostic channel-sample clusters

For the class-agnostic approach, the k-means clustering algorithm is used to cluster similar representations and provide cluster labels for each representation. Given that k-means does not determine the optimal number of clusters on its own, we use different numbers of clusters ranging from 2 to 15 and select the number of clusters that results in the lowest DBI score.

In Figure 3.7 we present the number of optimal clusters for each layer, while Figure 3.8 presents the DBI scores produced by the number of clusters at each layer. An interesting observation

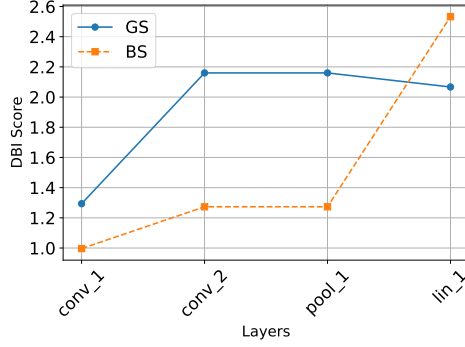


(a) BS and GS

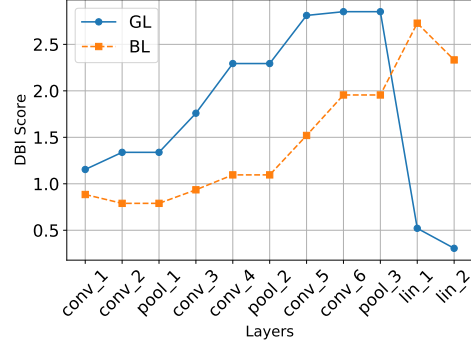


(b) BL and GL

Figure 3.7: The optimal number of class-agnostic clusters per layer for both the good (blue) and bad (orange) models, selected by the lowest DBI score.



(a) BS and GS



(b) BL and GL

Figure 3.8: Class-agnostic DBI score per layer, for both the good (blue) and bad (orange) models. A lower DBI scores indicate higher purity.

is that BL has no layers with any other number of optimal clusters other than two. We also observe that in Figure 3.7a, BS only has two optimal clusters for both `Conv_1` and `Lin_1`, while the number of optimal clusters for GS tends towards the number of classes in the dataset. The GL model lacks a stable count of clusters within the convolutional layers, but notably, it transitions to using 10 clusters for each of the fully connected layers. Interestingly, this aligns with the number of classes present in the dataset, suggesting that the GL model effectively differentiates among them. This implies that – in both the small and large setups – clearer representational clusters can be identified in the better performing models.

Observing the DBI scores in Figure 3.8, the opposite trend to that seen in Section 3.3.6.1 is evident, with the BS and BL models having purer clusters compared to the GS and GL models,

with the exception of the fully connected layers. In the fully connected layers, both GS and GL produce lower DBI scores than their counterparts. There is also a trend in the purity scores for both architectures, where the DBI score increases in the convolutional layers from layer to layer. (This trend does not hold for the fully connected layers.)

3.3.7 Discussion

In this section, we performed initial probing experiments to broaden our knowledge of the characteristics of hidden representations. We started off by investigating the activation distributions of models with a generalisation gap, after which we visualised the dimensionality-reduced representations. We also investigated the activation patterns of the channels within the layers of a network. Lastly, we investigated the clusterability of the representations with the use of both class-based and class-agnostic clustering methods.

We found that there appear to be differences in the activation patterns between the good and poorly generalising models, the most prominent being the scale difference in the activation values as observed in Figures 3.2c and f. To fairly compare the models from here on we decided to normalise the representations by dividing each representation with its Euclidean norm.

Furthermore, we found that the better generalising models appear to have sparser representations in the fully connected layers, indicating more specific activations for certain samples, which was observed in Figure 3.2f and in Figure 3.5b where most of the channels did not activate for most of the samples.

Comparing models with different generalisation abilities by class-based cluster purity, we observe similar results to what Natekar et al. [18] observed with better generalising models producing purer internal clusters. This did not hold for the convolutional layers when comparing class-agnostic clusters, but did hold for the fully connected layers. Another interesting observation is the number of class-agnostic clusters that formed. Most of the layers only produced two clusters, which should produce less pure clusters, since there are more dissimilar values grouped together. However, we observed that these clusters (mostly in the poorer generalising models) were more pure compared to the multiple clusters of the better generalising models. We also observed that for larger architecture models, GL produced 10 class-agnostic clusters in the fully connected layers which were more pure than the two of BL.

3.4 Case study: batch normalisation

BatchNorm, as introduced in Section 2.2.4 is a popular layer normalisation technique used when training deep neural networks and has been shown to enhance the training speed and accuracy of the models. However, the mechanics by which BatchNorm achieves these benefits is an active area of research, and different perspectives have been proposed [50]–[52].

In this section, we investigate the effect of BatchNorm on the resulting hidden representations. Specifically, the sparsity of these representations is considered, as well as their implicit clustering – the creation of groups of representations that are similar to some extent. We contrast image classification models trained with and without batch normalisation and highlight the consistent differences observed.

We select these two elements, specifically:

- Representational sparsity is generally considered a desirable property for DNNs, and is considered to be one of the reasons for the performance of ReLU activation functions [19]. Although the sparsity of neural network parameters (parametric sparsity) is well studied, the sparsity of the internal representations (representational sparsity) is less explored [2, p. 251]. The effect of BatchNorm on representational sparsity has been studied to a limited extent for MLPs [54] but no such analysis has yet been performed for CNNs.
- The clustering characteristics of representations have been shown to be a promising avenue to better understand generalisation in DNNs [15], [16], [18]. As such, comparing the clusters formed by models trained with and without BatchNorm could provide insight into how BatchNorm increases generalisation performance.

In summary, the objective of this section is to compare models trained with and without BatchNorm and to analyse the effect of BatchNorm on the hidden representations. Due to this section focussing on the effect of BatchNorm on the convolutional layers of the networks, the fully connected layers will not be analysed.

3.4.1 Experimental setup

In this section, we describe the approach used to investigate the effect of BatchNorm on CNN representations. We select a set of CNN architectures and datasets and develop comparative models. Sparsity metrics and techniques to evaluate the cluster characteristics of representations are also developed.

3.4.1.1 Models

We consider two different datasets and network architectures for experimentation. For datasets, both MNIST [32] and CIFAR10 [33] (as introduced earlier in Section 3.3.2) are selected. MNIST, while simple, is useful for probing generalisation from a clustering perspective as it consists of well-separated classes. The analysis of a probing dataset (MNIST) along with a more natural dataset (CIFAR10) allows us to verify whether the results hold for both datasets. We select architectures that match the datasets: a simple CNN with 4 convolutional layers trained on the MNIST dataset and a standard, deeper architecture with 13 convolutional layers trained on the CIFAR10 dataset. These models resemble those introduced in Section 3.3.2, but have additional convolutional and pooling layers compared to the illustration in Figure 3.1. For each of these architectures, a model is trained with and without BatchNorm using 4 different random initialisation seeds, each.

MNIST

For the MNIST dataset, we make use of an architecture similar to the ‘standard architecture’ described by Nakkiran et al. [10]. This 5-layer CNN consists of 4 convolutional layers with [16, 32, 64, 128] channels, respectively. There are also max pooling layers after the 2nd and 3rd convolutional layers with a stride of 2. The architecture is altered by adding a fully connected layer of 100 nodes preceding the output layer of 10 nodes, such that it is more similar to the models we use with the CIFAR10 dataset (discussed shortly). This contrasts to the architecture of Nakkiran et al. [10] which only has a single fully connected output layer of 10 nodes.

A ReLU activation is applied after each hidden layer, with cross-entropy as loss function. The hyperparameters are chosen based on a hyperparameter sweep that tracks the validation accuracy at the point of interpolation. We perform a grid search for both the Adam and SGD

optimisers with learning rates of [0.1, 0.01, 0.001, 0.0001] and a learning rate schedule that multiplies the learning rate by 0.99 every 10 epochs. Also included is a Nesterov momentum of 0.99. Furthermore, batch sizes of [32, 64, 128] are used and training is carried out for 1 000 epochs. The training is terminated once the network interpolates the training data (reaches 100% training accuracy).

The hyperparameters that performed the best include a batch size of 64, a learning rate of 0.01, and SGD as optimiser for both the BatchNorm and non-BatchNorm models. The accuracy achieved during training is shown in Table 3.4. Both sets of models interpolated on the training data with the BatchNorm models performing a bit better on the validation and test sets than the non-BatchNorm. The variation between seeds is minimal for both the BatchNorm and non-BatchNorm models.

CIFAR10

For the CIFAR10 dataset, we rely on the well-known VGG-16 [34] architecture. This architecture consists of 13 ReLU-activated convolutional layers with [64, 64, 128, 128, 256, 256, 512, 512, 512, 512, 512, 512] channels and a kernel size of 3×3 . There are also 2×2 max pooling layers after the 2nd, 4th, 7th, 10th, and 13th convolutional layers with a stride of 2. We do not use the hyperparameters from the original implementation of Simonyan and Zisserman [34] but rather perform a hyperparameter sweep and select the models based on performance on the validation set. This is done to achieve as close to current benchmarks on the dataset and architecture as possible. Specifically, the cross-entropy loss in combination with the Adam optimiser is used. An initial learning rate of 0.001 for the models with BatchNorm and a smaller learning rate of 0.0001 for those without is used. The learning rate is multiplied by 0.99 after each epoch, and the momentum and stability terms of Adam remain at the default values. The same batch size of 64 is used for both sets of models. We train for 300 epochs using early stopping with a patience of 20% on the validation error.

The accuracy for both sets of models is presented in Table 3.4. Both sets of models achieved near interpolation on the training data, with the BatchNorm models significantly outperforming the non-BatchNorm models on the validation and evaluation sets. Additionally, BatchNorm models exhibited minimal variation between seeds, whereas the accuracy of the non-BatchNorm models showed much greater variability across different seeds on the unseen data.

Table 3.4: Train, validation, and evaluation classification accuracy for the BatchNorm (BN) and non-BatchNorm (NBN) models on MNIST and CIFAR10. $\pm$ indicates the standard deviation across 4 random initialisation seeds.

Model	Training Acc (%)	Validation Acc (%)	Evaluation Acc
CNN MNIST BN	100.000 $\pm$ 0.000	99.355 $\pm$ 0.010	99.198 $\pm$ 0.100
CNN MNIST NBN	100.000 $\pm$ 0.000	99.250 $\pm$ 0.087	99.150 $\pm$ 0.072
VGG-16 CIFAR10 BN	99.988 $\pm$ 0.024	87.080 $\pm$ 0.315	85.99 $\pm$ 0.320
VGG-16 CIFAR10 NBN	99.982 $\pm$ 0.021	81.030 $\pm$ 1.008	79.435 $\pm$ 1.059

3.4.1.2 Sparsity

For this case study, we investigate two types of sparsity: channel and layer sparsity. As specified in Section 2.3.2, we continue to use a zero value as our measure of sparsity. Although activation values close to zero could have minimal impact on the networks’ decision-making process and could therefore also be considered as values for which the network effectively does not activate [2, p. 251], such small values are not considered sparse elements. To calculate sparsity, the inactive elements are first identified – those with activation values of zero. The tensor is then reshaped to focus on the specified axis. For the channel sparsity, the resulting 2D matrix is $\mathbf{X}_B^l \in \mathbb{R}^{C \times NHW}$ and the sparsity calculated as:

$$s_c = \frac{1}{N \cdot H \cdot W} \sum_{j=1}^{NH} \mathbb{I}(\mathbf{x}_{c,j}^l = 0) \quad (3.2)$$

where $\mathbf{x}_{c,j}^l$ is the element of the matrix at position (c, j) where c is the channel, and $\mathbb{I}(\cdot)$ is the indicator function, which equals 1 if the condition inside is true and 0 otherwise. To calculate the layer sparsity we use the vector $\mathbf{x}_B^l \in \mathbb{R}^{1 \times NCHW}$ as follows:

$$s_l = \frac{1}{N \cdot C \cdot H \cdot W} \sum_{j=1}^{NCHW} \mathbb{I}(\mathbf{x}_j^l = 0) \quad (3.3)$$

resulting in a single sparsity value per layer (per model and seed).

3.4.1.3 Clustered representations

We analyse the cluster characteristics of the representations to establish which differences, if any, are present in the models’ activation space. The clusters are analysed using class-based and class-agnostic clustering methods as described in Section 3.3.6. For class-based clustering,

the same method as described in Section 3.3.6 is used, where the class of the sample is assumed to be the cluster label. We then make use of the DBI to measure the purity of the clusters. For the class-agnostic approach, again the k-means clustering algorithm is used to group similar representations. To find the optimal number of clusters, a sweep is performed for 2 to 15 clusters, selecting the optimal number of clusters as those that produced the lowest DBI score. This is repeated for each layer of the model.

We make use of the same representations ($\mathbf{x}^l$) as used in the previous sections. Due to the high dimensionality of the representations and the computational complexity associated with the high dimensionality, the representations, dimensionality is reduced using Equation 3.1, to $\mathbf{X}_{\text{flat}}^l \in \mathbb{R}^{N \times C}$. After which each sample representation is normalised by dividing it by its Euclidean norm to produce $\mathbf{x}^l$ to be $\in [0, 1]$.

3.4.2 Sparsity analysis and results

In this section, we analyse and compare the sparsity of networks trained with and without BatchNorm. We consider the sparsity of the representations both per channel (Section 3.4.2.1) and per layer (Section 3.4.2.2).

3.4.2.1 Channel sparsity

We begin by analysing the sparsity of individual channels within the network for selected layers. The analysis uses the representations of the hidden layers with dimensions $\mathbf{X}_{\text{flat}}^l \in \mathbb{R}^{C \times NHW}$, as described in Section 3.4.1.2. The sparsity for each layer is calculated according to Equation 3.2, using 5 000 randomly selected training samples. A subset of layers spread through the networks is selected for evaluation, the same layers for each model type.

Figures 3.9 and 3.10 illustrate the average channel sparsity for the four models for BatchNorm and non-BatchNorm configurations, across the MNIST and CIFAR10 datasets, respectively. The standard deviation of metrics across different seeds is indicated by shading. We choose layers 6, 10 and 13 to probe the networks in the deeper layers at consistent intervals, and measure the DBI of the clusters at each of these layer of the network.

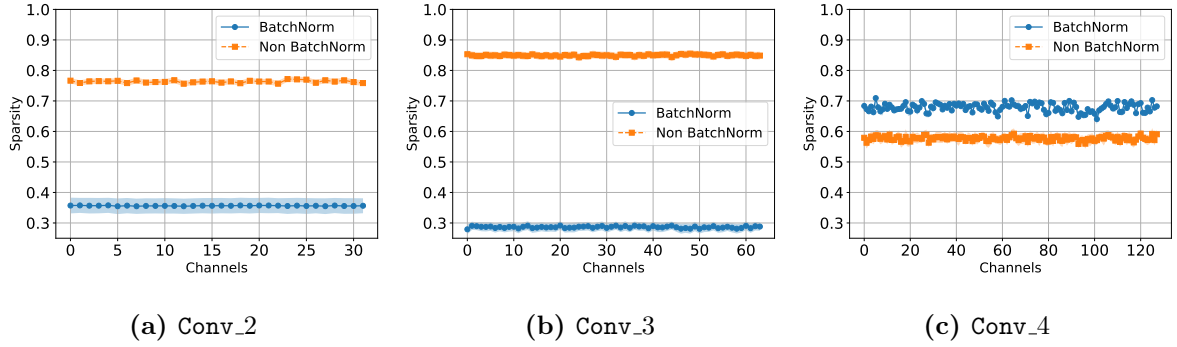


Figure 3.9: Average channel sparsity for BatchNorm (blue) and Non-BatchNorm (orange) models for selected convolutional layers of the standard CNN models trained on MNIST. Error bars show the standard deviation across 4 random initialisation seeds.

For the standard CNN models trained on MNIST, as shown in Figure 3.9, we observe that the models trained with BatchNorm generally exhibit lower channel sparsity compared to those trained without BatchNorm. This observation is consistent across all channels and layers of the model, with the exception of the final layer, where channels in the BatchNorm models are more sparse than those in the Non-BatchNorm models.

We note that channel sparsity remains consistent across different initialisation seeds and is also relatively uniform within each layer. The one exception is the last layer, where there is a slightly larger degree of variation in channel sparsity, especially for BatchNorm models.

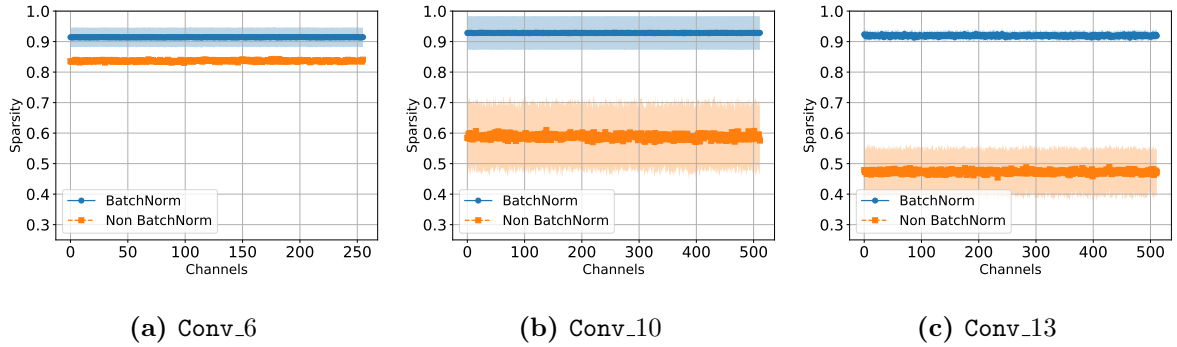


Figure 3.10: Average channel sparsity for BatchNorm (blue) and Non-BatchNorm (orange) models in selected VGG-16 convolutional layers trained on CIFAR10, with error bars showing the standard deviation across 4 random initialisation seeds.

However, when we turn our attention to Figure 3.10, which shows the sparsity per channel for the VGG-16 models trained on CIFAR10, the opposite behaviour is observed. The VGG-16 models trained with BatchNorm tend to be more sparse than those trained without BatchNorm. We

also note greater variation among different initialisation seeds, especially for the Non-BatchNorm models. That said, it is noted that each channel shows similar sparsity within a layer.

When comparing Figures 3.9 and 3.10, it is evident that the channels in the BatchNorm-trained CIFAR10 models are significantly more sparse than their MNIST counterparts. This could potentially be an artefact of the different datasets, as MNIST consists of many zero-valued inputs in itself whereas CIFAR10 consists of more uniformly distributed values larger than zero. It is therefore strange that the CIFAR10 BatchNorm models are *more* sparse than those of MNIST.

3.4.2.2 Layer sparsity

In this section, instead of analysing a single channel individually, we analyse the sparsity of each layer within the networks. This analysis uses the representations of the hidden layers with dimensions $\mathbf{X}_{\text{flat}}^l \in \mathbb{R}^{N \times CHW}$, as described in Section 3.4.1.2. Once again only 5 000 randomly selected training samples are used. For each layer, the sparsity is calculated according to Equation 3.2.

Figure 3.11 illustrates the average layer sparsity for both the MNIST (left) and CIFAR10 (right) architectures with and without BatchNorm. We again include error bars indicating the standard deviation across the 4 random initialisation seeds.

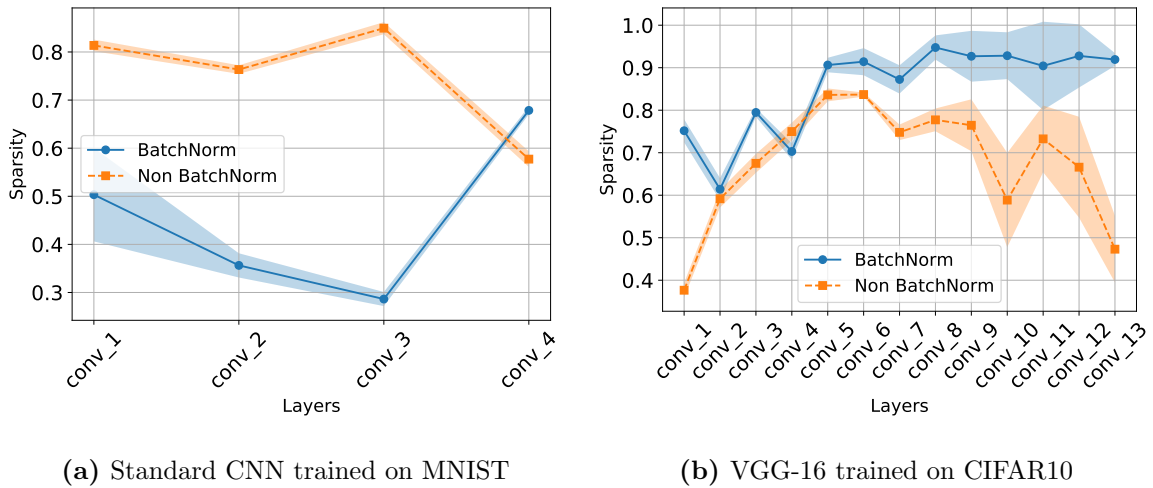


Figure 3.11: Average layer sparsity for models trained with (blue) and without (orange) BatchNorm. Error bars indicate the standard deviation across 4 random initialisation seeds.

Figure 3.11 reveals a pronounced difference in layer sparsity between the standard CNNs and VGG-16 models. For the standard CNNs, the representations of the BatchNorm models are generally less sparse than those of the models trained without BatchNorm, with the exception of the final layer. Conversely, for the VGG-16 models, the BatchNorm models exhibit greater overall sparsity compared to the Non-BatchNorm models. The sparsity values are consistent across different seeds for both architectures, except for the last layers of the VGG-16 models where the variation is higher between seeds. Notably, the sparsity in the VGG-16 Non-BatchNorm models decreases significantly in the final layers, while the sparsity in the BatchNorm models remains consistent and relatively high.

It has been speculated that sparser representations are more desirable, as they suggest that the network is selectively activating in response to specific features. However, it is evident that BatchNorm neither consistently induces nor prevents sparsity in the representations. Yet, in both cases, the models with BatchNorm generalise better. This points to the notion that the presence or absence of sparse representations does not appear to be a primary factor that determines the generalisation ability of a network. Instead, it is likely that other effects of BatchNorm play a more significant role in the networks' generalisation performance.

3.4.3 Clustering analysis and results

In this section, we analyse the effect of BatchNorm on the purity of representational clusters formed by the network. We consider this from both a class-specific (Section 3.4.3.1) and a class-agnostic (Section 3.4.3.2) perspective.

3.4.3.1 Class-based analysis

We first analyse the purity of the clusters formed in each layer of the network, under the assumption that the class of each sample corresponds to the cluster to which it belongs. 15 000 randomly selected training samples are used and the representations are dimensionally reduced as described in Section 3.4.1.3.

Figure 3.12 presents the DBI scores for each layer of the networks trained with and without BatchNorm, for both the MNIST and CIFAR10 architectures, respectively. The average DBI

scores across the 4 different initialisation seeds, along with the corresponding standard deviation, are displayed. For the VGG-16 network, we only show from convolutional layer six onward as the earlier layers are not informative and show unstable behaviour among different seeds.

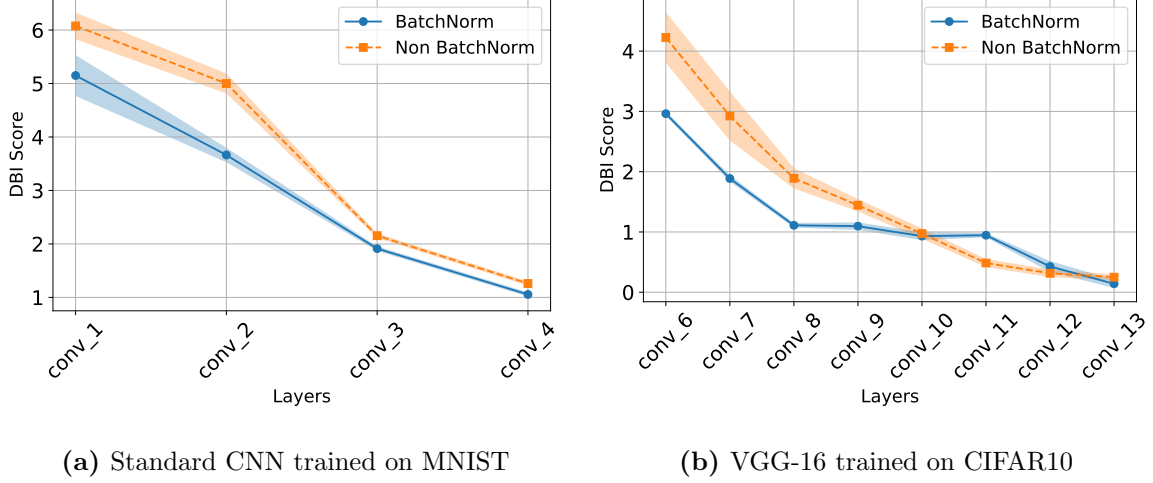


Figure 3.12: Average class-based DBI score per layer for BatchNorm (blue) and Non-BatchNorm (orange) models, with error bars showing standard deviation across 4 random initialisation seeds. Lower scores indicate higher purity.

For both network architectures, the models trained with BatchNorm generally exhibit higher cluster purity compared to those trained without BatchNorm. An exception is observed in **Conv_11** and **Conv_12** of the VGG-16 model; however, the overall cluster purity is better in the BatchNorm models. Furthermore, we also observe that the purity values are relatively consistent across the different seeds, with minimal variation.

Since only the 10 dataset classes serve as the cluster labels, and each sample can only belong to the cluster corresponding to its class label, both models start off with very low purity when traversing the layers. However, in deeper layers, the clusters become increasingly pure, eventually nearing complete purity, as observed in **Conv_13** of the VGG-16 models.

We suspect that the common features shared among different classes could lead to the formation of either fewer or more clusters than there are classes in the earlier layers. Overall, observations are consistent with the well-known belief that earlier layers learn more general patterns in CNNs [27]. In deeper layers, we suspect that more class-specific features, with fewer shared features between classes, likely contribute to the higher clustering purity observed.

3.4.3.2 Class-agnostic analysis

We now turn our attention to the class-agnostic clustering perspective. As described in Section 3.4.1.3, clusters are assigned using the k-means clustering algorithm and then measuring the resulting purity. As in the previous experiment, a total of 15 000 randomly selected training samples are used for these calculations.

Optimal number of clusters

The optimal number of clusters for each layer is shown in Figure 3.13 for both the MNIST (left) and CIFAR10 (right) architectures. The error bars again report the standard deviation across the 4 random initialisation seeds. We show only from the convolutional `Conv_6` onwards for the VGG-16 models as the initial layers are not informative and unstable between seeds.

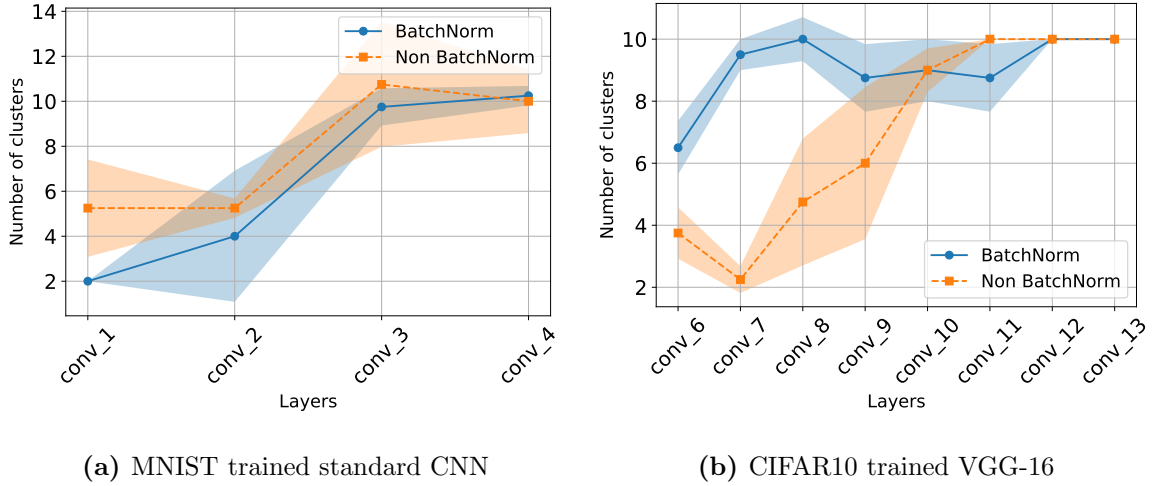


Figure 3.13: Average optimal number of class-agnostic clusters per layer for BatchNorm (blue) and Non-BatchNorm (orange) models, with error bars showing the standard deviation across 4 random initialisation seeds.

In Figure 3.13 we observe that for both architectures, the optimal number of clusters initially starts small, but then increases. In the final layers, the optimal number converges to 10 clusters, corresponding to the number of classes in the dataset. For both the standard CNNs and the VGG-16 models, the increase towards the optimal number of clusters occurs closer to the final layers of the model. This is true for both the models trained with and without BatchNorm. For VGG-16 models, the optimal cluster patterns differ between model types.

In BatchNorm models, the number of optimal clusters increases in the early layers, stabilising until the last two layers, where it matches the number of classes. For Non-BatchNorm models, the optimal number of clusters is initially reached but significantly decreases in the middle layers, then increases again in the later layers, ultimately matching the number of classes in the final layers.

These observations support the hypothesis presented in Section 3.4.3.1 that optimal class-wise clusters are achieved only in the later layers. It suggests that while different class features contribute to cluster formation, the clusters do not become class-specific until the final layers of the network. These plots also suggest that the improved generalisation ability of BatchNorm models is likely due to the greater consistency of clusters formed by the representations throughout the network, compared to those in models trained without BatchNorm.

Cluster purity

We now consider the DBI score of each optimal cluster found using k-means. This is shown for the MNIST and CIFAR10 models in Figure 3.14. Again, only Conv_6 onwards for the VGG-16 models is shown, as the initial layers are not informative and unstable between seeds. In

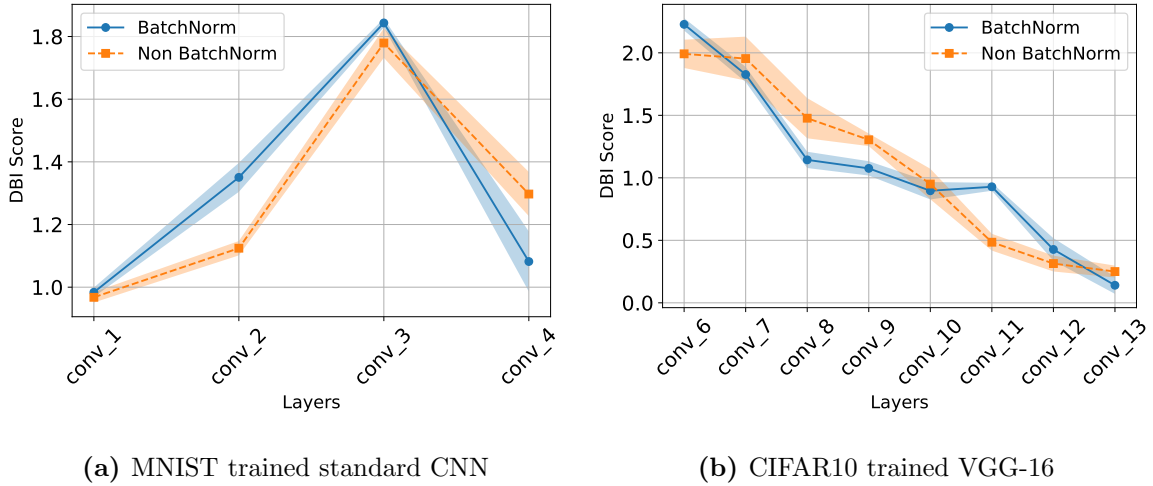


Figure 3.14: Average class-agnostic DBI score for k-means per layer for BatchNorm (blue) and Non-BatchNorm (orange) models, with error bars indicating standard deviation across 4 random initialisation seeds. Lower scores indicate higher purity.

Figure 3.14 we observe that the standard CNNs, for both model types, follow the same pattern where the DBI score increases significantly until Conv_3 and then decreases. The BatchNorm models also have a slightly higher score than the Non-BatchNorm except for the last layer, where

the BatchNorm has a lower score. Comparing these results with what was seen in Figure 3.12 with Figure 3.14 there is a difference: the scores in Figure 3.12 only decrease, while the DBI scores in Figure 3.14 increase before decreasing in the last layer. One thing to note is that the DBI scores in Figure 3.12 are significantly higher (poorer) than those in Figure 3.14.

For the VGG-16 network, the overall DBI score of the BatchNorm models is better compared to the non-BatchNorm models. Variation between seeds is more stable for the BatchNorm models than for the non-BatchNorm models. This suggests that the clusters are more stable for the BatchNorm models over seeds. Comparing Figure 3.14 with Figure 3.12, the same pattern emerges for the VGG-16 models, where the BatchNorm models have purer clusters except for `Conv_11` and `Conv_12`. This suggests better cluster forming in general, and thus better generalisation in the BatchNorm models, especially in the last layer.

3.4.4 Discussion

Let us summarise the main observations. In the context of convolutional neural networks and the experiments conducted in Section 3.4:

1. BatchNorm does not consistently induce nor prevent sparsity in hidden representations.
2. Sparse representations do not correlate with better generalisation.
3. BatchNorm tends to produce purer representational clusters than models trained without it, especially in the case of class-based representations. For class-agnostic representation, the results are less clear, except at the final layer, where BatchNorm clusters are again purer.
4. BatchNorm forms class-agnostic clusters earlier in the network, and these early clusters are more consistent with the actual number of classes.

It is challenging to determine the exact cause of the sparsity in the representations studied because we observed significant differences between the models trained on MNIST and those trained on CIFAR10. However, a clear distinction exists between models with BatchNorm and those without BatchNorm. For standard CNNs, applying BatchNorm results in much less sparse

representations, which is consistent with the findings of Pretorius et al. [54]. In contrast, VGG-16 models exhibit the opposite behaviour, where BatchNorm models produce significantly more sparse representations than their non-BatchNorm counterparts.

The difference in sparsity might be influenced by the dataset on which the models are trained or by the architectural differences between the models. Specifically, VGG-16 models with BatchNorm seem to activate for more specific features, leading to increased sparsity, particularly around `Conv_5`. On the other hand, non-BatchNorm models in VGG-16 also show increased sparsity until `Conv_5`, but revert to activating more general features in deeper layers, causing a reduction in sparsity. Also note the variability in the results across seeds for the deeper layers of the standard CNN (Figure 3.9c) – this could be indicative of other underlying processes at play.

Overall, we found that the BatchNorm models tend to produce purer clusters compared to non-BatchNorm models. In the VGG-16 models, both clustering methods yield similar results, and BatchNorm has purer clusters overall, with the exception of `Conv_11` and `Conv_12`. For the layers where non-BatchNorm has purer clusters in class-agnostic clustering, it results in more clusters and a wider range of cluster numbers across seeds compared to the BatchNorm models.

3.5 Conclusion

Studying the characteristics of DNNs internal representations can provide insight into the underlying mechanisms that contribute to good generalisation. We used a limited set of representative models with different architectures, trained on different datasets, and with different generalisation gaps to investigate the hidden representations obtained after the ReLU activation function. The same training data is used to analyse the models with different generalisation gaps to ensure that both models have ‘memorised’ the data and to be able to compare them.

From the results of both case studies in Section 3.3 and Section 3.4 we found the following with regard to our limited model selection:

- For the PGDL models, the better generalising models tend to have constant activation values, whereas their counterparts’ activation values tend to increase in scale with each

subsequent layer.

- When visualising representations that are dimensionally reduced, better generalising models have purer class-based clusters compared to those of the poor generalising models.
- Better generalising models tend to have sparser representations. Although this observation does not hold in all cases of better generalising BatchNorm models – the opposite was only observed in the standard MNIST models – it holds for the majority of cases we observed.
- Models that generalise better tend to produce purer class-based clusters.
- Models that generalise better tend to have closer numbers of optimal class-agnostic clusters to the number of classes than those of their counterparts.
- While poor generalising models produce purer class-agnostic clusters in the convolutional layers, better generalising models consistently produce purer clusters in the fully connected layers.
- It remains unclear whether more distinct class-agnostic clusters are superior.

To summarise: It seems that models that generalise better activate for more distinct features including specific object features, especially in the fully connected layers, resulting in channels that do not activate for every class but rather activate for specific classes. This suggests that channels ‘learn’ to identify specific object features associated with distinct classes, unlike the poorer generalising models that activate for more general object features common to all classes. However, there is no indication of class-specific activation in the convolutional layers.

Overall, better generalising models tended to produce purer representational clusters, both in the case of class-based and class-agnostic analysis. Although this was not observed in the convolutional layers for class-agnostic clustering, it was always observed for the fully connected layers. In addition, more class-specific clusters formed earlier in the better generalising networks, which were more consistent with the actual number of classes.

In this chapter, our objective was to provide initial results and motivate the importance and potential of studying the characteristics of DNN representations to shed light on the underlying mechanisms controlling the generalisation ability of these models. These initial probing experiments indicated that there might be underlying characteristics in the representations of models

with different generalisation gaps. However, we could not investigate these characteristics with the clustering methods we used, as they only cluster based on one axis, while the characteristics appear to be connected with the relationships between the samples and channels. Our intention is to investigate this in the next chapter.

In this chapter, we identify two problems that we will investigate further in the following chapters. The first being biclusters in the sense of sample/feature clusters where some samples with varying activation values for specific channels are clustered together (Chapter 4). The second is a representation summary method, which reduces the dimensionality of the representations while keeping the key features (Chapter 5).

Chapter 4

Biclustering of representations

“Curiously enough, the only thing that went through the mind of the bowl of petunias as it fell was Oh no, not again. Many people have speculated that if we knew exactly why the bowl of petunias had thought that we would know a lot more about the nature of the Universe than we do now.”

- Douglas Adams, The Ultimate Hitchhiker’s Guide to the Galaxy Page 91 [1]

4.1 Introduction

Conventional clustering only allows for clustering along a single dimension, either samples or features. Up to this point, we have clustered samples together based on their representations. The techniques used only account for clustering along a single dimension, which prevents the capture of the local structure created by the relationships between features and samples, which is essentially our main objective. Biclustering offers a way to explore these relationships by sub-clustering samples and features that are correlated in some way. Through biclustering, we can uncover complex patterns of local structures within representations that we cannot explore with traditional clustering methods.

The aim of this chapter is to apply biclustering to some of the models we previously examined to

determine if we can identify localised structures in their representations and to observe whether there are localised differences between models with different generalisation capabilities. Using this info, we construct metrics based on these local structures, and explore how predictive they are of generalisation in the context of the same PGDL task analysed in the previous chapter.

In this chapter, we conduct two primary experiments:

1. Investigating whether purer biclusters are associated with better generalising models.
2. Investigating whether bicluster characteristics are predictive of generalisation.

We first describe the experimental setup of both experiments in Section 4.2. We then conduct each of the experiments separately (Sections 4.3 and 4.4) before discussing the results of the experiments in Section 4.5.

4.2 Experimental setup

We describe the models, methods, and quality measures applied throughout this chapter. As in the previous chapter, we again extract the representations as described in Section 2.3.1, where the representations are recorded after the activation function. For computational efficiency, we do not use the full representation of a sample but rather extract the average channel representations in the same way as done in Section 3.3.5. Moreover, to facilitate comparisons of models with different generalisation abilities, we normalise the representations in the same way as done in Section 3.3.6. As in the previous chapter, we make use of the samples as used in Section 3.2, unless otherwise specified for a particular experiment.

4.2.1 Models

We use a different set of models for each of our experiments. For the biclustering of the representations (Section 4.3), we use certain models from the PGDL Task 1, specifically GS, BS, GL, and BL as described in Section 3.3.2. Once again, we concentrate primarily on the GS and BS models, incorporating outcomes from the BL and GL models exclusively when they present distinctive or noteworthy insights that enhance the findings derived from GS and BS.

When predicting generalisation (Section 4.4.3), we use all of the PGDL Task 1 models (Section 3.3.2), to ensure a comprehensive representation of different model architectures and generalisation scores. When considering all of the Task 1 models, there are some models with a slight variation in architecture compared to GS, BS, GL, and BL. This is the inclusion of an additional fully connected layer (`Lin_2`) for some of the smaller architecture models. Both the smaller and larger architectures also have different variations in regard to the number of channels each convolutional layer of the models has. When using the whole of Task 1 we categorised the models based on architecture, specifically on the number of convolutional layers and not the number of fully connected layers, separating the smaller and the larger architectures.

4.2.2 Biclustering methods

As introduced in Section 2.6.2, biclustering is an unsupervised learning technique that simultaneously clusters both rows and columns of data. It differs from other clustering techniques due to its ability to identify submatrices within the data that are similar across rows or columns. This proves valuable when dealing with high-dimensional data where patterns are localised to specific regions. Unlike global clustering techniques, biclustering is able to capture coherent patterns in complex datasets, for instance, in situations where condition-specific relationships between rows and columns are present. Various biclustering methods have been developed to capture patterns, such as scaling, shifting, and combinations of both, found in datasets. A discussion of the different biclustering techniques is provided in greater depth in Section 2.6.2.

Given the exploratory nature of this study (we do not yet know which bicluster details would be informative) we require a method that is well established, reliable, and straightforward to implement. The method should also be able to identify non-overlapping biclusters among the samples and channels. Taking into account these requirements, we selected the Spectral Biclustering algorithm available in the Sklearn toolbox [36].

When applying this biclustering algorithm, it is important to take into account its specific limitations. Due to the checkerboard structure of the algorithm, each feature is moved to the cluster with which it is associated the most. Consequently, the feature’s placement is restricted, as it can only move within the structure of its corresponding row/column. Since the clusters are composed of adjacent features within this structure, this ‘adjacency’ is identical across

all rows and all columns. When maintaining a rectangular structure while grouping similar features, not all features are necessarily aligned with their adjacent counterparts if the bicluster extends beyond the confines of the structure. This arrangement establishes order within the biclusters; however, it may also affect their purity and composition. If the algorithm optimises for the highest cluster purity, it could lead to the optimal number of biclusters, equating to the dimensionality of the representations since each feature is in its own bicluster.

4.2.3 Biclustering quality metrics

The purity of biclusters cannot be assessed using the same criteria as those used for traditional clustering methods, since biclusters are simultaneously formed based on both the rows and columns. Due to no ground truth existing for biclustered hidden representations, it becomes difficult to evaluate whether similar features are clustered together. Fortunately, the majority of research on biclustering comes from the medical field, where it is mostly applied to gene research, which encounters the same issue, as there is no definite reference to determine gene similarity. This led to a number of quality metrics (QMs) being developed to produce cluster quality results without the need for ground truth labels [75], [79]–[82]. Unfortunately, this implies that these techniques may not be applicable to our data because they were originally designed specifically for gene research [74], [75].

In the remainder of this section, we first discuss the different types of biclusters we expect to encounter in Section 4.2.3.1, and provide an example of each. We then discuss the QMs we select in Section 4.2.3.2, before providing a demonstration of each QM method on the example biclusters in Section 4.2.3.3.

4.2.3.1 Types of biclusters

It is important to first consider the various types of biclusters that we might encounter in the representations of neural networks before choosing QMs. We identified five different types of biclusters in Section 2.6.1 as follows: ‘perfect’, ‘shifted’, ‘scaled’, ‘row’, and ‘column’ biclusters, for which we show examples in Table 4.1. A perfect cluster (Table 4.1a) is composed of elements that closely resemble each other, in our example, exactly the same. The row (Table 4.1d) and column clusters (Table 4.1e) are transposed versions of each other, possessing similar elements

along an axis that allows them to be clustered together. A shifted cluster (Table 4.1b) is composed of elements that are aligned along an axis, each displaced from the next by a consistent amount along that axis. A scaled cluster (Table 4.1c) is similar to a shifted cluster; however, the elements are multiplied by a constant factor instead of adding a constant term. Even though we list the row and column clusters as unique biclusters, any shifted or scaled cluster can be evaluated in either of these forms. We present the non-perfect shifted variant in Tables 4.1d and 4.1e with an added scaled row/column to demonstrate our metrics.

Table 4.1: Different types of bicluster examples that we may encounter.

1.0	1.0	1.0	1.0	1.0	2.0	3.0	4.0	1.0	2.0	3.0	4.0
1.0	1.0	1.0	1.0	3.0	4.0	5.0	6.0	2.0	4.0	6.0	8.0
1.0	1.0	1.0	1.0	2.0	3.0	4.0	5.0	3.0	6.0	9.0	12.0
1.0	1.0	1.0	1.0	7.0	8.0	9.0	10.0	4.0	8.0	12.0	16.0
(a) Perfect bicluster				(b) Perfect shifted row bicluster				(c) Perfect scaled row bicluster			
1.0	2.0	3.0	4.0					1.0	3.5	0.5	4.0
3.5	4.5	5.5	6.5					2.0	4.5	1.0	5.0
0.5	1.0	1.5	2.0					3.0	5.5	1.5	6.0
4.0	5.0	6.0	7.0					4.0	6.5	2.0	7.0
(d) Non-perfect shifted row bicluster								(e) Non-Perfect shifted column bicluster			

4.2.3.2 Choosing bicluster quality measures

The QMs that we intend to use are mean squared residue (MSR), scaling mean squared residue (SMSR), virtual error (VE), and virtual error transposed (VEt). A more detailed description of their operation is available in Section 2.6.3: here, we offer a concise overview to contrast their respective strengths and limitations.

VE [81] measures the general tendency within a row bicluster and can recognise both perfect-shifting and perfect-scaling biclusters, but not both at the same time. Although VEt [82] is calculated similarly to VE, it considers the transposed bicluster while recognising both perfect-shifting and perfect-scaling biclusters independently or combinations of the two which differs from VE which cannot detect combinations of both. MSR [75] evaluates the coherence of the rows and columns of the bicluster; however, it struggles with scaling biclusters and can only recognise shifting clusters. Lastly, SMSR [80] is based on the same working principals as MSR except that

it has the added benefit of recognising perfect-scaling biclusters although not perfect-shifting biclusters. Each of these metrics yields a score of zero when assessing a perfect bicluster.

To select the specific QMs that we will use, we need to rely on our knowledge from previous experiments on representations. In Chapter 3 we observe that hidden representation can have significantly scaled values. Even though normalisation lowers the representation scale across each sample, it has a global effect and not a local effect, which diminish the impact of scale differences resulting in more fair comparisons. This brings us to a method that is scale invariant. Additionally, we require a technique that consistently identifies row biclusters as we are mainly interested in samples with similar representations. However, it might also be interesting to investigate the alternative, whether there are channels which react in a similar way to samples. Finally, we need a method capable of detecting shifted biclusters, since the activation values of similar samples/channels might be slightly different, altered by a constant bias.

With these requirements in mind, we choose VEt and SMSR. Both are scale-invariant, while VEt is also selected for its ability to recognise shifted biclusters, and SMSR for its consistency in recognising both row and column biclusters.

4.2.3.3 QM toolset demonstration

To demonstrate the QM methods and showcase their characteristics, we apply them to the artificial toy dataset that we describe in Table 4.1. For each of the example biclusters, we use a perfect version in the cases of perfect, shifted, and scaled biclusters, whereas the row and column biclusters are non-perfect shifted versions of their respective type. We purposely selected these clusters as this is the bicluster versions that we expect to find in our representations. For each of the QM methods, we calculate the purity score of each test case and record it in Table 4.2. If the QM is able to distinguish between a perfect bicluster and a non-perfect bicluster, it will produce a score larger than zero. However, if the QM recognises a scaled or shifted pattern which is equivalent to a perfect bicluster, a score of zero will be produced.

By analysing the results in Table 4.2 we can observe if the QMs perform as expected when applied to our synthetic data. Naturally, we cannot compare the QMs directly with one another, since their scores are not on the same scale. Since each of them produce a score of zero when a perfect bicluster is measured, we can compare them based on if the same bicluster is detected as a

Table 4.2: QM purity scores for each of the example biclusters as showed in Table: 4.1.

QM	Perfect (4.1a)	Shifted (4.1b)	Scaled (4.1c)	Row (4.1d)	Column (4.1e)
VE	0.0000	0.0000	0.0000	0.0558	0.000
VEt	0.0000	0.0000	0.0000	0.0000	0.0558
MSR	0.0000	0.0000	1.5625	0.0586	0.0586
SMSR	0.0000	0.0098	0.0000	0.0067	0.0067

perfect bicluster or not.

Starting with the perfect bicluster (4.1a) all the QM methods produce a score of 0, recognising it as such. Next, all the methods recognise 4.1b as a perfect shifting bicluster, except SMSR which has a score greater than 0, confirming that it struggles with shifted biclusters. Moving on, all QMs indicate 4.1c as a perfect scaled cluster except MSR, confirming that it is unable to recognise scaled biclusters. Although, VE performs better on the column bicluster, VEt performs better on the row bicluster, when comparing them with each other as each produces a zero score for the respective bicluster. Lastly, for both the row and column biclusters, MSR produces constant scores, as does SMSR. Although 4.1d and 4.1e are not perfect row and column biclusters since none of the methods produced a zero score, we can still use them to expose QMs that struggle with these types of bicluster, as is the case with VE and VEt, which did not perform consistently.

4.3 Biclustering of representations

In this section, we investigate the possibility of underlying patterns in the representations of models with different generalisation scores. We first describe the experimental setup (Section 4.3.1) whereafter we observe the QM scores from different bicluster combinations (Section 4.3.2) and lastly we analyse the resulting best-performing bicluster combinations (Section 4.3.3). Note that we refer to the specific number of row and column clusters employed by the bicluster algorithm as a ‘bicluster combination’.

4.3.1 Method

We make use of a spectral biclustering algorithm (Section 2.6.1) which allows us to find and analyse these sample/feature clusters. We again use the representations from BS and GS since their small architecture is perfect for probing and they have a large generalisation gap making it easier to find differences in the local structure. We then measure the purity of the resulting biclusters using the QMs identified in Section 4.2.3. Since the QMs provide a score for each bicluster, we calculate the average score for the bicluster combination, which we report. For example: a representation matrix biclustered using a bicluster combination of 2 sample by 2 channel biclusters results in 4 bicluster QM scores. We report on the average of those 4 scores.

The use of the Spectral Biclustering algorithm from the Scikit Learn toolbox [36] presents a number of hyperparameters that need to be defined. The most important hyperparameters are the number of row and column clusters that are present in the data (bicluster combination). Since we do not know the exact number of biclusters present in the data, except for the number of classes in the dataset, we perform a grid search that consists of 2 to 24 biclusters for both rows and columns.

During initial investigations, we used different combinations of the singular value decomposition (SVD) vectors: [6, 10, 15, 25, 50] and the best singular vectors: [3, 5, 10, 15, 25]. We observed an increase in the overall purity of the bicluster combinations when using more vectors. Due to computational complexity, we did not try any combinations greater than this. We select the number of vectors used in the SVD calculations as 50 and the number of best singular vectors as 25 as these values provided the best results during the initial investigations. Lastly, we select a fixed random seed of 1 for this experiment, which is used for the initialisation of the k-means algorithm, which makes the randomness deterministic and the results replicable. In addition, all other hyperparameters are left at their default values, which include the singular vector normalisation method, as well as the initialisation hyperparameters used for the k-means++ algorithm.

4.3.2 Result: Bicluster purity

The results of the grid search for the optimal bicluster combinations for both convolutional layers are shown in Figure 4.1 and the fully connected layer in Figure 4.2. For example: consider the top left value in the top left VEt plot of Figure 4.1a. This indicates that for two sample clusters and two channel clusters (the two-by-two bicluster combination), the average VEt over the four clusters is ± 0.4 . The plots also show the mean score obtained during the grid search process, as well as the lowest score achieved. We also summarise the average and best scores of each layer from the grid search in Table A.1.

Table 4.3: Summary of the average and best VEt and SMSR from the bicluster combination grid search.

Layer	VEt average	VEt best	SMSR average	SMSR best
BS Conv_1	0.3831	0.2336	0.0341	0.0201
GS Conv_1	0.3736	0.2908	0.0426	0.0211
BS Conv_2	0.4497	0.3248	0.0492	0.0287
GS Conv_2	0.6476	0.5428	0.2408	0.1458
BS Lin_1	0.6635	0.5307	0.4453	0.2621
GS Lin_1	0.6910	0.6052	0.8177	0.5294

We first consider the differences between the scale of the QM values compared between the two models in Figures 4.1 and 4.2. In the case of **Conv_1**, there is no apparent scale difference between the models. However, we do notice a significant scale difference in the QM scores between GS and BS for both **Conv_2** and **Lin_1**. This is also apparent when considering the scores in Table A.1. These scores indicate that GS has much less pure biclusters compared to BS, especially when looking at the SMSR scores of **Lin_1**. Similar results are observed for GL and BL (Appendix A.2.1), however GL has more pure biclusters in **Conv_1** and **Lin_2** than BL.

Next, we observe the patterns formed by the scores in Figures 4.1 and 4.2. The VEt scores exhibit a consistent trend across layers: they are highest (worst purity) at the top, improving progressively to their lowest (best purity) at the bottom left, corresponding to fewer channel clusters and more sample clusters. Although there is some variation, this pattern is evident in all of the layers. Similarly, SMSR scores are highest in the top left, where biclusters are fewest, and decrease along both the sample and channel cluster axes, reaching their lowest in the bottom right, where the number of biclusters is greatest.

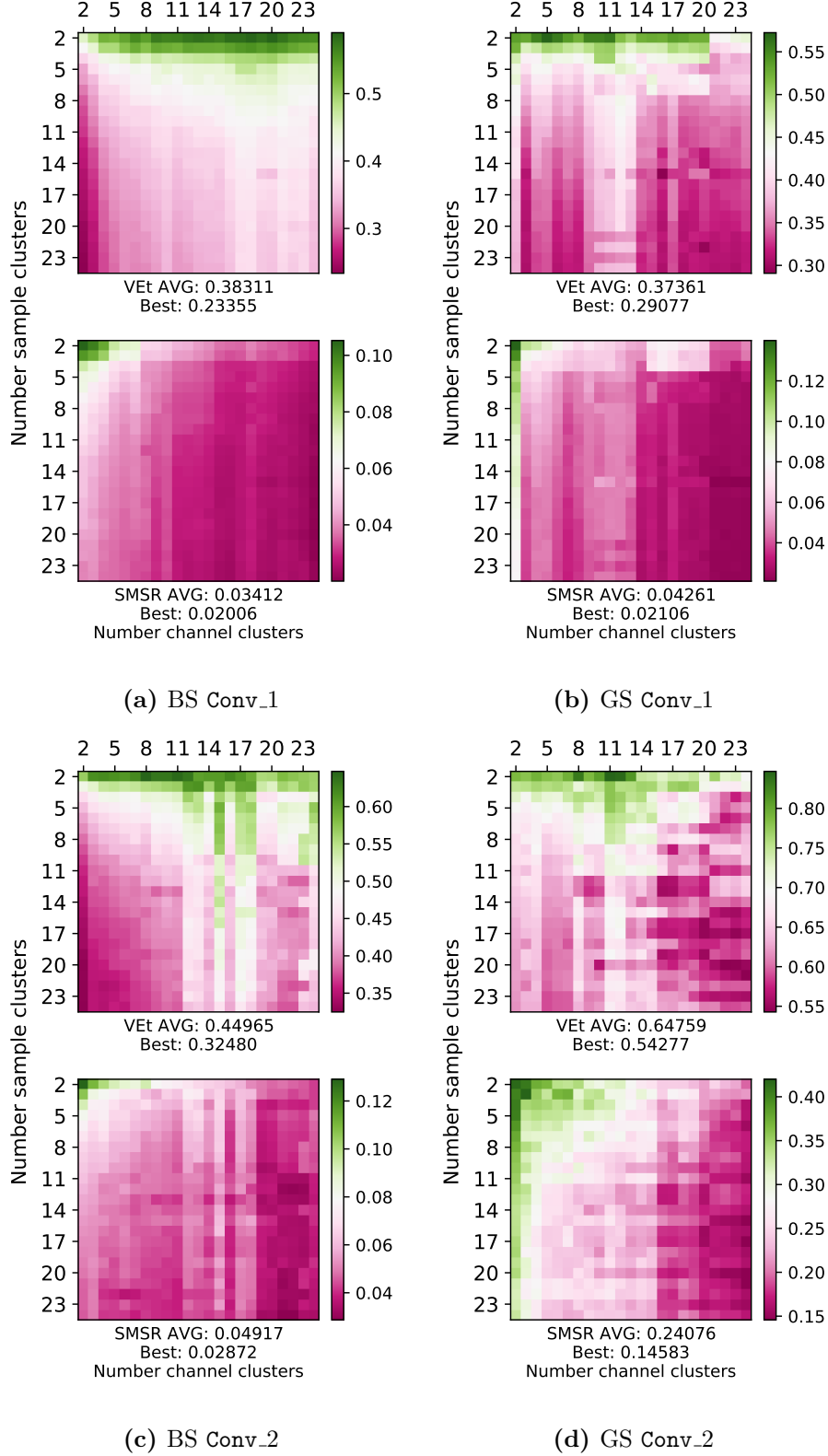


Figure 4.1: Average VEt and SMSR for all tested bicluster combinations. Both convolutional layers for both BS and GS are shown. A lower score indicates a purer bicluster average. (Take note of the scale difference between models.)

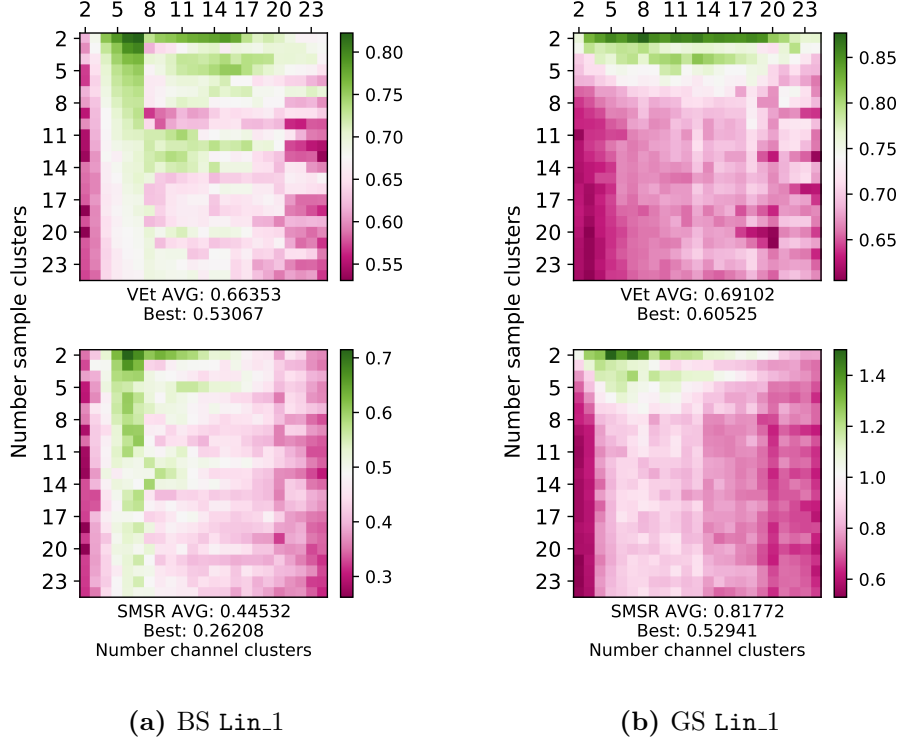


Figure 4.2: Average VET and SMSR for all tested bicluster combinations. The fully connected layer for both BS and GS are shown. A lower score indicates a purer bicluster average. (Take note of the scale difference between models.)

4.3.3 Results: Visualising best performing combinations

We now direct our attention to the bicluster combinations that produced the lowest scores (highest purity). These are selected for each layer and for each model. For example, in Figure 4.1b we identify the lowest score at 15 samples and 16 channel clusters. Each plot presents the representations in matrix format where the channels are on the x-axis and the samples are on the y-axis. The samples and channels are sorted according to the cluster ID assigned to each sample/channel by the clustering algorithm and each cluster is indicated with the x/y ticks on the plot.

For example, in Figure 4.3a we observe a sample cluster from index 0 to 257 that spans across all channels, alternatively we observe a channel cluster that spans from 0 to 50 on the x-axis across all samples. To make the figures easier to interpret, we clamp the maximum representation value at 95% and the minimum value at 5% of their respective ranges, since outliers can significantly influence the colour scale of the plots, making them difficult to interpret.

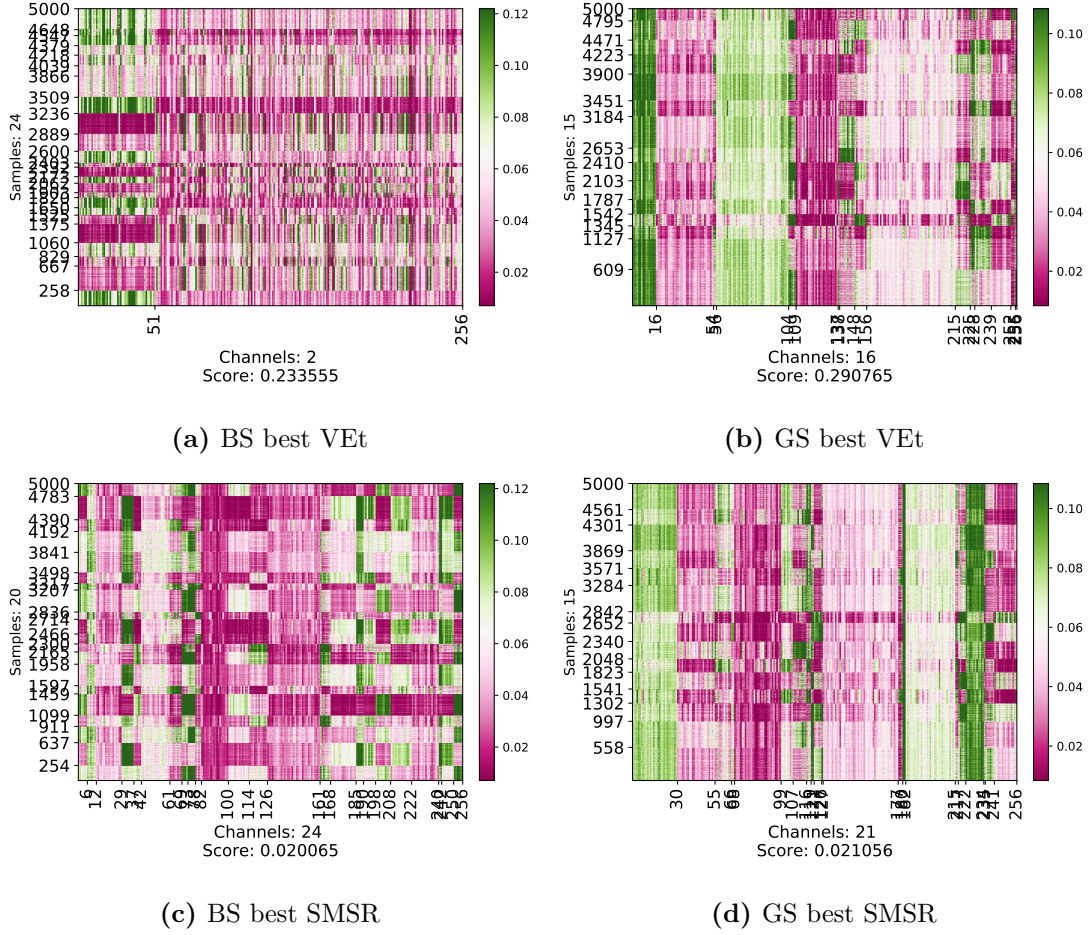


Figure 4.3: BS and GS Conv_1 representations sorted to the cluster labels generated by the best performing bicluster combinations.

We present the representations sorted by the cluster labels of the best scoring bicluster combination of VET and SMSR for Conv_1 in Figure 4.3. As mentioned in Section 4.3.2, we observe that BS has purer biclusters for both QM methods compared to GS. Visually inspecting the representation matrix reveals local structures that formed where samples had similar activation values for similar clusters. These are more visible in some of the sorted representations (Figures 4.3c and d) and not so visible in others (Figures 4.3a) which are difficult to interpret.

In GS Conv_1 we observe that there are channel clusters that have similar activation values for most samples, and can be presented as a single sample cluster, although they are divided into smaller biclusters, they are still prominent (Figures 4.3b and d). Other smaller biclusters are present, but they do not make up the majority of the biclusters, as with visual inspection there are more *channel clusters* than biclusters. Then again, when observing BS (Figure 4.3c), we see

that there are many smaller coherent biclusters that have formed compared to GS. This could be an indication as to why BS has more pure biclusters.

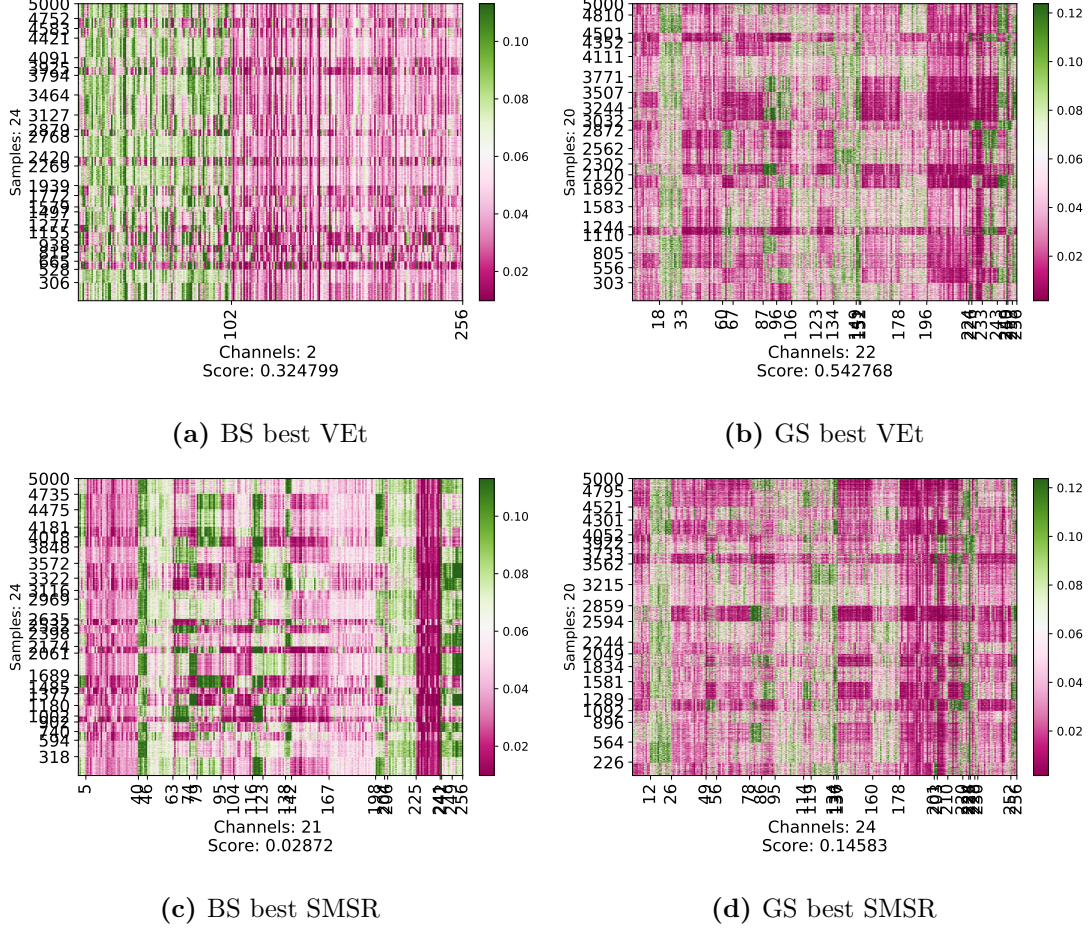


Figure 4.4: BS and GS Conv_2 Representations sorted to the cluster labels generated by the best performing bicluster combinations.

We present the best scoring bicluster combination for **Conv_2** in Figure 4.4. Here, both models have much more evenly distributed activation values, especially GS, making it more difficult to spot the local structures. GS also have less prominent channel clusters compared to **Conv_1**. However, the same observations are made for **Conv_2** as was done in **Conv_1** where GS has less visible biclusters and BS has more coherent biclusters.

Furthermore, we note that, for both layers, some of the biclusters can be combined and rearranged to form more coherent clusters. An example of this is visible in Figure 4.3c where some channels have similar activation values for some samples, but due to other sample clusters, these channel clusters are divided. If we aim for such biclusters, the structure of each sample and

channel, as well as the rectangular biclusters, will be disrupted; however, purer biclusters will emerge. Due to the limitation of forming a chequerboard pattern, this is not possible with the spectral biclustering algorithm that we use.

The best scoring bicluster combination results of the fully connected layer is presented in Appendix A.2.2.

4.4 Predicting generalisation

Given the correlation between cluster purity and generalisation ability of a model observed in Chapter 3, we want to investigate whether there is any correlation between bicluster quality scores and the generalisation ability of a network. It is evident that the different models have different numbers of optimal bicluster combinations, making it difficult to make a fair comparison. We investigate these optimal bicluster combinations in two different experiments to identify a trade-off combination on which we can compare all of the models fairly.

We first consider the relationship between the bicluster combinations and the QM scores in different models (Section 4.4.1) and then the impact of the number of sample clusters and channel clusters on the QM scores (Section 4.4.2). Using these results, we can construct a metric to predict generalisation (Section 4.4.3).

4.4.1 Investigating QM scores at different bicluster combinations

We consider the relationship between the bicluster combinations and the QM scores in different models. This allows us to see any patterns regarding the optimal scores and the bicluster combinations. If a pattern can be identified, it will not only allow for a fair comparison between the models but will also reduce the computational burden of performing a bicluster grid search for each of the 96 Task 1 (Section 3.3.1) models.

For this investigation, we use five randomly selected models from the PGDL Task 1 from which we extract 5 000 hidden representations for each model. Next, we perform a grid sweep that consists of 2 to 26 biclusters for both the samples and the channels, similar to what is described in Section 4.3.1. We then calculate the mean QM score (SMSR and VEt) of the five models for

each of the bicluster combinations. After which, the matrix is flattened to a vector, removing the grid structure of the bicluster combinations. The vector is then sorted in ascending order to be used as an index list by which each individual model’s QM scores are sorted.

The five models’ score matrices are also flattened and individually sorted according to the mean score index. This results in each of the models scores being sorted by the collective mean while still keeping the structure of the same bicluster combinations. If each model’s QM scores were sorted independently, the bicluster combinations would misalign due to the lowest or highest scores in one model not corresponding to the same bicluster combination in another model. We present each of the five models with SMSR scores sorted for Conv_1 in Figure 4.5 while the rest of these plots are available in the Appendix A.2.3.

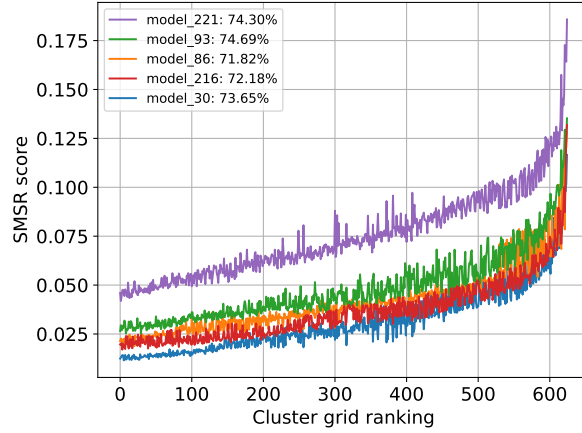


Figure 4.5: SMSR for each bicluster combination, for each of the five randomly selected models. The scores are sorted by the index of the sorted mean score of all five models in ascending order. Each colour represents a different model.

Figure 4.5 effectively illustrates why the bicluster combination index for each model does not align when sorted individually. Although each plot exhibits a similar trend, the order by ascending group mean does not result in a smooth line. These plots do not provide a suitable bicluster combination for comparing all models, as there is a lot of variation in scores between the models. This leads us to our next investigation of the impact of the bicluster combinations on the QM scores.

4.4.2 Investigating optimal bicluster combinations

In order to determine the effect of the bicluster combinations on the QM scores, we can use the same sorted mean score index as was used in the previous section. Since the bicluster combinations that produced each score are known, we are able to identify the number of the sample clusters and channel clusters of each of the same 5 models responsible for each score. We create a similar plot to Figure 4.5, where the x-axis represents the mean sorted score index and, instead of the score on the y-axis, the number of sample/ channel clusters of each model for that index. In addition, the samples and channels are presented on their own plots to enhance our observation of potential patterns. This permits a comparison to determine which number of clusters along the sample and channel axes is responsible for which scores.

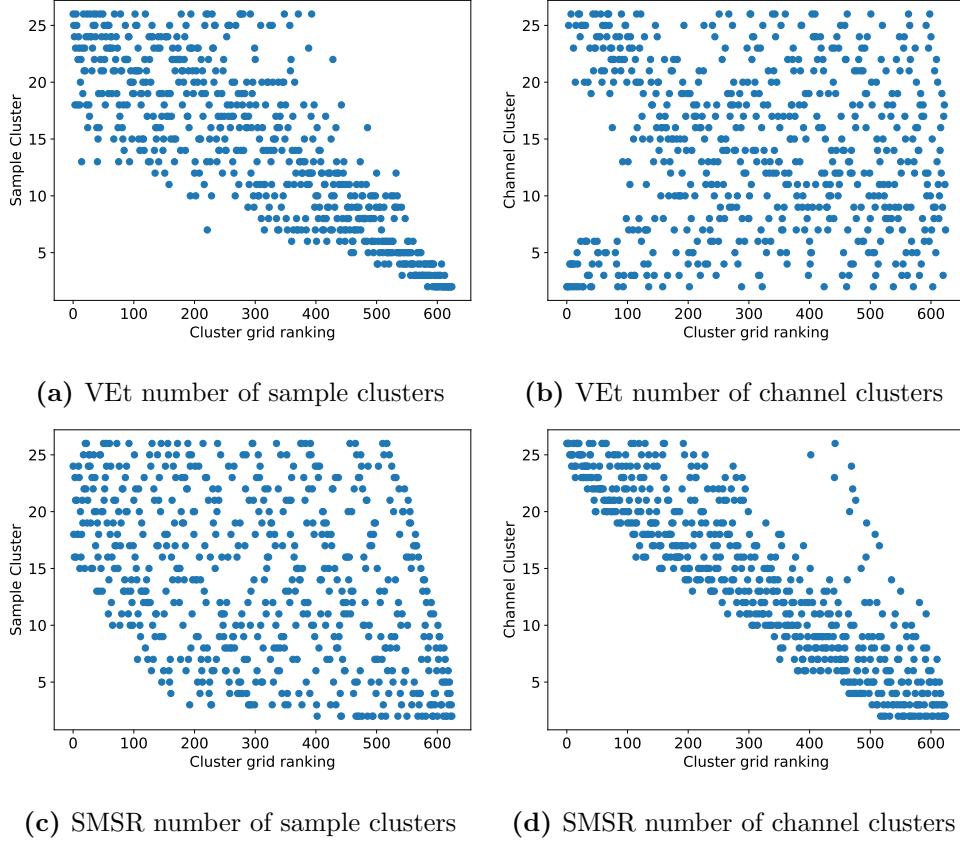


Figure 4.6: Number of sample and channel clusters responsible for each of the five sampled models Conv.2 QM sorted scores as is shown in Figure 4.5. The x-axis denotes the index of the mean scores when sorted in ascending order, while the y-axis denotes the number of clusters for the samples and channels respectively.

Upon examining Figure 4.6, it becomes evident that there exist strong correlations between the

best QM scores and the number of biclusters. Specifically, VEt achieves lower scores when there are more sample biclusters (Figure 4.6a), and SMSR achieves lower scores with more channel biclusters (Figure 4.6d). Although there is no strong correlation between the VEt score and the number of channel biclusters (Figure 4.6b), we observe that none of the better scores were achieved with biclusters in the range of 8 to 18. Likewise, for SMSR and the number of sample biclusters (Figure 4.6c), we observe that none of the better scores were achieved by the sub-15 sample biclusters.

As expected, we observe that the number of biclusters plays a role in the QM score, with some of the top-performing bicluster combinations being located at the search boundaries. This is not always the case with some strong combinations observed for a smaller number of sample clusters (both metrics) and a smaller number of channel clusters (VEt specifically). This makes it difficult to compare models, as we are unable to identify optimal bicluster combinations without doing a full grid search, which is prohibitively expensive. We consider alternative solutions in the next section.

4.4.3 Biclustering as a measure of generalisation

Given our expanded understanding of the chosen QMs and how they function, we can investigate whether biclusters can be used as a measure to predict generalisation. Since we perform our experiment on the entire set of PGDL Task 1 models, the number of representations is reduced to 1 000 for computational efficiency. Although this set of samples is smaller than what we used in previous experiments, it is still representative.

We make use of a limited selection of bicluster combinations as a result of previous experimental results (Section 4.4.1 and Section 4.4.2). This selection will serve as a basis for a grid search, as an effective representation of the optimal bicluster combinations found in each model. We select the values [2, 3, 12, 19, 27, 29] for both the number of sample and channel clusters based on the previous investigations. We have slightly increased our grid with a few more channel and sample clusters to account for models who prefer the added benefit of more biclusters.

We report the purity results in three different ways:

- Lowest QM score obtained on the grid search.

-
- Average QM score of the entire grid search.
 - The average on the diagonal of the grid search (2 by 2 to 29 by 29).

Although the QMs tend to the favour grid edges, the optimal bicluster combination is not always found at the edge of the grid, as previously observed in Section 4.4.2. This allows us to report on the minimum score for our first method. The other two methods originate from the concept of sampling the area below the curve with reference to Figure 4.5. Since we cannot calculate the whole area for each model by densely sampling the plot, we subsample the same bicluster combinations for each model. We suspect that this could be a more fair comparison, as each model is judged not on a single instance, but on an average result. This final method relies more on the QM bias of a better score when allowing for the use of more biclusters. This method might result in SMSR achieving lower scores than VEt, since we are essentially not measuring the better performing VEt bicluster combinations. However, we are primarily concerned with the results of biclusters being predictive of generalisation and not with the comparison between different QMs.

Table 4.4: Kendall’s rank coefficient between the evaluation accuracy and various QM score for PGDL
Task 1 small models

	VEt			SMSR		
	Minimum	Average	Diagonal	Minimum	Average	Diagonal
Conv_1	0.3702	0.4447	0.4945	0.4057	0.5033	0.5459
Conv_2	0.4767	0.5655	0.5530	0.6933	0.6578	0.6454
Lin_1	−0.4270	−0.5158	−0.5158	0.2690	0.4341	0.4660
Lin_2	−0.5580	−0.5362	−0.5580	−0.5217	−0.5145	−0.4638

We report our findings in Table 4.4 which shows the Kendall rank correlation (Section 2.5.1) between the QM score and the evaluation accuracy of each small model. We report each of the 3 measures: Minimum, average, and diagonal average scores for both SMSR and VEt. We also highlight the highest absolute value for each of the layers we examined. The significance values (p-value, Table A.2) are included in Appendix A.2.4 along with the larger model’s Kendall rank coefficient (Table A.3) and significance values (Table A.4).

When comparing the different scoring methods with each other, none consistently outperformed the other for either SMSR or VEt. However, we note that the diagonal average score performed the best overall on the VEt scores, while the minimum and the average diagonal scores had higher correlations for the SMSR scores. Therefore, settling for a single scoring method is challenging.

When comparing VEt and SMSR with each other, we observe that SMSR has higher correlations in the convolutional layers, while VEt has higher negative correlations in the fully connected layers. We also observe that VEt has a negative correlation starting at the fully connected layers, whereas SMSR only has a negative correlation in `Lin_2`. The strongest correlation is found in `Conv_2` where the minimum SMSR score is used.

Since a pure bicluster is indicated by a 0 score, the positive correlation indicates that the biclusters found in the poorly generalising models are more pure than those of their counterparts. This is a characteristic which we noticed in Section 4.3.3. Then again, in the fully connected layers, the better generalising models produce purer biclusters. This could confirm what was observed in Section 4.3.3 where poorly generalising models tend to have more specific activations in the convolutional layers while having more general activations in the fully connected layers. Whereas the better generalising models have more general activations in the convolutional layers while having more specific activations in the fully connected layers.

We also show the three measures on the SMSR scores in Figure 4.7 for `Conv_2`, which produced the strongest correlation with generalisation. Each of the methods produces similar graphs with slight variations between them and a scale difference in Figure 4.7a, which shows the minimum SMSR score. This is expected since it is the minimum score of each model. It is worth noting that the models that struggle with generalisation tend to have nearly identical lowest scores, as though it could not decrease any further. This might stem from the inability to organise the biclusters in a manner to produce a purer chequerboard pattern; since the samples consist of slightly different activation values, a perfect bicluster cannot be attained. We also present all the correlation plots for all layers of both QMs in Appendix A.2.4.

4.5 Discussion

When comparing BS and GS based on their respective best QM scores, we see that BS achieves purer biclusters compared to GS. This holds for both VEt and SMSR and for all three layers of the models. There is a particular large difference between the SMSR scores of `Conv_2`, where BS has a smaller score than GS with a magnitude of $1e^{-1}$. Observing Figures 4.4c and d, there are some differences that stand out. BS seems to activate more specifically, and its clusters being more uniform compared to those of GS. GS on the other hand seems to activate more

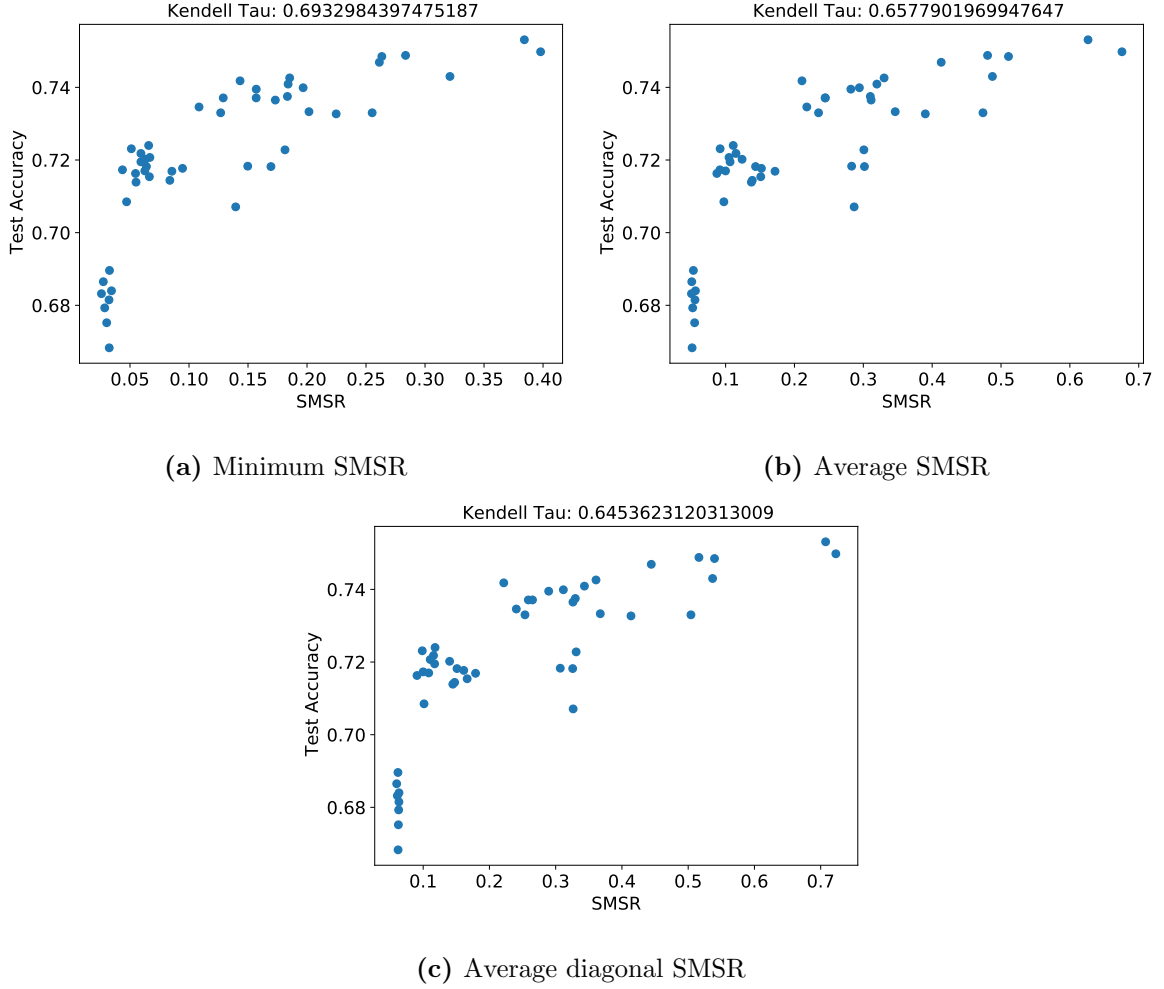


Figure 4.7: SMSR for each model for Conv_2, measured with each of the proposed methods and plotted against the evaluation accuracy of each model. We also indicate the Kendall's rank of each plot.

generally, without distinct clusters forming. The opposite is then observed in the fully connected layers, where GS seems to activate more *specifically* and forms more uniform clusters, whereas BS activates more *generally* and has no distinct clusters.

Analysing the numbers of clusters that the models perform the best on, we observe that when measuring purity with VEt, the best performing clusters are achieved with more sample clusters, while cluster purity is not as significantly affected by the number of channel clusters. Then again measuring the purity with SMSR we observe that the number of channel clusters influences the purity with more clusters, resulting in a higher purity, whereas the number of sample clusters does not influence the purity as much.

Furthermore, biclustering seems to be predictive of generalisation, where a purer cluster score

indicates a poorly generalising model. This holds for the convolutional layers, while we observe the opposite in the fully connected layers. Here, a lower VEt score indicates a better generalising model for both layers, while a lower SMSR score only indicates a better generalising model in `Lin_2`. We also observe a lower bound in the purity, where no model can achieve a lower purity score. This could be due to the variance in each sample representation and that a *perfect* bicluster is unachievable in this case. However, we can still distinguish between models with different generalisation gaps given this information.

Although useful in our quest, the spectral biclustering algorithm did not produce the results we are after, but rather ‘perfect’ biclusters in the sense of similar features that have similar values across different samples and channels. We are specifically interested in biclusters in the sense of features that vary over specific samples, such as in Figure 4.3b. This is not possible with this biclustering algorithm as it is restricted by the chequerboard pattern and the need to maintain the structure of the sample and channel corresponding to each feature. This was observed in Figures 4.3 and 4.4, where visual inspection revealed that the clusters can still be rearranged to produce purer biclusters.

Another constraint that complicates the usage of the spectral biclustering algorithm, is the need to specify the number of row and column biclusters. Given that our research targets data without ground truth values, grid searches are conducted to optimise this hyperparameter, making it computationally expensive.

4.6 Conclusion

In this section, we investigated whether there are any local structures in representations as well as what the differences in these structures are when comparing models with different generalisation gaps. We found that there are local structures where the channels are activated for more specific samples. We also found that poorly generalising models produce purer biclusters compared to their counterparts in the convolutional layers. The opposite is also true when comparing the fully connected layers, where the better generalising models produce purer biclusters. This could be due to *specific* activation patterns resulting in purer biclusters while *general* activation patterns result in lower bicluster purities.

We also show that the spectral biclustering purity measured using VEt or SMSR is somewhat predictive of generalisation, where a low purity in the convolutional layers and a higher purity in the fully connected layers are indicative of a model with a better generalisation ability. We also see that the poorly generalising models reach a purity threshold in the convolutional layers where a higher purity cannot be obtained, indicating a lower bound in the purity which cannot be crossed. This holds for the PGDL Task 1 models that we observed.

We now turn our focus to investigating representation summary methods that aim to reduce the dimensionality of representations while minimizing information loss. This investigation is detailed in Chapter 5.

Chapter 5

Autoencoder representations

“It’s only half completed, I’m afraid – we haven’t even finished burying the artificial dinosaur skeletons in the crust yet,”

- Douglas Adams, The Ultimate Hitchhiker’s Guide to the Galaxy Page 116 [1]

5.1 Introduction

In this section, we explore a technique to reduce the dimensionality of CNN representations to facilitate a more streamlined exploration thereof. We introduce a novel approach by making use of Autoencoders (AEs) to achieve this reduction. By training an AE on the representations produced by a ReLU-activated layer, we aim to compress the representation to retain only the most important features. This approach not only reduces dimensionality but also purifies the representation, improving its interpretability and exploration. This chapter primarily functions as a proof-of-concept that demonstrates this idea.

In Section 5.2, we discuss our methodology and implementation detail. Following this, Section 5.3 outlines the experimental setup we use for the AE, including the parameter settings and hyperparameter tuning processes involved. Subsequently, Section 5.4 provides an analysis of the training performance achieved by the AEs, as well as a comparison of their effectiveness

against other methods to reduce representations.

5.2 Methods

This section provides a detailed description of our methodology (Section 5.2.1) and also elaborates on the implementation process (Section 5.2.2). We introduce an AE as the key component in our approach, explaining its role in dimensionality reduction. To avoid confusion, we refer to the primary network in which the AE is embedded as “the model,” while the autoencoder itself is consistently termed “the AE.”

5.2.1 Methodology

One primary challenge we face when clustering CNN representations, is their considerable size ($\pm 40\,000$ features per sample, depending on the network architecture). This is influenced by both the number of channels employed in each hidden layer and the dimensions of the representation feature-maps generated by these channels. Unfortunately, we cannot simply train smaller networks, as their capacity for generalisation is partly due to their size, and we are specifically interested in their existing representations in a realistic setting. However, we have reduced the representations using the mean activation value of each channel, leading to a loss of some information as we condense an $n \times n$ matrix into a single value. This loss of information motivates us to investigate alternative methods to effectively reduce the dimensionality of representations, preserving only the most critical features, while maintaining the inter-channel and feature relationships.

Autoencoders (AE) are unsupervised machine learning models that compress their input and reconstruct it from the compression (Section 2.4.3). When used in an encoder-decoder configuration, the encoder reduces the input to a latent dimension, after which the decoder reconstructs the input from the latent dimension. Given the unsupervised nature of the AE, the model is trained to identify underlying patterns in the data that accurately reflect its characteristics to perfectly reconstruct the input. This satisfies our requirement of dimensionality reduction while preserving the essential characteristics of the input. Even though principal component analysis (PCA) could also be used as a dimensionality reduction tool, it is not capable of preserving

the non-linear relationships in the representations, due to the linear transformation that PCA employs. (We discussed dimensionality reduction techniques in Section 2.4.)

Our aim is to incorporate an AE between each of the convolutional layers of a model whose representations we want to examine. We only focus on the convolutional layers, as the fully connected layers representations have sufficiently low dimensionality.

5.2.2 Autoencoder implementation

To implement our idea, we develop a method that seamlessly integrates with the original model without replacing or altering any of its layers. We start off by loading the trained parameters of the model, freezing the layers so as not to change the parameters of the model but only those of the AE during backpropagation. This includes a shortened version of the trained model, which only consists of the layers succeeding the layer of interest on which the AE is trained. This shortened model is used to calculate the loss and accuracy of the model with the reconstructed representation (the AE’s output) as input.

Our approach involves attaching the AE to the model at the point where the representation is generated, allowing us to train the AE separately from the model itself and subsequently produce a latent vector using the models’ representation as input. We provide a visual representation of the implementation for `Conv_1` of a small architecture model in Figure 5.1.

During the training process, each batch of samples is passed through the model where the representations of the layer of interest ($\mathbf{X}^l$) are recorded and the model output ($\hat{\mathbf{y}}$) is produced. Next, the current layer’s batch of generated representations ($\mathbf{X}^l$) is passed through the AE, producing the output at the decoder, referred to as $\hat{\mathbf{X}}_{\text{AE}}^l$. The next step involves passing $\hat{\mathbf{X}}_{\text{AE}}$ through the shortened model to produce the classifications ($\hat{\mathbf{y}}_{\text{AE}}^l$).

After each batch of representations, the new gradients of the AE are calculated and updated using backpropagation. Again, only the parameters of the AE are updated, and not those of the original model. This process is repeated for each batch of input samples.

The recreation loss of the AE can be measured as:

$$\text{Loss} = L(\mathbf{X}, \hat{\mathbf{X}}_{\text{AE}}) \quad (5.1)$$

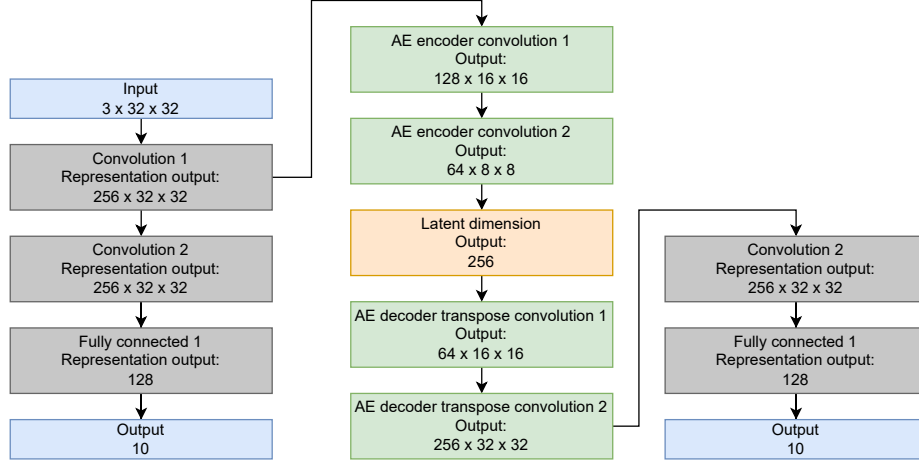


Figure 5.1: Autoencoder implementation for `Conv_1` of a small convolutional model. Grey block represents the frozen layers, green blocks indicate convolutional layers of the AE and orange blocks indicates fully connected latent dimensions. Included is a shortened model that uses the AE as input. Representation sizes is included for each layer.

with L being the specified loss function while the model loss can be measured as:

$$\text{Loss} = \ell(\hat{\mathbf{y}}, \hat{\mathbf{y}}_{\text{AE}}) \quad (5.2)$$

where ℓ is a standard loss function such as mean squared error (MSE) or binary cross-entropy. The accuracy of $\hat{\mathbf{y}}_{\text{AE}}$ is calculated and compared to the true sample label and the label produced by the model. The AE is tested on the validation set every five epochs, and the resulting validation loss and accuracy metrics are reported.

The AE is trained using the same dataset used for the training of the model, which includes using the same validation and evaluation sets for their respective purposes. Each AE for each layer is individually trained without the involvement of any preceding AEs. The implementation also allows for the training of a single AE for a specific layer, instead of an AE for each layer. Other functionalities include the option to save the parameters of the AEs to save memory, which proves useful in the case of hyperparameter sweeps where not all models will be used.

5.3 Experimental Setup

In this section, we provide a detailed overview of the AE setup, including the specific hyperparameters chosen to optimise its performance. We discuss the architecture (Section 5.3.1) and hyperparameter selection process (Section 5.3.2) that ensure that the autoencoder effectively captures the most relevant features during dimensionality reduction. Finally, we outline the experimental procedures (Section 5.3.3), explaining how we intend to compare the effectiveness of the autoencoder in compressing and purifying feature representations against other reduced representations.

5.3.1 Autoencoder setup

The models we use for this demonstration are GS and BS, as introduced and discussed in Section 3.3.2. Both are two-layer CNNs with a single fully connected layer preceding the output, making them ideal for this proof-of-concept demonstration. They demonstrate a significant generalisation gap difference between them and their architecture is small enough to implement our novel approach without requiring extensive training of multiple AEs or hyperparameter optimisation. A limitation of using the PGDL models is that they do not have a validation set, forcing us to use 5 000 of the training samples as a validation set for the AE. As a result, the models (GS, BS) have both fitted the data that we use as a validation set, although the AE has not.

Given that GS and BS are CNNs, we opted for an AE that is also CNN-based. The AE has an input and output size that correspond to the dimensions of the models representation. The encoder and decoder consist of two convolutional layers each, where the encoder has the number of channels $[C_i, C_i/2, C_i/4]$ and the decoder $[C_i/4, C_i/2, C_i]$, where C_i refers to the number of channels in the subsequent layer. Both encoder and decoder have BatchNorm layers after each ReLU activated convolutional layer. The latent dimension consists of a fully connected layer whose size is a hyperparameter that can be set before initialisation. The encoder makes use of a kernel size of 5, padding of 2 and a stride of 2 for both convolutional layers, while the decoder uses a kernel size of 6 and a stride of 2. Additionally, the padding of the decoder is calculated during initialisation so that the output of the decoder matches that of the representation.

We choose reducing the model loss as the optimisation metric, resulting in the key features being captured within the latent dimension. The key features being those that exert the most significant influence on the model’s output, rather than features solely capable of reproducing the representation. The loss is determined using the mean squared error (MSE) function. The training process is stopped either when the AE interpolates the training data or when the set maximum number of epochs is reached. We also employ early stopping to record the best model at the lowest validation model loss. For consistency between AEs, we use the same initialisation seed of 10. The rest of the hyperparameters are set according to the selection process which we cover in Section 5.3.2.

5.3.2 Optimal hyperparameter selection

Given that this is an exploratory approach, we are uncertain which specific hyperparameters will have the greatest impact on performance. To address this, we conduct small hyperparameter sweeps using a limited number of epochs, aiming to identify the hyperparameters that most influence the training dynamics. This process helps us to refine our hyperparameter choices and fine-tune the models effectively while managing computational resources and time.

Our initial hyperparameter sweeps use SGD and Adam as optimisers. For Adam and SGD, we use the learning rates [0.01, 0.001, 0.0001]. We also test the effect of a learning rate schedule implemented every five epochs with a momentum of [0.99]. For the latent dimensions, we test the values [32, 64, 128, 256, 512, 1024, 2048, 4096], with batch sizes [16, 32, 64, 128, 256]. The AEs were trained for 50 epochs, which proved adequate to assess the performance of the selected hyperparameters.

We also use three other optimisers, DadaptAdam, DadaptSGD, and Prodigy [41], [45] with a learning rate of [1.0] (default value), all of which requires less tuning. Dadapt is a hyperparameter-free approach to automatically set the learning rate to achieve optimal convergence rates, while Prodigy is a modification of Dadapt that significantly improves its convergence rate. (We covered these methods in more detail in Section 2.2.3.) Although these optimisers do have changeable hyperparameters, we leave all of them at the default, since this is a proof-of-concept instead of a fully optimised solution, and our main focus is on attaining viable results rather than determining the optimal hyperparameters. In addition, the authors achieved competitive results during

their study [41], [45].

Following the initial sweep, we choose DadaptAdam as optimiser as it produced the best performing AEs overall. Furthermore, we choose 256 nodes to use in the latent dimension as it matches C_i , offering the same dimensionality as employing the maximum or mean across each channel. Although we did perform a sweep of different latent dimensions, we did not observe any significant performance differences, and thus decided to use a fixed latent dimension that is comparable to our other reduced representations. For the AEs of BS, we opt for a batch size of 265, while those of GS uses a batch size of 16. We train the final models for 250 epochs, which is sufficient for the model’s convergence. The final performance measures of the AEs are reported in Section 5.4.1. A visual implementation of the `Conv_1` AE with layer sizes is presented in Figure 5.1.

5.3.3 Comparing different representations

To observe whether the AE is effective at dimensionality reduction and whether the key features are indeed captured in the latent dimension, we compare it against traditional representation summary methods (TRSM). Specifically, we use the raw representation ($\mathbf{X}^l \in \mathbb{R}^{N \times CHW}$), the average of each channel ($\mathbf{X}^l \in \mathbb{R}^{N \times C}$), the maximum of each channel ($\mathbf{X}^l \in \mathbb{R}^{N \times C}$) and the PCA dimensionally reduced representation as benchmarks for comparison. For the PCA we use 256 principal components since all other representations, except raw, have these dimensions, which would lead to a fair comparison.

To compare the TRSM with the latent dimension, we use class-based and class-agnostic clustering approaches, as described in Sections 3.3.6.1 and 3.3.6.2. We again use 5 000 training samples to produce the representations. For the class-agnostic clustering, we set the potential number of clusters from 2 to 25, selecting the optimal cluster count based on the lowest DBI score. As the fully connected layer is sufficiently small, we have neither TRSM nor a latent dimension to compare against each other; hence, we exclude it from the results. We present our class-based results in Table 5.6 (Section 5.4.2) and the class-agnostic results in Table 5.7 (Section 5.4.3).

5.3.4 Predicting generalisation using latent dimension representations

In order to further investigate whether AEs are a viable method for investigating representations, we compare models with different generalisation gaps. We specifically choose models from Task 1, which are all equally spaced, based on their evaluation scores. Among these six models, GS and BS are included. We present these models along with their respective accuracies and hyperparameters in Tables 5.1 and 5.2. All of the models selected only have one fully connected layer, which makes for a more fair comparison.

Table 5.1: Chosen PGDL Task 1 models performance metrics sorted in ascending order of evaluation accuracy.

Model	Train Accuracy (%)	Evaluation Accuracy (%)
Model 29	100.00	75.31
Model 29	99.97	74.18
Model 20	99.97	73.30
Model 22	100.00	72.02
Model 87	99.92	71.44
Model 90	100.00	66.83

Table 5.2: Chosen PGDL Task 1 models and corresponding model-specific training hyperparameters.

Model	Channels	Batch Size	Drop probability	Weight Decay
Model 29	512	8	0.5	0.001
Model 85	512	32	0.5	0.000
Model 20	256	32	0.5	0.001
Model 22	256	32	0.0	0.001
Model 87	512	32	0.0	0.000
Model 90	256	512	0.0	0.000

The same training procedure is followed as in Section 5.3.1, while we also use the same hyperparameter selection approach as discussed in Section 5.3.2. However, there is a change as we only use the final selection process, as we use a fixed latent dimension of size 256 while only performing a search for different batch sizes. Although models 29, 85, and 87 have 512 channels in both layers, we still use a latent dimension of 256 to make them comparable with the other models. The training results for `Conv_1` are presented in Table 5.4, and in Table 5.5 for `Conv_2`.

To compare the models against each other, we use the class-based DBI score, as it has been shown to be predictive of generalisation when used on model representations [18]. The method used is the same as described in Section 3.3.6.1. Again, 5 000 training samples are used and the DBI is compared with the evaluation accuracy of the model and not that of the AE. Our

findings are presented in Section 5.4.4.

5.4 Results

In this section, we present the performance metrics of the best performing AEs that were trained to reduce the dimensionality of the BS and GS representations (Section 5.4.1). We analyse variations of the representations, including raw, average, maximum, and AE latent dimensions. These representations are compared in both a class-based (Section 5.4.2) and a class-agnostic (Section 5.4.3) clustering setting, to highlight differences in the purity of the formed clusters.

5.4.1 Model results

We present the performance metrics of the best performing AEs in Table 5.3 for both BS and GS. Each model has two AEs due to their two convolutional layer architecture. All AEs are trained according to the hyperparameter selection as presented in Section 5.3.2. The optimal hyperparameters are selected on the basis of the lowest validation model loss produced by the first and second AE in each model.

All AEs reached near interpolation (fitted all training samples) except BS Conv_2. Although the models fitted the validation set, all the AE seems to struggle more with it. This is more evident when looking at the evaluation set, as there is a $\pm 10\%$ difference when comparing the validation and evaluation accuracies. Although higher accuracies could be achieved, we achieved the best performance given our limited compute resources.

Table 5.3: AE performance metrics for both GS and BS. Included is GS and BS performance metrics for comparison. Also includes the evaluation delta which is the difference between the AE evaluation accuracy and that of the model.

Model	Train (%)	Validation (%)	Evaluation (%)	Evaluation delta (%)	Best Epoch
BS Conv_1	99.98	78.88	68.35	1.52	225
GS Conv_1	99.36	80.42	73.55	0.25	65
BS Conv_2	95.73	77.98	68.11	1.28	30
GS Conv_2	99.85	81.88	72.01	-1.29	205
BS	100.00	–	66.83	00.00	–
GS	99.97	–	73.30	00.00	–

Despite the AEs struggling with the validation set, both reached evaluation accuracies close

to those achieved by the models, suggesting that the AEs can recreate most (if not all) of the representations that the models can correctly classify. In particular, all AEs achieved better accuracy than the standard model on the evaluation set, except GS in `Conv_2`. This could indicate that the AEs compressed the representations down to the key features essential for the models classification ability, which allows the model to achieve better generalisation performance. This alone suggests that reducing the representation dimensions with the help of an AE can result in isolating the key features used in the model decision making process. Although it is difficult to compare the AEs of GS and BS with each other due to different training data and training procedures, we do however note that both AEs of BS improved it’s generalisation performance, while only the `Conv_1` AE of GS slightly improved it’s performance and the `Conv_2` AE resulted in a loss of performance. This might result from the fact that GS has likely learnt better, more generalisable features, which are harder to capture with the AE in comparison to BS.

We also present the training results for the other models in Table 5.4 (`Conv_1`) and Table 5.5. Here we observe that the AEs of the models that generalise the best also perform the best on the evaluation set. However, we note that this does not relate to improving the evaluation delta, meaning that better generalising AEs does not improve the model performance but slightly decreases it. A conclusive answer cannot be provided due to the absence of a consistent pattern. As previously mentioned, the only AE that consistently improved the model was BS.

Table 5.4: AE performance metrics `Conv_1`. Also includes the evaluation delta which is the difference between the AE evaluation accuracy and that of the model.

Model	Train (%)	Validation (%)	Evaluation (%)	Evaluation delta (%)	Best Epoch
Model 29	99.82	79.98	74.08	-0.12	60
Model 85	99.82	80.50	72.80	-1.38	160
Model 20	99.36	80.42	73.55	0.25	65
Model 22	99.98	81.04	71.59	-0.43	250
Model 87	99.86	82.92	69.22	-2.22	205
Model 90	99.98	78.88	68.35	1.52	225

Table 5.5: AE performance metrics for `Conv_2`. Also includes the evaluation delta which is the difference between the AE evaluation accuracy and that of the model.

Model	Train (%)	Validation (%)	Evaluation (%)	Evaluation delta (%)	Best Epoch
Model 29	99.96	83.04	72.81	-2.5	230
Model 85	99.85	81.22	73.39	-0.79	220
Model 20	99.85	81.80	72.01	-1.29	205
Model 22	96.26	81.68	71.60	-0.42	20
Model 87	99.87	83.94	69.97	-1.47	240
Model 90	95.73	77.98	68.11	1.28	30

5.4.2 Class-based clustering analysis

We now examine the results of the class-based clustering analysis, as described in Section 5.3.3. Table 5.6 presents the DBI scores for both BS and GS using TRSM and the latent dimension. When comparing the results across the TRSM and the latent dimension, we observe that the raw representations perform the worst, followed by the maximum value of each channel and PCA. The average of each channel is similar to the performance of the maximum, but is the best among the TRSM. Finally, the latent dimension significantly outperforms all other methods, indicating a marked improvement in clustering performance.

Table 5.6: GS and BS DBI scores for class-based clustering. TRSM as well as the latent dimension of the autoencoder trained on the representation are shown.

	Raw	Maximum	Average	PCA	AE Latent Dim
BS Conv_1	11.52	11.56	10.99	10.77	6.31
GS Conv_1	13.02	11.76	10.16	11.54	4.63
BS Conv_2	11.94	10.15	9.92	10.98	4.85
GS Conv_2	16.00	9.33	9.05	10.96	2.94

Analysing the differences between the layers, we observe that Conv_2 performs consistently better than Conv_1, in both models and for all representations, except for the raw representations and PCA. The same behaviour is also observed in Section 3.3.6.1. Lastly, we compare the performance of GS and BS. GS surpasses BS in all situations, having a lower DBI score, with the exception of Conv_1 maximum channel and PCA, as well as Conv_2 raw representations. Overall, the latent dimension DBI is the purest we have seen of the representations of these models so far, when considering class-based clustering. This could be indicative of key features that the model uses to distinguish between the classes of the dataset, being preserved in the latent dimension.

5.4.3 Class-agnostic clustering analysis

We now turn our attention to the class-agnostic clustering results (Table 5.7). Here, the latent dimension produces the worst DBI scores compared to the TRSM, although its DBI is lower than that observed in the class-based clustering results. Furthermore, the average channel representations performed the best overall. Turning to the comparisons between the layers, here

we observe that `Conv_2` performs worse than `Conv_1` for both BS and GS, in all cases except for the latent dimension, where `Conv_2` has purer clusters than `Conv_1`. Lastly, comparing GS and BS, the latter outperforms GS for all cases except that of the latent dimension, where GS again has purer clusters.

Table 5.7: GS and BS DBI scores for class-agnostic clustering. TRSM and the latent dimension of the autoencoder trained on the representation are shown. Representations are clustered with k-means between 2 and 25 optimal clusters with the DBI score of the best-performing number of clusters being shown.

	Raw	Maximum	Average	PCA	AE Latent Dim
BS <code>Conv_1</code>	2.71	1.58	1.00	2.48	4.14
GS <code>Conv_1</code>	3.69	2.36	1.29	3.16	2.87
BS <code>Conv_2</code>	3.74	2.43	1.27	3.37	3.73
GS <code>Conv_2</code>	8.04	3.10	2.16	4.99	2.62

We again observe that the latent dimension performs differently from the rest of the representations as observed in Section 5.4.2. Although it did not produce the purest clusters for this type of clustering, compared to the TRSM, it is the only representation in which GS outperforms BS showing that more similar features are clustered together. This indicates that the key features which are responsible for the generalisation performance of both models are correctly captured in the latent dimension. It is also the only representation where `Conv_2` is more pure than `Conv_1`, again indicating that the AE captures the most prominent features attributed to the generalisation ability of these models. The number of optimal clusters for each method is presented in Table A.5.

5.4.4 Predicting generalisation with autoencoders

The class-based DBI score is plotted against the evaluation accuracy of each model, and is presented in Figure 5.2. For both layers a high negative correlation is observed, with `Conv_1` having a higher correlation than `Conv_2`. A decrease in DBI scores is also observed between the layers with `Conv_2` having lower scores compared to `Conv_1`.

These results indicate that AEs can be used as an effective representation dimensionality reduction tool. Although this method can be used as a generalisation prediction method, even more than that, it presents a way in which key features in representations can be investigated. These

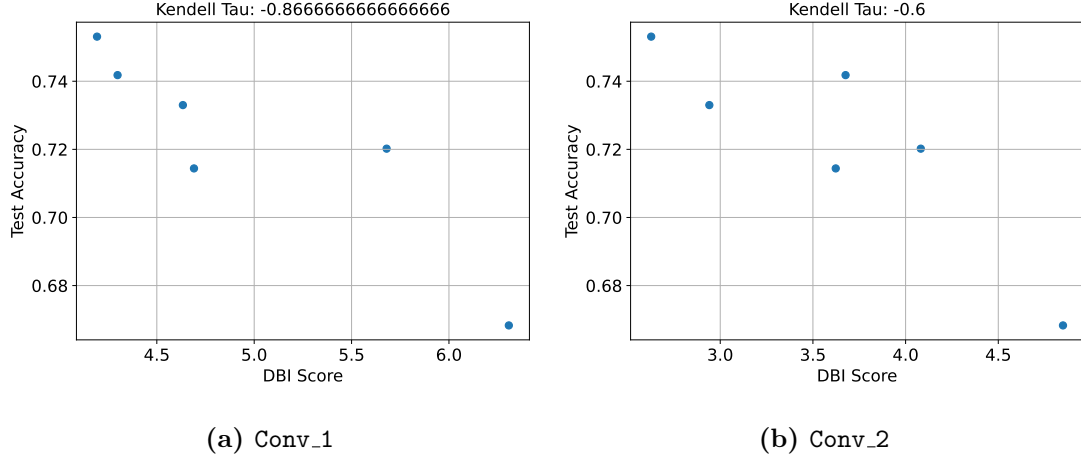


Figure 5.2: Layer wise class-based DBI score of each of the five AE plotted against the evaluation accuracy of each respective model. We also indicate the Kendall’s rank of each plot.

key features are trivial for the model in order to make the correct classifications. The fact that the AE is able to preserve these key features lies in its ability to compress the representations to fewer dimensions and still retain class-specific features, indicated by the high correlation between the evaluation accuracy and the DBI score. This correlation also indicates that better generalising models have better class-specific key features than poorly generalising models.

5.5 Discussion

Both AEs of both BS and GS performed well on the training data and, despite challenges with the validation samples, produced representations that improved the generalisation performance (evaluation accuracy) of the original models. Since the AEs can only extract features from the representations, not directly from the samples, their performance is inherently limited to that of the original models. This suggests that AEs are a viable method of investigating the representations by reducing their dimensions.

We explored the latent dimensions to a limited extent, which consisted of a small experiment to observe the effects of the latent dimension and compare it with other traditional representation summary methods (TRSM). The results we obtained indicate that the latent dimension behaves differently compared to other TRSM. Firstly, the latent dimensions were the only representation in which the models were correctly ranked considering the generalisation gap for both clustering

methods. All TRSM ranked BS higher than GS in the class-agnostic clustering with BS having more pure clusters. A fact that was more clearly shown when investigating the latent dimension purity of models with different generalisation gaps. Secondly, the latent dimension achieved the lowest DBI scores in the class-based analysis, and although it did not produce the purest clusters in the class-agnostic analysis, it is the only method where `Conv_2` had purer clusters compared to `Conv_1`, which is a characteristic expected in the models. This indicates that the latent features are representative of the representation features that affect the decision-making process of the model. We hypothesise that these features are in some way responsible for separating classes/object features that enables the model to distinguish between classes.

We also investigated whether the compressed representations, when clustered by class-based measures, correlate with the generalisation ability of the model. In fact, these clusters are predictive of generalisation with purer clusters correlated with better generalising models, with `Conv_1` being more predictive than `Conv_2`. It was also found that these compressed representations preserve the key features, with the better generalising models having more class-specific features than their counterparts. It is also observed that these features become more class-specific in the later layers as `Conv_2` was observed to be more pure than `Conv_1`.

Given that we need to train the same number of AEs as there are convolutional layers, the operation quickly becomes highly computational. Another factor adding to the computational complexity is the need for the decoder, although our primary interest lies in the latent vector produced by the encoder, it is important during the training process. In addition, the model also needs to generate representations during the AE's training, along with the new output on the reconstructed representation, to calculate the loss associated with the reconstruction.

Since this is a proof of concept, we did not investigate this further to determine whether the results hold across different architectures and datasets. Additionally, our results show that reducing the representation size by using the average feature value of each channel captures the most essential information when a less computationally expensive method is needed, even outperforming the raw representation itself. Due to time constraints, an optimal hyperparameter search was not feasible, which may have affected our results. Although further improvements could likely be made with a more thorough hyperparameter search, we achieved satisfactory model performance.

5.6 Conclusion

This chapter presents a method to mitigate the high dimensionality of representations by using an autoencoder to compress representations to a practical size while maintaining the key features. This is done by training AEs on the representations of each of a model’s convolutional layers to reproduce the representation. The AE compresses the representation down to a latent dimension where the key features are captured. Using the latent dimension representation rather than the raw or modified representation, we aimed to retain the most key features, thereby enhancing our ability to analyse the representations.

From our results, we show that the latent dimension behaves differently from that of other TRSM, being more pure during class-based analysis and becoming more pure across the layers in class-agnostic analysis. Although we present the findings for just two models without performing a comprehensive hyperparameter search for the AEs during training, we demonstrate that this approach effectively reduces representations. Simultaneously, it retains critical features, potentially improving our insights into which features within a representation significantly influence a networks generalisation ability.

We also briefly investigated whether these compressed representations are predictive of generalisation and found that they are in fact predictive with better generalising models having purer class-based clusters. It was observed that `Conv_1` is more predictive than `Conv_2`. Although this was only done for five models, the results are promising. It also strengthens the hypothesis that better generalising models have better key features, and that these key features are responsible for the models’ generalisation ability.

This is a useful step in the direction of exploring the core features within the representations that the model uses for its decision-making process. Furthermore, this proof of concept demonstrates that training AEs to reduce the dimensionality of representations prior to further analysis is a viable investigative approach, one that we hope to explore further in the future.

Chapter 6

Conclusion

“’Look’ said Arthur, ‘would it save you a lot of time if I just gave up and went mad now?’”
- Douglas Adams *The Ultimate Hitchhikers guide to the galaxy* Page 109 [1]

6.1 Introduction

The purpose of this study (as discussed in Chapter 1) was to investigate whether there are clusters formed by the hidden representations, and if they do form, what effect these clusters have on the generalisation ability of a network. Specifically, we are interested in how these clusters differ between models with different generalisation gaps. This chapter evaluates the extent to which our initial objectives have been achieved. We provide a summary of our study, along with our key findings, as well as the implications of these findings. Lastly, we discuss any future research that might stem from this study.

6.2 Summary

In this study, we started by investigating representations under different conditions and comparing those of models with different generalisation gaps from the PGDL challenge (Section 3.3).

We also investigated whether class-based as well as class-agnostic cluster purity is a useful cluster characteristic to investigate, before moving on to studying the characteristics of representations, specifically also investigating the sparsity in models trained with and without BatchNorm (Section 3.4).

Moving on from these two case studies in the pursuit of sample-feature clusters, we investigated the local structures found in the representations by using a biclustering algorithm to form clusters of both samples and the channels (Chapter 4). Furthermore, we aimed to determine whether bicluster characteristics are predictive of generalisation. This was investigated by measuring the purity of the biclusters and assessing their correlation with the evaluation accuracy of the model. Lastly, we attempted to mitigate the complexity of the high dimensional representations by compressing them with an autoencoder (Chapter 5): A solution that not only reduces their dimensions but also extracts their key features by removing any unnecessary noise. Finally, we briefly investigate whether these latent dimensions are predictive of generalisation.

6.3 Key findings

In this section, we discuss the key findings of this study and the implications thereof. We isolated three key findings; the first being representational clusters, in which we review our findings of each clustering method we used. Next, we analyse the characteristics we found within the representations as well as the cluster characteristics. Lastly, we discuss the key features within the representations and how these affect the generalisation ability of a model.

6.3.1 Representational clustering

In this study, we investigated the clusterability of hidden representations and how these clusters' characteristics relate to the generalisation ability of a model. During this investigation different models were used, such as selected models of the PGDL Task 1, standard CNNs trained with and without BatchNorm, and also AEs that compress the representations of other models. Our findings differed according to the clustering method we used, which included: class-based, class-agnostic (k-means), and biclustering.

We found that when using class-based clustering methods, considering the sample class label

as the cluster label, better generalising models produce purer clusters. This was observed for all the experiments that we performed and on different architectures (Sections 3.3.6.1, 3.4.3.1, and 5.4.2). This also correlates with the findings of Natekar et al. [18]. However, we did find that in some cases when summarising the representations with PCA, the maximum value of each channel, or the unaltered representations themselves, the poorly generalising models produced purer clusters (Table 5.6). We discuss this specific finding further in Section 6.3.3. However, these results indicate that the better generalising models internally cluster the representations better based on class compared to the poorly generalising models. This was especially observed in the fully connected layers, where the better generalising models produced significantly purer representation clusters, indicating superior class classification.

When investigating class-agnostic clustering methods (k-means), considering algorithmically assigned cluster labels, it is found that poorly generalising models tend to have purer representational clusters in the convolutional layers (Sections 3.3.6.2, 3.4.3.2, and 5.4.3). However, we found that the better generalising models have purer clusters in the fully connected layers. These results were observed in all of the experiments we performed and for most models, with the exception of the VGG-16 BatchNorm models (Figure 3.14b), as well as the AEs (Table 5.7). In these two cases, the better generalising models had purer clusters in the convolutional layers. After investigating the number of optimal clusters generated, we uncovered that the poor generalising models tend to have fewer optimal clusters, while the better generalising models have a range of different numbers of clusters which tends to the number of classes in the dataset in the last layers.

Similar results to those of the class-agnostic clustering were observed when biclustering the representations (Table 4.4). Again, the poorly generalising models had purer biclusters in the convolutional layers compared to the better generalising models. However, in the fully connected layers we observed that the better generalising models have purer biclusters, with the exception of `Lin_1` when measuring the purity of the biclusters with SMSR.

6.3.2 Representation characteristics

In this study, we investigated the characteristics of clustered representations and how these characteristics affect the generalisation performance of models. One of the characteristics that we

observed was a scale difference in the activation values between layers of a network. We specifically note that the activations of the better generalising models seem to stay constant throughout the layers, while those of the poorly generalising models seem to grow larger (Figure 3.2).

Another characteristic that we investigated was the sparsity of representations. First, we found that BatchNorm does not induce or prevent sparsity, since both cases were observed during this investigation. We found that BatchNorm-trained standard CNNs were less sparse than their counterparts (Figure 3.11a), while BatchNorm-trained VGG-16 models were consistently more sparse than their counterparts (Figure 3.11b). We also observed that all channels had the same sparsity and that no channel was more or less sparse than another in the same layer (Figures 3.9, 3.10).

We also investigated the sparsity of representations in selected models of the PGDL Task 1 (Section 3.3.3). During this investigation, we found that the convolutional layers have few sparse activation values, while the fully connected layers were much more sparse (Figures 3.2 and 3.4). Unlike the BatchNorm investigation, which showed sparsity in the convolutional layers, here representational sparsity was observed only in the fully connected layers and as a characteristic of the better generalising models. This was observed in more detail during the investigation of channel-specific activations (Figure 3.5), where the channels of the better generalising model were almost completely sparse.

Interesting observations were made about the representation characteristics when the representations were biclustered. Local structures emerged within the representations, indicating activation patterns for certain samples (Figure 4.4, 4.4, and A.2). Specifically in the convolutional layers, it is observed that the poorly generalising model activates *specifically*, with activation values being high or in the mid range but not completely off (sparse). However, for the better generalising model we observed that the activation values are more distributed between the activation value ranges. This changes when observing the fully connected layer where the better generalising model activates more *specifically* with high or sparse activation values, while the poorly generalising model has a distribution of activation values. These local structures are formed more clearly when there are more specific activations than distributed activation values (Table 4.4).

6.3.3 Representational key features

One of the interesting observations we made was that there are key features that are necessary for the model to make correct classifications. These were first noticed when biclustering the representations as local structures emerged (Section 4.3.3). These local structures consisted of samples with similar activation values and also of channels that are responsible for these values. These local structures mostly consist of the highest activation values or of sparse values. Investigating this from another angle reveals that some channels are active for specific classes, while not for others (Section 3.3.5). We also found that there are only a few active channels for specific classes in the fully connected layer of the better generalising models.

Extracting these key features was made possible by using an AE to compress the representations. Since we compared traditional representation summary metrics with the latent dimension of the AE, the latent dimension was the only representation in which the better generalising model always had purer clusters than its counterpart. These features captured in the latent dimension are correlated with the generalisation ability of the model (Section 5.4.4) indicating that these features are responsible for the generalisation ability of the models. Although these same key features are present in some of the other representation summary methods, they are not presented as key features indicated by the low purity scores observed when clustering PCA, the maximum of the channel, and the unaltered representation by class: A method which has been shown to be predictive of generalisation.

6.4 Evaluation

In this section, we compare our key findings with our research questions and evaluate if these objectives have been achieved.

1. **What different types of sample clusters (such as per node, per layer, per channel, or combinations of these) formed within the hidden representations of a network are potentially useful to investigate?** For this study, we set out to find useful sample clusters within the hidden representations to investigate. Using both class-based and class-agnostic clustering methods, we found that there are indeed clear clusters within the representations at each layer of the network. We also investigated whether the

biclustering of representations would uncover sample-feature clusters, which could prove useful to investigate. In this case, we found local structures within the representations that depict the relationships between the samples and the features. We also compared different representation summary methods with each other, such as the average and maximum of each channel, PCA, and the latent dimension of an AE trained on the representations of a network layer. Here, we found that the best low computational cost method for representation investigation is the average of each channel, although the overall best method is the AE as it not only reduces the dimensions but it also captures key features important to the classification process.

2. **What are appropriate ways in which to analyse clusters formed within the hidden representations?** By investigating class-based clustering, we can determine whether a model clusters according to its classes. We found that these clusters become remarkably pure in the deeper layers, signifying that hidden representations are inherently clustered by the network according to their class labels. Then again, by using a class-agnostic clustering method, we were able to track the optimal number of clusters formed for the representations. Here we found that models tend to have the same number of optimal clusters as classes in the deeper layers, suggesting that representations indeed tend to cluster according to class labels.
3. **What cluster characteristics show a difference in behaviour between models that generalise well and those that generalise poorly?** During the investigation of class-based clustering, we measured the purity of the representation clusters using the class label as the cluster label. The same can be done using the assigned cluster label during class-agnostic clustering as well as biclustering. These measures of purity proved to be useful cluster characteristics with which we can compare models with different generalisation gaps. We also found that these measures serve as indicators of generalisation for prediction purposes. In the case of class-based clusters, a higher purity indicates a better generalising model, while higher purity class-agnostic clusters and biclusters indicates a poorly generalising model. However, in the case of class-agnostic clusters of the AE latent representations, a higher purity indicates a better generalising model.

Another measurable cluster characteristic is the number of optimal class-agnostic clusters and biclusters. Although we did not find this metric useful, we did however, find that in some cases the number of optimal class-agnostic clusters for the better generalising model

tends to equal the number of classes in the dataset in earlier layers as opposed to the poorly generalising models.

6.5 Future work

In this section, we describe any future research that could result from this study.

- One of the aspects that we did not investigate was using a sparsity value larger than zero. We saw that the models tend to have very small activation values, and by considering these values as sparse, one could possibly get a more definitive answer on whether sparsity is beneficial in representations or if it is associated with poorly generalising models.
- This study consisted mainly of a few CNN architectures and only two datasets. This was mostly due to the computationally expensive nature of some of the experiments which limited our experimental extent. By investigating other models, architectures, and datasets, our hypothesis could be better investigated and our results could be confirmed or denied.
- One of the more interesting observations that we made was that representations have local structures where similar samples have similar features. We were able to extract these with the AE but due to the computationally expensive nature of this process another method could be useful. One such method is a biclustering algorithm, which does not rely on the structure of the representations but rather clusters samples/features together only based on similarity. This is an easy task for a human when presented with a simple representation and ordered to form biclusters. However, this task becomes increasingly difficult when defined regions need to be set on what defines a cluster. It is also more difficult if clusters need to be combined and broken up.
- Our study of AEs was very limited, since we only tested it on a few models of the same architecture, as well as not fully optimising them with extensive hyperparameter sweeps. These AEs proved useful not only in compressing the representations but also in capturing the key features in the representations. There are countless possibilities for further investigation, especially into why some models generalise better than others by investigating these key features.

6.6 Concluding remarks

Understanding why DNNs generalise and how to improve their generalisation ability remains an unanswered question. In this study, we attempted to understand the question by investigating these models from the inside. We found that not only are we able to cluster representations and that they become more class based throughout the layers, but that these clusters' characteristics can be predictive of generalisation.

Although this may turn out to not be significant in the grand scheme of things, we found great enjoyment in our work and hope that you did as well.

Bibliography

- [1] D. Adams, *The Ultimate Hitchhiker's Guide* (Arthur Dent series). Wings Books, 1996, ISBN: 9780517149256.
- [2] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning*. MIT Press, 2016.
- [3] A. Krizhevsky, I. Sutskever, and G. E. Hinton, “Imagenet classification with deep convolutional neural networks,” *Communications of the ACM*, vol. 60, no. 6, pp. 84–90, 2017.
- [4] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova, “Bert: Pre-training of deep bidirectional transformers for language understanding,” *arXiv preprint arXiv:1810.04805*, 2018.
- [5] G. Hinton, L. Deng, D. Yu, *et al.*, “Deep neural networks for acoustic modeling in speech recognition: The shared views of four research groups,” *IEEE Signal Processing Magazine*, vol. 29, no. 6, pp. 82–97, 2012.
- [6] A. Zhang, Z. C. Lipton, M. Li, and A. J. Smola, *Dive into Deep Learning*. Cambridge University Press, 2023.
- [7] S. J. Prince, *Understanding deep learning*. MIT press, 2023.
- [8] C. Zhang, S. Bengio, M. Hardt, B. Recht, and O. Vinyals, “Understanding deep learning requires rethinking generalization,” in *International Conference on Learning Representations*, 2017.
- [9] M. Belkin, D. Hsu, S. Ma, and S. Mandal, “Reconciling modern machine-learning practice and the classical bias–variance trade-off,” *Proceedings of the National Academy of Sciences*, vol. 116, pp. 15 849–15 854, 2019.
- [10] P. Nakkiran, G. Kaplun, Y. Bansal, T. Yang, B. Barak, and I. Sutskever, “Deep double descent: Where bigger models and more data hurt,” *Journal of Statistical Mechanics: Theory and Experiment*, vol. 2021, 2019.

-
- [11] P. Foret, A. Kleiner, H. Mobahi, and B. Neyshabur, “Sharpness-aware Minimization for Efficiently Improving Generalization,” *ArXiv*, vol. abs/2010.01412, 2020.
- [12] O. Bousquet and A. Elisseeff, “Algorithmic stability and generalization performance,” *Advances in Neural Information Processing Systems*, vol. 13, 2000.
- [13] C. Mouton, M. W. Theunissen, and M. H. Davel, “Input margins can predict generalization too,” *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 38, no. 13, 2024.
- [14] K. Kawaguchi, Z. Deng, X. Ji, and J. Huang, “How does information bottleneck help deep learning?” In *International Conference on Machine Learning*, PMLR, 2023, pp. 16 049–16 096.
- [15] R. Liao, A. Schwing, R. Zemel, and R. Urtasun, “Learning deep parsimonious representations,” *Advances in Neural Information Processing Systems*, vol. 29, 2016.
- [16] C.-Y. Chuang, Y. Mroueh, K. Greenewald, A. Torralba, and S. Jegelka, “Measuring generalization with optimal transport,” *Advances in Neural Information Processing Systems*, vol. 34, pp. 8294–8306, 2021.
- [17] Y. Jiang, D. Krishnan, H. Mobahi, and S. Bengio, “Predicting the generalization gap in deep networks with margin distributions,” in *7th International Conference on Learning Representations*, 2018.
- [18] P. Natekar and M. Sharma, “Representation based complexity measures for predicting generalization in deep learning,” *arXiv preprint arXiv:2012.02775*, 2020.
- [19] X. Glorot, A. Bordes, and Y. Bengio, “Deep sparse rectifier neural networks,” in *Proceedings of the fourteenth International Conference on Artificial Intelligence and Statistics*, vol. 15, PMLR, 2011, pp. 315–323.
- [20] M. J. Villani and N. Schoots, “Any deep ReLU network is shallow,” *arXiv preprint arXiv:2306.11827*, 2023.
- [21] 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.
- [22] S. Lundberg and S.-I. Lee, “A unified approach to interpreting model predictions,” *arXiv e-prints*, arXiv-1705, 2017.
-

-
- [23] V. Nair and G. Hinton, “Rectified linear units improve Restricted Boltzmann machines,” in *Proceedings of the 27th International Conference on International Conference on Machine Learning*, 2010, pp. 807–814.
- [24] F. Chollet, “Xception: Deep learning with depthwise separable convolutions,” in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2017, pp. 1251–1258.
- [25] M. Davel, T. Theunissen, A. Pretorius, and E. Barnard, “Dnns as layers of cooperating classifiers,” *Association for the Advancement of Artificial Intelligence*, p. 8, 2019.
- [26] D. G. Haasbroek and M. H. Davel, “Exploring neural network training dynamics through binary node activations,” *Southern African Conference for Artificial Intelligence Research*, 2020.
- [27] M. Caron, P. Bojanowski, A. Joulin, and M. Douze, “Deep clustering for unsupervised learning of visual features,” in *Proceedings of the European Conference on Computer Vision*, 2018, pp. 132–149.
- [28] Davel, Marelie H, *Paths in feedforward networks*, CAIR FAIR 2018, Nov. 2018.
- [29] D. L. Davies and D. W. Bouldin, “A cluster separation measure,” *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 1, no. 2, pp. 224–227, 1979.
- [30] X. Zhan, J. Xie, Z. Liu, Y.-S. Ong, and C. C. Loy, “Online deep clustering for unsupervised representation learning,” in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2020, pp. 6688–6697.
- [31] J. Li, P. Zhou, C. Xiong, and S. Hoi, “Prototypical contrastive learning of unsupervised representations,” in *9th International Conference on Learning Representations*, 2021, pp. 4–8.
- [32] Y. LeCun, C. Cortes, and C. Burges, “The MNIST database of handwritten digits,” <http://yann.lecun.com/exdb/mnist/>, vol. 2, 1998.
- [33] A. Krizhevsky and G. Hinton, “Learning multiple layers of features from tiny images,” *Citeseer*, 2009.
- [34] K. Simonyan and A. Zisserman, “Very deep convolutional networks for large-scale image recognition,” in *3rd International Conference on Learning Representations*, 2015.
- [35] A. Paszke, S. Gross, S. Chintala, *et al.*, “Automatic differentiation in PyTorch,” in *NIPS Autodiff Workshop*, 2017.
-

-
- [36] F. Pedregosa, G. Varoquaux, A. Gramfort, *et al.*, “Scikit-learn: Machine learning in Python,” *Journal of Machine Learning Research*, vol. 12, pp. 2825–2830, 2011.
- [37] H. L. Potgieter, C. Mouton, and M. H. Davel, “Impact of batch normalization on convolutional network representations,” Southern African Conference for Artificial Intelligence Research, 2024.
- [38] C. Mouton, R. Rabe, D. G. Haasbroek, M. W. Theunissen, H. L. Potgieter, and M. H. Davel, “Is network fragmentation a useful complexity measure?” In *NeurIPS 2024 Workshop on Scientific Methods for Understanding Deep Learning*, 2024.
- [39] C. Zhang, S. Bengio, M. Hardt, B. Recht, and O. Vinyals, “Understanding deep learning (still) requires rethinking generalization,” *Communications of the ACM*, vol. 64, no. 3, 2021.
- [40] D. P. Kingma and J. Ba, “Adam: A method for stochastic optimization,” in *3rd International Conference on Learning Representations*, 2015.
- [41] A. Defazio and K. Mishchenko, “Learning-rate-free learning by d-adaptation,” *The 40th International Conference on Machine Learning*, 2023.
- [42] J. Duchi, E. Hazan, and Y. Singer, “Adaptive subgradient methods for online learning and stochastic optimization.,” *Journal of Machine Learning Research*, vol. 12, no. 7, 2011.
- [43] D. Kingma and J. Ba, “Adam: A method for stochastic optimization,” *International Conference on Learning Representations*, Dec. 2014, arXiv preprint arXiv:1412.6980.
- [44] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning*. MIT Press, 2016, pp. 151–152.
- [45] K. Mishchenko and A. Defazio, “Prodigy: An expeditiously adaptive parameter-free learner,” *arXiv preprint arXiv:2306.06101*, 2023.
- [46] S. Ioffe and C. Szegedy, “Batch normalization: Accelerating deep network training by reducing internal covariate shift,” in *Proceedings of the 32nd International Conference on Machine Learning*, vol. 37, 2015.
- [47] Z. Li and S. Arora, “An exponential learning rate schedule for deep learning,” in *8th International Conference on Learning Representations*, 2019.
- [48] S. Santurkar, D. Tsipras, A. Ilyas, and A. Madry, “How does batch normalization help optimization?” *Advances in Neural Information Processing Systems*, vol. 31, 2018.
- [49] N. Bjorck, C. P. Gomes, B. Selman, and K. Q. Weinberger, “Understanding batch normalization,” in *Advances in Neural Information Processing Systems*, vol. 31, 2018.
-

-
- [50] P. Benz, C. Zhang, and I. S. Kweon, “Batch normalization increases adversarial vulnerability and decreases adversarial transferability: A non-robust feature perspective,” in *Proceedings of the IEEE/CVF International Conference on Computer Vision*, 2021.
- [51] R. Balestrierio and R. G. Baraniuk, “Batch normalization explained,” *arXiv preprint arXiv:2209.14778*, 2022.
- [52] H. Daneshmand, A. Joudaki, and F. Bach, “Batch normalization orthogonalizes representations in deep random networks,” *Advances in Neural Information Processing Systems*, vol. 34, 2021.
- [53] Y. Bengio, A. Courville, and P. Vincent, “Representation learning: A review and new perspectives,” *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 35, no. 8, 2013.
- [54] A. M. Pretorius, E. Barnard, and M. H. Davel, “Relu and sigmoidal activation functions,” in *Proceedings of South African Forum for Artificial Intelligence Research*, 2019.
- [55] K. Pearson, “LIII. on lines and planes of closest fit to systems of points in space,” *The London, Edinburgh, and Dublin Philosophical Magazine and Journal of Science*, vol. 2, no. 11, pp. 559–572, 1901.
- [56] D. A. Jackson, “Stopping rules in principal components analysis: A comparison of heuristical and statistical approaches,” *Ecology*, vol. 74, no. 8, pp. 2204–2214, 1993.
- [57] L. Guttman, “Some necessary conditions for common-factor analysis,” *Psychometrika*, vol. 19, no. 2, pp. 149–161, 1954.
- [58] H. F. Kaiser, “The application of electronic computers to factor analysis,” *Educational and Psychological Measurement*, vol. 20, no. 1, pp. 141–151, 1960.
- [59] H. F. Kaiser, “A note on guttman’s lower bound for the number of common factors,” *British Journal of Statistical Psychology*, 1961.
- [60] K. E. Zerba and J. P. Collins, “Spatial heterogeneity and individual variation in diet of an aquatic top predator,” *Ecology*, vol. 73, no. 1, pp. 268–279, 1992.
- [61] L. v. d. Maaten and G. Hinton, “Visualizing data using t-SNE,” *Journal of Machine Learning Research*, vol. 9, no. 11, pp. 2579–2605, 2008.
- [62] B. Kang, D. Garcia Garcia, J. Lijffijt, R. Santos-Rodríguez, and T. De Bie, “Conditional t-sne: More informative t-sne embeddings,” *Machine Learning*, vol. 110, pp. 2905–2940, 2021.

-
- [63] 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.
- [64] P. Baldi and K. Hornik, “Neural networks and principal component analysis: Learning from examples without local minima,” *Neural Networks*, vol. 2, no. 1, pp. 53–58, 1989.
- [65] S. Rifai, P. Vincent, X. Muller, X. Glorot, and Y. Bengio, “Contractive auto-encoders: Explicit invariance during feature extraction,” in *Proceedings of the 28th International Conference on International Conference on Machine Learning*, 2011, pp. 833–840.
- [66] N. Japkowicz, S. J. Hanson, and M. A. Gluck, “Nonlinear autoassociation is not equivalent to pca,” *Neural Computation*, vol. 12, no. 3, pp. 531–545, 2000.
- [67] J. MacQueen *et al.*, “Some methods for classification and analysis of multivariate observations,” in *Proceedings of the fifth Berkeley Symposium on Mathematical Statistics and Probability*, vol. 1, 1967, pp. 281–297.
- [68] F. Nielsen and F. Nielsen, “Hierarchical clustering,” *Introduction to HPC with MPI for Data Science*, pp. 195–211, 2016.
- [69] J. Shi and J. Malik, “Normalized cuts and image segmentation,” *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 22, no. 8, 2000.
- [70] D. Arthur and S. Vassilvitskii, “K-means++: The advantages of careful seeding,” in *Proceedings of the Annual ACM-SIAM Symposium on Discrete Algorithms*, vol. 8, 2007, pp. 1027–1035.
- [71] J. C. Dunn, “A fuzzy relative of the ISODATA process and its use in detecting compact well-separated clusters,” *Journal of Cybernetics*, vol. 3, no. 3, 1973.
- [72] A. K. Jain and R. C. Dubes, *Algorithms for clustering data*. Prentice-Hall, Inc., 1988.
- [73] Y. Jiang, P. Natekar, M. Sharma, *et al.*, “Methods and analysis of the first competition in predicting generalization of deep learning,” in *NeurIPS 2020 Competition and Demonstration Track*, ser. Proceedings of Machine Learning Research, vol. 133, 2021, pp. 170–190.
- [74] J. A. Hartigan, “Direct clustering of a data matrix,” *Journal of the American Statistical Association*, vol. 67, no. 337, pp. 123–129, 1972.
- [75] Y. Cheng and G. M. Church, “Biclustering of expression data,” *Proceedings. International Conference on Intelligent Systems for Molecular Biology*, vol. 8, pp. 93–103, 2000.
-

-
- [76] Y. Kluger, R. Basri, J. T. Chang, and M. Gerstein, “Spectral Biclustering of Microarray Data: Cocustering Genes and Conditions,” *Genome Research*, vol. 13, no. 4, pp. 703–716, 2003.
- [77] S. C. Madeira and A. L. Oliveira, “Biclustering algorithms for biological data analysis: A survey,” *IEEE/ACM Transactions on Computational Biology and Bioinformatics*, vol. 1, no. 1, pp. 24–45, 2004.
- [78] J. S. Aguilar-Ruiz, “Shifting and scaling patterns from gene expression data,” *Bioinformatics*, vol. 21, no. 20, pp. 3840–3845, 2005.
- [79] B. Pontes, R. Gírldez, and J. S. Aguilar-Ruiz, “Quality Measures for Gene Expression Biclusters,” *PLoS ONE*, vol. 10, no. 3, e0115497, 2015.
- [80] A. Mukhopadhyay, U. Maulik, and S. Bandyopadhyay, “A novel coherence measure for discovering scaling biclusters from gene expression data,” *Journal of Bioinformatics and Computational Biology*, vol. 7, no. 5, pp. 853–868, 2009.
- [81] F. Divina, B. Pontes, R. Gírldez, and J. S. Aguilar-Ruiz, “An effective measure for assessing the quality of biclusters,” *Computers in Biology and Medicine*, vol. 42, no. 2, pp. 245–256, 2012.
- [82] B. Pontes, R. Gírldez, and J. S. Aguilar-Ruiz, “Measuring the Quality of Shifting and Scaling Patterns in Biclusters,” in *Pattern Recognition in Bioinformatics*, Berlin, Heidelberg: Springer, 2010, pp. 242–252.
- [83] Y. Jiang, P. Foret, S. Yak, *et al.*, “NeurIPS 2020 competition: Predicting generalization in deep learning,” *arXiv preprint arXiv:2012.07976*, 2020.
- [84] Y. LeCun, L. Jackel, L. Bottou, *et al.*, “Comparison of learning algorithms for handwritten digit recognition,” in *International Conference on Artificial Neural Networks*, vol. 60, 1995, pp. 53–60.
- [85] M. Lin, Q. Chen, and S. Yan, “Network in network,” in *International Conference on Learning Representations*, 2014, pp. 14–16.
- [86] A. Krizhevsky, V. Nair, and G. Hinton, *CIFAR10 dataset*, <https://www.cs.toronto.edu/~kriz/cifar.html>.
- [87] Y. Netzer, T. Wang, A. Coates, A. Bissacco, B. Wu, and A. Y. Ng, “Reading digits in natural images with unsupervised feature learning,” in *NIPS Workshop on Deep Learning and Unsupervised Feature Learning*, 2011.
-

-
- [88] L. N. Darlow, E. J. Crowley, A. Antoniou, and A. J. Storkey, “CINIC-10 is not ImageNet or CIFAR-10,” *CoRR*, vol. abs/1810.03505, 2018.
 - [89] M.-E. Nilsback and A. Zisserman, “Automated flower classification over a large number of classes,” in *Sixth Indian Conference on Computer Vision, Graphics, Image Processing*, 2008, pp. 722–729.
 - [90] O. M. Parkhi, A. Vedaldi, A. Zisserman, and C. V. Jawahar, “Cats and dogs,” in *IEEE Conference on Computer Vision and Pattern Recognition*, 2012, pp. 3498–3505.
 - [91] H. Xiao, K. Rasul, and R. Vollgraf, “Fashion-MNIST: a novel image dataset for benchmarking machine learning algorithms,” *arXiv preprint arXiv:1708.07747*, 2017.
 - [92] C. G. Northcutt, A. Athalye, and J. Mueller, “Pervasive label errors in test sets destabilize machine learning benchmarks,” in *Conference on Neural Information Processing Systems Datasets and Benchmarks Track*, 2021.

Appendix A

Supplemental content

“‘That’s right!’ shouted Vroomfondel, ‘we demand rigidly defined areas of doubt and uncertainty!’”

- Douglas Adams, The Ultimate Hitchhiker’s Guide to the Galaxy Page 114 [1]

A.1 Appendix: Chapter 3

This section provides supplementary material for Chapter 3.

A.1.1 Low-dimensional perspective

Here we present the low-dimensional representations of GL and BL for `Conv_6`, `Lin_1`, and `Lin_2`. These representations were reduced dimensionally, as described in Section 3.3.4 to only two dimensions.

We observe similar results as in Section 3.3.4 where GL has more separated classes, especially in the fully connected layers than BL. We also observed that although these models are deeper, they are not able to form clearly separated clusters in the convolutional layers.

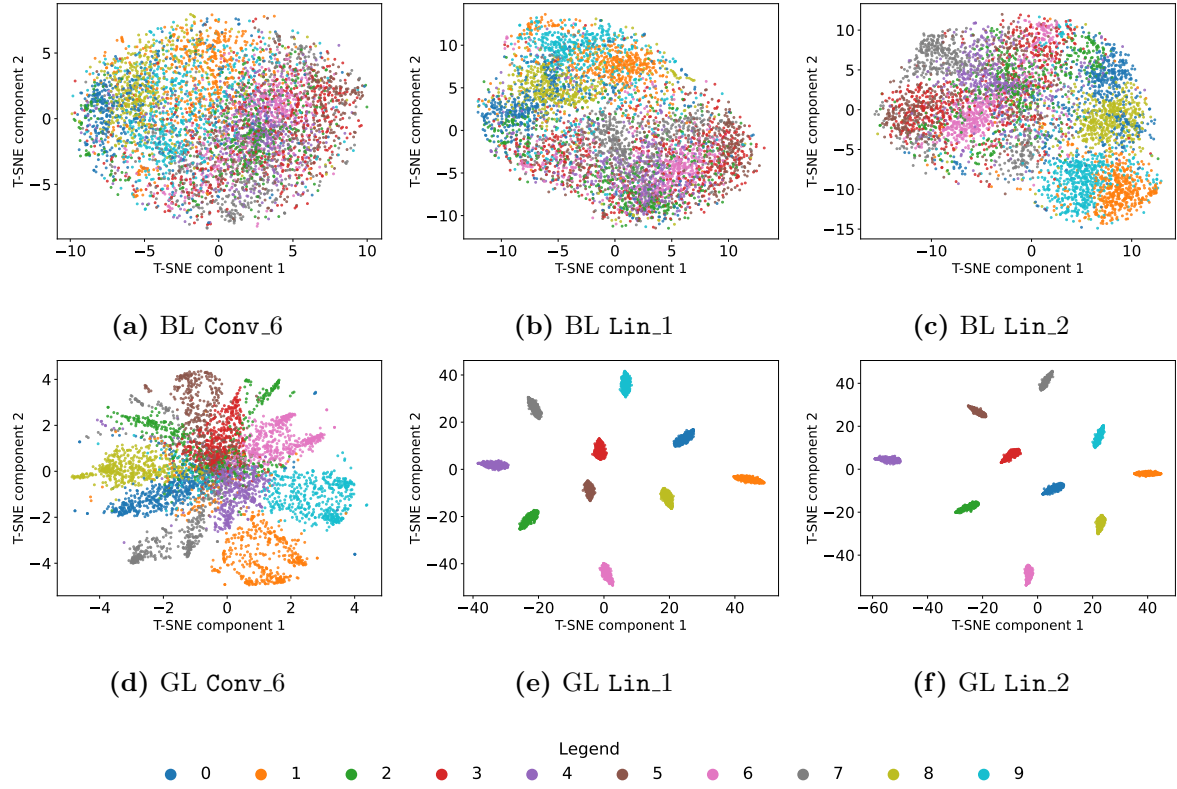


Figure A.1: Lower dimensional views of BL (a, b and c) and GL (d, e and f) hidden representations of each layer reduced by t-SNE to two dimensions. Colours indicate the true class label of each sample point.

A.2 Appendix: Chapter 4

This section provides supplementary material for Chapter 4.

A.2.1 Results: Bicluster purity

In this section, we present the results of the bicluster grid sweep for both GL and BL. The experimental setup is discussed in Section 4.3.1

A.2.2 Visualising best performing clusters

In this section, we present the best performing bicluster combinations of GS and BS Lin_1 for both SMSR and VEt. The experiment is performed according to Section 4.3.1.

Table A.1: Summary of the average and best VEt and SMSR from the bicluster combination grid search for GL and BL.

Layer	VEt average	VEt best	SMSR average	SMSR best
BL Conv_1	0.3509	0.2233	0.0302	0.0175
GL Conv_1	0.2751	0.2013	0.0263	0.0087
BL Conv_2	0.3198	0.1660	0.0237	0.0123
GL Conv_2	0.4059	0.3072	0.0488	0.0207
BL Conv_3	0.3452	0.1620	0.0200	0.0115
GL Conv_3	0.5326	0.4290	0.0691	0.0385
BL Conv_4	0.3926	0.2106	0.0295	0.0153
GL Conv_4	0.6911	0.5672	0.1898	0.1372
BL Conv_5	0.4871	0.3047	0.0790	0.0448
GL Conv_5	0.7571	0.6552	0.3920	0.3223
BL Conv_6	0.6605	0.4889	0.1047	0.0824
GL Conv_6	0.7909	0.6606	0.9919	0.7820
BL Lin_1	0.7073	0.5635	0.3992	0.2343
GL Lin_1	0.5661	0.2489	0.6955	0.3751
BL Lin_2	0.6761	0.4690	0.3241	0.1735
GL Lin_2	0.2940	0.1173	0.0988	0.0182

Here we observe that GS has more separable biclusters than BS, which is the opposite that we observed in Section 4.3.3. It is also interesting to note that both VEt and SMSR had low scores for similar bicluster combinations since both have 2 channel clusters and 18 and 20 respective sample clusters, while GS has the same number of sample clusters for both methods.

A.2.3 Investigating QM scores at different bicluster combinations

In this section, we provide the QM scores of the different layers as described in Section 4.4.1.

The results are discussed in Section 4.4.1.

A.2.4 Predicting generalisation

In this section, we present the correlation plots between the purity of the bicluster and the model evaluation accuracy as described in Section 4.4.3. We present the strongest correlation for each layer for both VEt and SMSR.

The results are discussed in Section 4.4.3

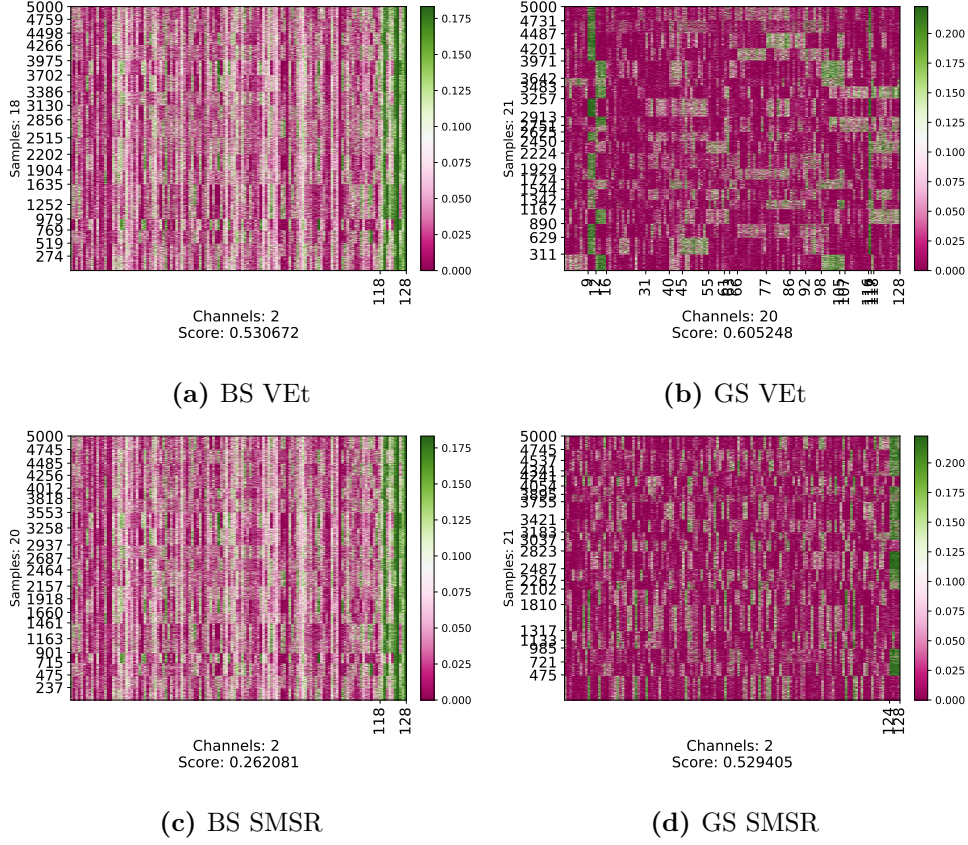


Figure A.2: Lin_1 Representations sorted to the cluster labels generated by the best performing bicluster combinations according to VEt and SMSR for both BS and GS.

We also present the significance values of the Kendall rank (Table A.2) of the PGDL Task 1 small models along with the Kendall rank (Table A.3) and the significance values (Table A.4) for the larger models.

A.3 Appendix: Chapter 5

This section provides supplementary material for Chapter 5.

A.3.1 Class-agnostic clustering results

In this section, we present the number of optimal clusters formed for each of the TRSMs as described in Section 5.3.

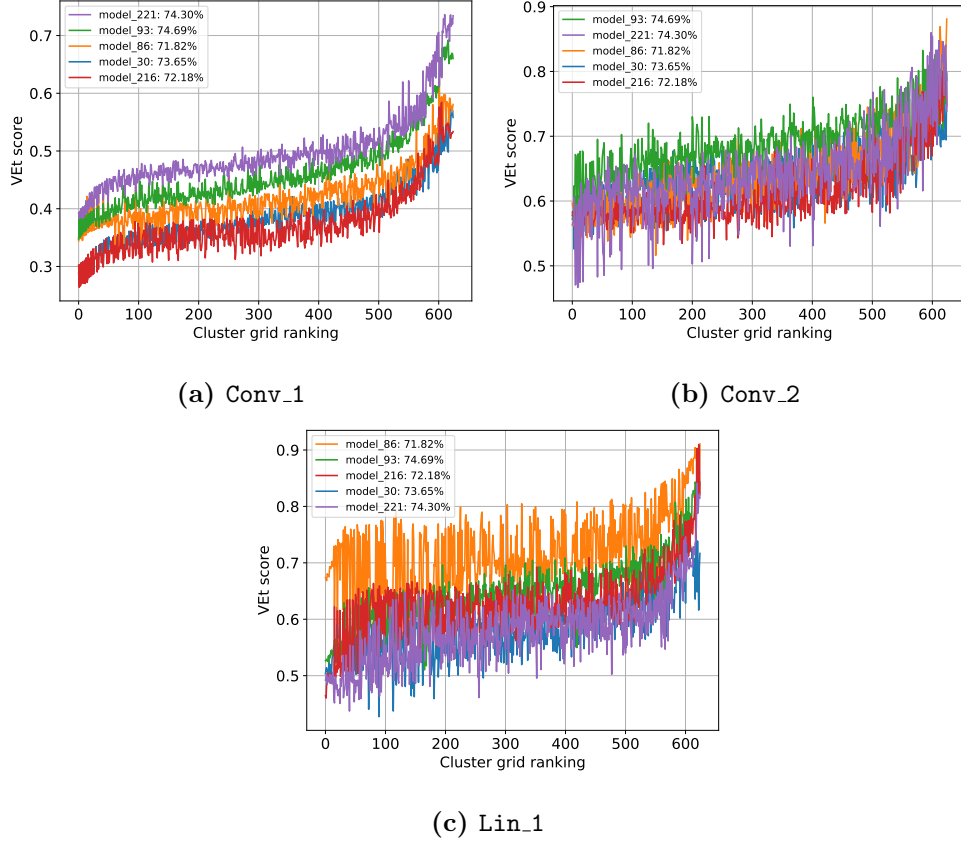


Figure A.3: Average VET for each bicluster combination, for each of the five randomly selected models. The scores are sorted by the cluster index of the mean score over all five models (low to high) for each bicluster combination. Each colour represents a different model.

We find that most of the methods only have two optimal clusters, while the average and max channel representations has a variation of different numbers of clusters which tend to the number of classes in the dataset. Since we did not explore this any further we cannot provide any concrete conclusions.

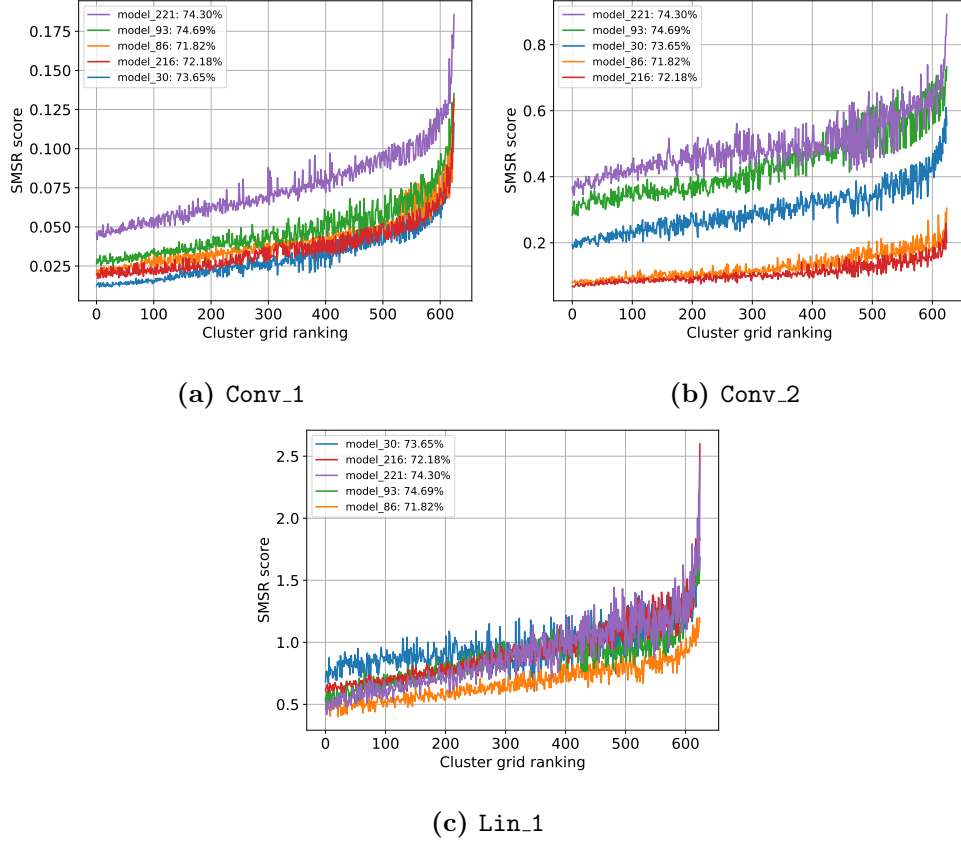


Figure A.4: Average SMSR for each bicluster combination of the three layers, for each of the five randomly selected models. The scores are sorted by the cluster index of the mean score over all five models (low to high) for each bicluster combination. Each colour represents a different model.

Table A.2: P-values associated with the Kendall's rank correlation using different QM scores for PGDL Task 1 small models. All results are significant at $p < 5e-4$, except the bolded results that are only significant at $p < 1e-4$.

	Vet			SMSR		
	Minimum	Average	Diagonal	Minimum	Average	Diagonal
Conv_1	0.2099e-3	8.45069e-6	7.3748e-7	4.8588e-5	4.6525e-7	4.5825e-8
Conv_2	1.8110e-6	1.4931e-8	3.0611e-8	3.8554e-12	4.4935e-11	1.0307e-10
Lin_1	1.9058e-5	2.4100e-7	2.4100e-7	7.0728e-3	1.3816e-5	3.0597e-6
Lin_2	6.5036e-5	0.1357e-3	6.5036e-5	0.2164e-3	0.2714e-3	1.1772e-3

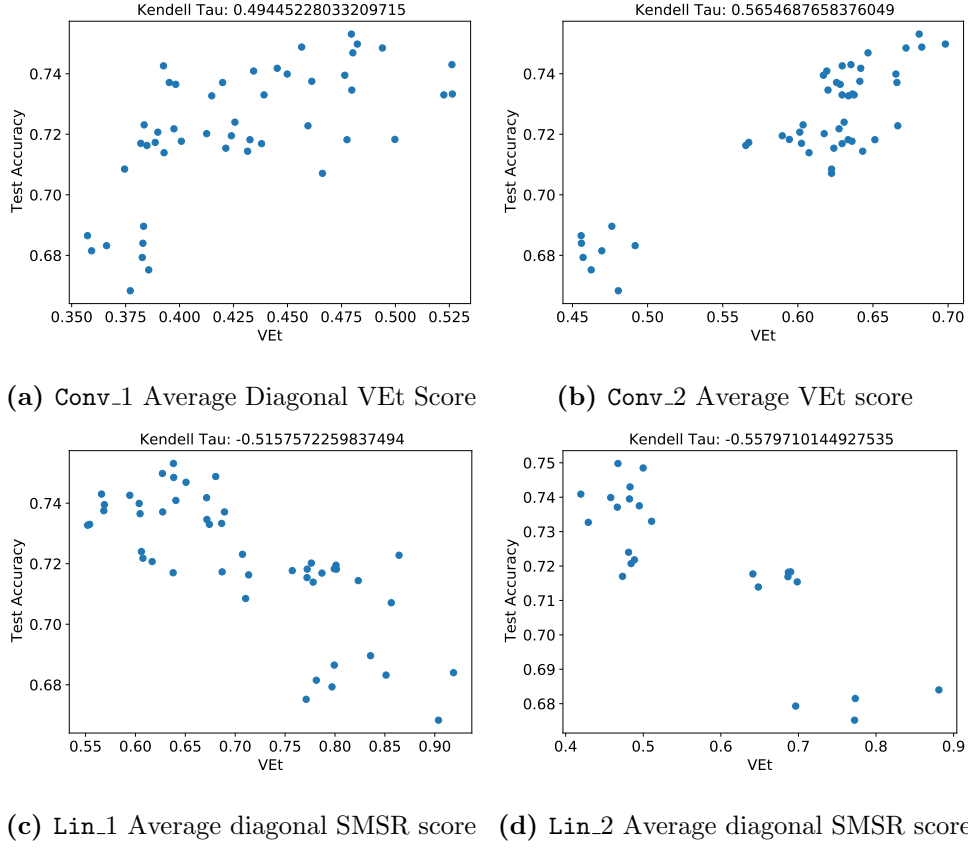


Figure A.5: Highest correlation VEt for each of the layers, measured with each of the proposed methods plotted against the evaluation accuracy of each model. We also indicate the Kendall's rank of each plot.

Table A.3: Kendall's rank coefficient between the evaluation accuracy and various QM score for PGDL Task 1 large models

	VEt			SMSR		
	Minimum	Average	Diagonal	Minimum	Average	Diagonal
Conv_1	-0.0266	-0.0053	0.0106	-0.0319	0.2092	0.2642
Conv_2	0.4876	0.4397	0.4770	0.3954	0.5567	0.5603
Conv_3	0.6206	0.4876	0.5426	0.4770	0.4787	0.5035
Conv_4	0.6418	0.4840	0.5496	0.6259	0.6472	0.6348
Conv_5	0.7411	0.7943	0.8050	0.7677	0.7748	0.7748
Conv_6	0.3599	0.4982	0.5195	0.7943	0.7979	0.8032
Lin_1	-0.5816	-0.5408	-0.5762	-0.2766	-0.1383	0.0213
Lin_2	-0.5797	-0.6667	-0.6594	-0.5797	-0.5290	-0.3986

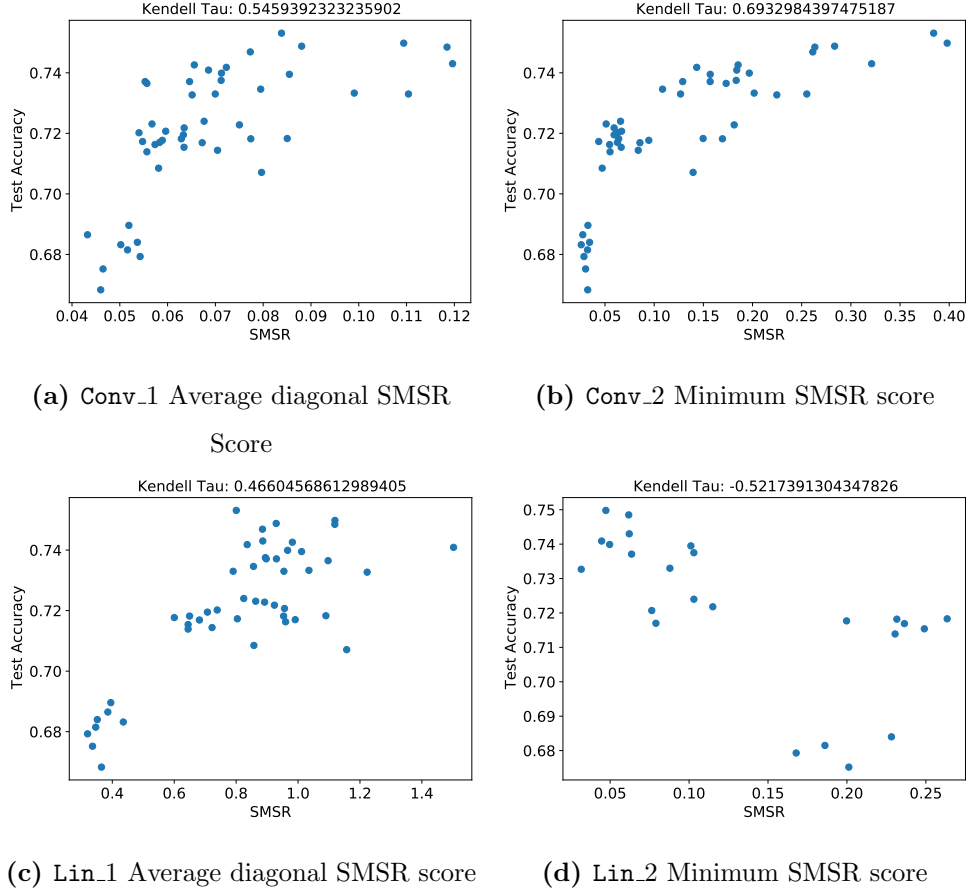


Figure A.6: Highest correlation SMSR for each of the layers, measured with each of the proposed methods plotted against the evaluation accuracy of each model. We also indicate the Kendall's rank of each plot.

Table A.4: P-values associated with the Kendall's rank correlation using different QM scores for PGDL Task 1 large models. All results are significant at $p < 1$, except the bolded results that are only significant at $p < 1e-2$

	Vet			SMSR		
	Minimum	Average	Diagonal	Minimum	Average	Diagonal
Conv_1	0.7897	0.9575	0.9151	0.7490	0.0359	0.0081
Conv_2	1.0165e-6	1.0410e-5	1.7376e-6	7.3684e-5	2.3819e-8	1.9403e-8
Conv_3	4.9202e-10	1.0165e-6	5.3436e-8	1.7376e-6	1.5903e-6	4.4551e-7
Conv_4	1.2352e-10	1.2169e-6	3.5764e-8	3.4973e-10	8.6828e-11	1.9678e-10
Conv_5	1.0827e-13	1.6699e-15	7.0102e-16	1.3925e-14	7.9642e-15	7.9642e-15
Conv_6	0.3079e-3	5.8812e-7	1.9048e-7	1.6699e-15	1.2519e-15	8.1077e-16
Lin_1	5.5243e-9	5.9033e-8	7.5942e-9	5.6530e-3	0.1656	0.8311
Lin_2	2.9784e-5	7.6721e-7	1.0804e-6	2.9784e-5	0.1717e-3	5.9150e-3

Table A.5: GS and BS optimal number of clusters for class-agnostic clustering. TRSM and the latent dimension of the AE trained on the representation are shown. Representations are clustered with k-means between 2 and 25 optimal clusters with the optimal cluster being selected by the lowest DBI score

	Raw	Maximum	Average	PCA	AE Latent Dim
BS Conv_1	2	2	2	2	3
GS Conv_1	2	8	7	2	2
BS Conv_2	2	6	5	2	2
GS Conv_2	2	2	8	3	2