



# Clustering and Topological Data Analysis of Single-Cell RNA Sequencing Data

COM3001 Final Year Project Report  
2020/2021

University of Surrey

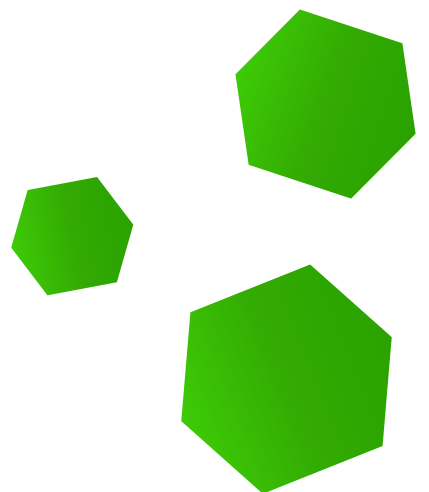
Faculty of Engineering and Physical Sciences

Department of Computing

Student: Tom Wilson

URN: 6515762

Supervisor: Tom Thorne



## Abstract

Computational biology is the use of computational methods to interpret biological systems and data. Biological experiments produce large amounts of data that would be hard to analyse by hand in a lab and so algorithms and models can be applied to work out previously unresolved problems.

This report looks at the application of dimensionality reduction techniques, clustering methods and topological data analysis commonly used in artificial intelligence to interpret one simulated and two real datasets produced by single-cell RNA-sequencing experiments. The goal being to reveal hidden variation in gene expression from cells of the same type.

The first part of the study focuses on creating an optimised autoencoder for the first dataset `sc_10x`. An autoencoder is then trained for each dataset and the suitability of applying encoding, standardization and PCA before clustering is compared and evaluated against a known ground truth. Six different traditional clustering methods are applied with focus on k-means and agglomerative hierarchical clustering. In addition, dimensionality reduction techniques ICA and NMF, t-SNE and spectral biclustering are also tested, though not as extensively. In the second part, the Mapper algorithm is utilized to view the high-dimensional space of each dataset.

Overall, this study has been highly successful at clustering genetic differences between cell lines with two out of three datasets achieving 99.9% and 100% accuracy respectively. It was discovered that these state-of-the-art results can be achieved through the correct combination of pre-processing and optimisation of the number of features to cluster through dimensionality reduction. This report could therefore be used as a guide of good practices when working with scRNA-seq data.

Code written in Python and R across two Jupyter notebooks and is freely available from:  
<https://github.com/TomMakesThings/Clustering-and-TDA-of-scRNA-seq-Data>.

## Acknowledgements

I would like to thank Luyi Tian for providing the open-source dataset [sc\\_10x](#), Luke Zappia, Belinda Phipson and Alicia Oshlack for making the gene count simulation tool [Splatter](#), Zeisel et al. for their dataset of [mouse cortex RNA counts](#) and van Veen et al. for developing [Kepler Mapper](#).

Finally, I would like to thank Stella Kazamia for her feedback, and my project sponsor, Dr Tom Thorne, for his continued support and guidance throughout the project.

# Table of Contents

Abstract .....	1
Acknowledgements.....	1
Table of Figures.....	5
Table of Tables.....	7
Table of Equations.....	8
Terms, Abbreviations and Notation.....	9
Terms .....	9
Acronyms and Abbreviations.....	9
Mathematical Notation .....	10
Statement of Ethics.....	11
Ethics Review .....	11
1. Introduction.....	12
1.1 Problem Background .....	12
1.2 Project Description .....	12
1.3 Aims and Objectives .....	12
1.4 Success Criteria.....	12
1.5 Report Structure.....	13
2. Literature Review .....	14
2.1 Single-Cell RNA Sequencing .....	14
2.1.1 Purpose .....	14
2.1.2 How It Works.....	14
2.1.3 Benefit Over Alternative Techniques .....	14
2.1.4 Challenges.....	15
2.2 Data Encoding Techniques.....	15
2.2.1 Principal Component Analysis (PCA).....	15
2.2.2 Autoencoders.....	15
2.1.2 PCA vs Autoencoders.....	16
2.1.3 Conclusion .....	16
2.2 Neural Network Optimisation .....	16
2.2.1 Optimisers and Learning Rate .....	17
2.2.2 Batch Size .....	17
2.2.3 Epochs.....	18
2.2.4 Conclusion .....	18
2.3 Clustering Techniques.....	18
2.3.1 K-Means Clustering .....	19
2.3.2 Hierarchical Clustering.....	19
2.3.3 K-Means vs Hierarchical Clustering .....	19
2.3.4 Conclusion .....	20
2.4 Topological Data Analysis.....	20
2.4.1 Purpose .....	20
2.4.2 Topological Signatures .....	20
2.4.3 The Vietoris-Rips Complex.....	20
2.4.4 Persistent Homology .....	21
2.5 Datasets .....	21
2.5.1 Human Lung Adenocarcinoma Benchmark Dataset.....	21

2.5.2	Splat Simulated Evaluation Dataset.....	21
2.5.3	Mouse Cortex Evaluation Dataset.....	21
2.6	Machine Learning Libraries .....	22
2.6.1	PyTorch.....	22
2.6.2	TensorFlow with Keras .....	22
2.6.3	PyTorch vs TensorFlow .....	22
2.6.4	Conclusion .....	22
3.	Requirements and Specification.....	23
3.1	Requirements .....	23
3.1.1	Sub-Objectives and Prioritisation .....	23
3.2	Hardware and Software Specification .....	24
3.2.1	Hardware Specification.....	24
3.2.2	Software Specification .....	24
3.3	Feasibility Analysis.....	24
3.3.1	Technical Feasibility.....	24
3.3.2	Legal Feasibility .....	25
3.3.3	Economic Feasibility .....	25
3.3.4	Scheduling Feasibility .....	25
3.4	Planning.....	25
3.4.1	Workplan .....	25
3.4.2	Risk Assessment .....	26
4.	System and Experiment Design.....	28
4.1	System Design .....	28
4.1.1	Autoencoder.....	28
4.1.2	Clustering.....	31
4.1.3	Topological Data Analysis .....	33
4.1.4	Creating Simulated Data.....	33
4.1	Experiment Design.....	34
5.	Implementation.....	36
5.1	Setting up the Datasets.....	36
5.1.1	Opening the Data.....	36
5.1.2	Splitting Data into Batches.....	36
5.2	Autoencoder.....	36
5.2.1	LitAutoencoder Class .....	36
5.2.2	K-Fold Cross Validation .....	37
5.2.3	Autoencoder Training.....	37
5.2.4	Saving Model State to File.....	37
5.2.5	Autoencoder Testing.....	38
5.3	Clustering.....	38
5.3.1	Set up Data and Labels.....	38
5.3.2	K-Means.....	38
5.3.3	Agglomerative Hierarchical Clustering .....	40
5.3.4	Alternative Clustering Algorithms.....	40
5.4	Topological Data Analysis.....	40
5.4.1	Kepler Mapper .....	40
5.5	Creating Simulated Data .....	41
5.5.1	Opening the Benchmarking Dataset.....	41



5.5.2	Creating a New Dataset.....	41
6.	Testing and Validation.....	43
6.1	Datasets .....	43
6.1.1	Benchmarking Data .....	43
6.1.2	Simulated Data .....	44
6.1.3	Mouse Cortex Evaluation Data .....	45
6.2	Autoencoder Optimisation.....	45
6.2.1	Testing the Initial Model.....	45
6.2.2	Learning Rate .....	46
6.2.3	Batch Size .....	47
6.2.4	Optimiser Selection .....	48
6.2.5	Hidden Layers.....	49
6.2.6	Activation Function.....	49
6.2.7	Number of Epochs .....	51
6.2.8	Summary of Autoencoder Optimisation.....	51
6.3	Training the Optimised Network .....	51
6.3.1	K-Fold Cross Validation .....	51
6.3.2	Training, Validation and Testing Results.....	52
6.4	Clustering.....	54
6.4.1	Ground Truth of Data.....	54
6.4.2	K-Means.....	56
6.4.3	Agglomerative Hierarchical Clustering .....	65
6.4.4	Visualising PCA Gene Expression .....	68
6.4.5	Alternative Algorithms .....	71
6.4.6	Biclustering.....	76
6.5	Topological Data Analysis.....	78
6.5.1	Benchmarking Data .....	78
6.5.2	Simulated Data .....	78
6.5.3	Mouse Cortex Data.....	78
6.5.4	Comparing TDA to Clustering .....	78
7.	Conclusions and Future Work .....	79
7.1	Overview.....	79
7.2	Evaluation Against Objectives .....	79
7.3	Results and Recommendations.....	80
7.4	Future Work.....	81
7.5	Final Statement.....	81
	References .....	82
	Appendix .....	87
	Appendix A - Ethics Review .....	87
	Sage-HDR Form.....	87
	Sage-AR Form .....	89
	Appendix B - Experiment Results.....	91
	Testing the Initial Model .....	91
	Activation Function.....	91
	Training the Optimised Network.....	92
	Clustering .....	93

## Table of Figures

Figure 1: RNA sequencing steps [97] .....	14
Figure 2: Single-cell vs bulk RNA sequencing [11] .....	14
Figure 3: Example of PCA example showing two principal components [16] .....	15
Figure 4: Standard deterministic autoencoder architecture (left) and variational autoencoder architecture (right) [19] .....	16
Figure 5: Example showing validation accuracy over time for different optimisers [23] .....	17
Figure 6: Example of learning rate optimisation using LR Range Test [26] .....	17
Figure 7: Learning rate comparison [24] .....	17
Figure 8: Trade-off between training and testing accuracy [32] .....	18
Figure 9: Comparison between underfitting and overfitting [101] .....	18
Figure 10: K-means clustering example [102] .....	19
Figure 11: Example of agglomerative hierarchical clustering dendrogram [33] .....	19
Figure 12: Low-dimensional simplices [105] .....	20
Figure 13: Example in which a simplicial complex is generated by increasing $\epsilon$ [43] .....	20
Figure 14: Project timetable .....	26
Figure 15: Flowchart for training and testing autoencoder .....	28
Figure 16: Autoencoder architecture diagram .....	29
Figure 17: Non-linear activation functions [107] .....	30
Figure 18: 5-fold cross validation [108] .....	30
Figure 19: Flowchart of clustering steps .....	31
Figure 20: Elbow method (left) and silhouette coefficient (right) [36] .....	32
Figure 21: Bipartite graph matching labels to predicted clusters .....	32
Figure 22: Mapper applied to 2D data [109] .....	33
Figure 23: Top left single, top right groups, lower left paths and lower right batches [72] .....	34
Figure 24: DataFrame showing 150 test samples from the benchmarking dataset .....	36
Figure 25: Checkpoint (CKPT) and test, train and validation DataLoader (PTH) files for the benchmarking autoencoder .....	37
Figure 26: Reduced size checkpoint (CKPT) and test, train and validation tex files for the benchmarking autoencoder .....	38
Figure 27: Print screen of assigning numerical values to the cell lines labels of the benchmarking dataset .....	38
Figure 28: Example graph showing cell lines of the benchmark dataset and its accuracy, ARI and silhouette coefficient .....	39
Figure 29: Example showing the effect of increasing principal components on ARI for the encoded benchmark dataset .....	39
Figure 30: Example graphs revealing the gain in variation with the addition of each principal component for the encoded benchmarking data .....	40
Figure 31: Example of testing ICA with different numbers of components on the benchmarking data .....	40
Figure 32: (Clickable) Testing epsilon values 0.05, 0.2, 0.4 and 0.8 for the benchmarking dataset .....	41
Figure 33: (Clickable) Testing 2, 5, 25 and 50 cubes for the benchmarking dataset .....	41
Figure 34: Single group of simulated cells (top) and four groups of simulated cells (lower) .....	41
Figure 35: Comparing distribution of mean gene expression between Splat and real data .....	42
Figure 36: Print screen of sc_10x .....	43
Figure 37: Print screen of benchmarking data cell lines .....	43
Figure 38: ARI for a range of clustering techniques [79] .....	44
Figure 39: Print screen of benchmarking ARI .....	44
Figure 40: Simulated data group labels .....	44
Figure 41: Print screen of simulated data .....	44
Figure 42: evaluation data group labels .....	45
Figure 43: Print screen of evaluation data .....	45
Figure 44: Loss (MSE) across 40 epochs for first run .....	45
Figure 45: Loss (MSE) across 40 epochs after shuffling .....	46
Figure 46: Loss (MSE) after 40 epochs for 5 different learning rates .....	46
Figure 47: Learning rate finder results (left) and MSE over epochs for 0.000794 learning rate (right) .....	46
Figure 48: Training loss (MSE) over 40 epochs for 10 batch sizes .....	47
Figure 49: Testing and training loss (MSE) after 40 epochs for 10 batch sizes .....	47
Figure 50: Testing and training loss (MSE) after 40 epochs for five optimisers .....	48
Figure 51: Training loss (MSE) over 40 epochs for five different optimisers .....	48
Figure 52: Training and validation loss (MSE) across 100 epochs for Adam and AdamW .....	48
Figure 53: Training loss (MSE) over 40 epochs for 1, 3 and 5 hidden layers .....	49
Figure 54: Training and validation loss (MSE) over 40 epochs for 5 hidden layers (left) and 3 hidden layers (right) .....	49
Figure 55: Testing and training loss (MSE) after 40 epochs for hidden layer activation functions .....	50
Figure 56: Training loss (MSE) after 40 epochs for combinations of activation functions .....	50
Figure 57: Training and validation loss (MSE) after 400 epochs .....	51
Figure 58: Benchmarking data autoencoder's training and validation loss over 250 epochs .....	52
Figure 59: Simulated data autoencoder's training and validation loss over 250 epochs .....	52
Figure 60: Mouse cortex data autoencoder's training and validation loss over 250 epochs .....	53
Figure 61: Average loss (MSE) per mini-batch during testing of benchmarking data autoencoder (top left), simulated (top right) and mouse cortex (lower left) .....	53
Figure 62: (Clickable) 2D plot of benchmark data's cell lines .....	54
Figure 63: (Clickable) 3D plot of benchmark testing data's cell lines from three angles .....	54
Figure 64: 2D plot of simulated data's expected groups .....	55

Figure 65: 3D plot of simulated test data's groups from three angles .....	55
Figure 66: (Clickable) 2D plot of mouse cortex data's expected groups.....	55
Figure 67: (Clickable) 3D plot of mouse cortex test data's expected groups from three angles .....	55
Figure 68: Elbow method (left) and silhouette coefficient (right) for benchmarking data.....	56
Figure 69: (Clickable) K-means prediction (top left), actual clusters (top right) for 2 PC encoded benchmarking data .....	56
Figure 70: (Clickable) K-means prediction (top left), actual clusters (top right) and difference (bottom) for 2 PC encoded benchmarking data .....	57
Figure 71: (Clickable) K-means prediction (top left), actual clusters (top right) and difference (bottom) for 2 PC pre-standardized encoded benchmarking data.....	57
Figure 72: (Clickable) K-means prediction (left), actual clusters (middle) and difference (right) for 3 PC encoded benchmarking data with standardization.....	58
Figure 73: (Clickable) K-means prediction (left), actual clusters (middle) and difference (right) for 3 PC encoded benchmarking data without standardization .....	58
Figure 74: K-means accuracy for unstandardized encoded data when testing different numbers of principal components (left) and a plot showing the incorrectly identified cell for 4 PCs (right) (Clickable).....	58
Figure 75: K-means accuracy for standardized encoded data when testing different numbers of principal components (left) and a 2D plot showing the incorrectly identified cell for 7 PCs (right) (Clickable).....	59
Figure 76: (Clickable) K-means using 2 PCs from the raw unencoded data (left) and standardized unencoded data (right).....	59
Figure 77: (Clickable) Incorrectly identified cells from k-means using 2 PCs for the raw unencoded data (left) and standardized unencoded data (right) .....	60
Figure 78: Comparing accuracy to number of principal components for k-means on the raw unencoded data (left) and standardized unencoded data (right) .....	60
Figure 79: (Clickable) K-means prediction (left), actual clusters (middle) and difference (right) for 3 PC original benchmarking data.....	60
Figure 80: (Clickable) K-means prediction (left) and actual clusters (right) for 3 PC original benchmarking data with standardization.....	60
Figure 81: Elbow method (left) and silhouette coefficient (right) for simulated data.....	61
Figure 82: (Clickable) 2 PC k-means on the encoded simulated data without standardization .....	61
Figure 83: (Clickable) 2 PC k-means on the unencoded simulated data without standardization.....	62
Figure 84: (Clickable) Predicted clusters (left) and incorrect cells (right) for k-means using 2 PCs on the encoded simulated data.....	62
Figure 85: (Clickable) Predicted clusters (left) and expected groups (right) for k-means using 2 PCs on the unencoded simulated data .....	62
Figure 86: (Clickable) 3D plot of predicted clusters (left) expected groups (right) for k-means using 6 PCs on encoded simulated data .....	63
Figure 87: Elbow method (left) and silhouette coefficient (right) for mouse cortex data.....	63
Figure 88: (Clickable) 2 PC k-means on the unencoded mouse cortex data without standardization (left) and BackSPIN labels (right).....	64
Figure 89: (Clickable) 2 PC k-means on the unencoded mouse cortex data after standardization .....	64
Figure 90: Finding the best number of principal components (left), and k-means predictions on the encoded mouse cortex data with standardization for 15 PCs (Clickable) .....	64
Figure 91: (Clickable) K-means predictions on the standardized, unencoded mouse cortex data with 16 principal components and t-SNE with perplexity 30.....	65
Figure 92: (Clickable) K-means on the standardized, unencoded mouse cortex data with 16 principal components and t-SNE perplexity 5, 10, 20 (top) and 50, 300, 600 (lower).....	65
Figure 93: Dendrogram for benchmarking data agglomerative hierarchical clustering .....	66
Figure 94: (Clickable) Hierarchical clustering prediction (left) and actual clusters (right) for 3 PC standardized unencoded benchmarking data .....	66
Figure 95: (Clickable) Hierarchical clustering prediction (left) and incorrect cell (right) for 2 PC standardized encoded benchmarking data .....	67
Figure 96: Dendrogram for simulated data agglomerative hierarchical clustering .....	67
Figure 97: (Clickable) Hierarchical clustering prediction (left) and incorrect cell (right) for 6 PC encoded benchmarking data.....	67
Figure 98: Dendrogram for mouse cortex data agglomerative hierarchical clustering .....	68
Figure 99: (Clickable) Hierarchical clustering prediction (left) actual clusters (middle) and incorrect predictions (right) for 9 PC standardized unencoded mouse cortex data.....	68
Figure 100: Loading scores for the top 20 principal components against 20 genes for the unstandardized (left) and standardized (right) benchmarking data.....	69
Figure 101: 10 genes with the highest loading scores for the first three principal components of the benchmarking data.....	69
Figure 102: Loading scores for the top 20 principal components against 20 genes for the unstandardized (left) and standardized (right) simulated data.....	70
Figure 103: 10 genes with the highest loading scores for the first three principal components of the simulated data .....	70
Figure 104: Loading scores for the top 20 principal components against 20 genes for the unstandardized (left) and standardized (right) mouse cortex data.....	70
Figure 105: 10 genes with the highest loading scores for the first three principal components of the mouse cortex data .....	71
Figure 106: Histogram displaying best accuracy of the six clustering algorithms, along with the optimal number of principal components and whether the benchmarking data was encoded or not.....	71
Figure 107: (Clickable) Hierarchical clustering prediction (left) and incorrect cell (right) for 3 PC standardized unencoded benchmarking data .....	71

Figure 108: Histogram displaying best accuracy of the six clustering algorithms, along with the optimal number of NMF basis components and whether the benchmarking data was encoded or not.....	72
Figure 109: (Clickable) Hierarchical clustering prediction (left) and incorrect cell (right) for 3 IC standardized unencoded benchmarking data.....	72
Figure 110: Histogram displaying best accuracy of the six clustering algorithms, along with the optimal number of independent components and whether the benchmarking data was encoded or not.....	72
Figure 111: (Clickable) K-means prediction (left) and true cell lines (right) for 2 NMF basis components on standardized benchmarking data.....	72
Figure 112: (Clickable) Gaussian mixture prediction (left) and identifying one of two incorrect cells (right) for 4 NMF basis components on standardized benchmarking data.....	73
Figure 113: Histogram displaying best accuracy of the six clustering algorithms with t-SNE, along with the optimal number of PCA components and whether the benchmarking data was encoded or not.....	73
Figure 114: (Clickable) Mini batch k-means prediction (left) and incorrectly identified cells (right) for 2 principal components and t-SNE for the encoded, standardized benchmarking data.....	73
Figure 115: Histogram displaying best accuracy of the six clustering algorithms, along with the optimal number of principal components and whether the simulated data was encoded or not.....	74
Figure 117: Histogram displaying best accuracy of the six clustering algorithms, along with the optimal number of independent components and whether the simulated data was encoded or not.....	74
Figure 116: (Clickable) BIRCH prediction (left) and cell lines (right) for 4 principal components on standardized benchmarking data.....	74
Figure 118: (Clickable) BIRCH prediction (left) and cell lines (right) for 4 independent components on standardized benchmarking data.....	74
Figure 119: Histogram displaying best accuracy of the six clustering algorithms, along with the optimal number of NMF basis components and whether the simulated data was encoded or not.....	75
Figure 120: (Clickable) Agglomerative hierarchical prediction (left) and true cell lines (right) for 11 NMF basis components on unstandardized benchmarking data.....	75
Figure 121: Histogram displaying best accuracy of the six clustering algorithms, along with the optimal number of t-SNE components and whether the simulated data was encoded or not.....	75
Figure 114: (Clickable) Spectral clustering prediction (left) and excepted groups (right) for 2 principal components with t-SNE for the encoded, unstandardized benchmarking data getting 100% accuracy.....	76
Figure 114: (Clickable) Spectral clustering prediction (left) and excepted groups (right) for 2 principal components with t-SNE for the encoded, unstandardized benchmarking data getting 50% accuracy.....	76
Figure 122: The original gene expression of the mouse cortex data (left) and bicluster regions (right).....	77
Figure 123: Gene expression of the standardized mouse cortex data.....	77
Figure 124: Rearrangement of the matrix after biclustering (left) and identification of the bicluster regions (right) for the standardized mouse cortex data.....	77
Figure 125: (Clickable) Simplicial complex of benchmarking data (left) and simulated data (right).....	78
Figure 126: (Clickable) Simplicial complex of the mouse cortex data.....	78
Figure 127: Print screen from testing the initial model showing a batch's input tensor, output tensor, accuracy, and average accuracy across all batches.....	91
Figure 128: Two batch's input tensors, outputs, accuracy, and average accuracy across all batches for ReLU-ReLU (left) and ReLU-Linear (right).....	91
Figure 129: Two batch's input tensors, outputs, accuracy, and average accuracy across all batches for ReLU-PReLU (left) and Linear-Linear (right).....	91
Figure 130: Results of 5-fold cross validation for the benchmarking dataset.....	92
Figure 131: Results of 5-fold cross validation for the simulated dataset.....	92
Figure 132: Results of 5-fold cross validation for the mouse cortex dataset.....	92
Figure 133: Batches and accuracy after training the simulated data autoencoder.....	93
Figure 134: Batches and accuracy after training the benchmarking data autoencoder.....	93
Figure 135: Batches and accuracy after training the simulated data autoencoder.....	93

## Table of Tables

Table 1: Terms.....	9
Table 2: Acronyms and Abbreviations.....	10
Table 3: Mathematical Notation.....	10
Table 4: Report structure.....	13
Table 5: Sub-objectives and prioritisation.....	24
Table 6: Risk identification and mitigation.....	27
Table 7: Planned experiments.....	35
Table 8: Loss of first run after 40 epochs.....	45
Table 9: Loss of second run after 40 epochs.....	46
Table 10: Performance for different learning rates.....	46
Table 11: Performance of recommended range test learning rate vs 1e-3 over 40 epochs.....	47
Table 12: Performance for different batch sizes over 40 epochs.....	47
Table 13: Performance for different optimisers over 40 epochs.....	48
Table 14: Performance of Adam vs AdamW over 100 epochs.....	48

Table 15: Performance based on number of hidden layers after 40 epochs.....	49
Table 16: Performance of different activation functions for hidden layers after 40 epochs.....	50
Table 17: Performance of combinations of activation functions after 100 epochs.....	50
Table 18: 5-fold validation loss for benchmarking, simulated and mouse cortex evaluation data .....	51
Table 19: Evaluation of objectives and sub-objectives.....	80
Table 20: Results of PCA and k-means on the benchmarking data .....	93
Table 21: Results of PCA and k-means on the simulated data.....	94
Table 22: Results of PCA and k-means on the mouse cortex data .....	94
Table 23: Results of k-means combined with PCA and t-SNE with perplexity 30 on the mouse cortex data .....	94
Table 24: Results of k-means and t-SNE for different perplexity values on the mouse cortex data .....	95
Table 25: Results of PCA and agglomerative hierarchical clustering on the benchmarking data .....	95
Table 26: Results of PCA and agglomerative hierarchical clustering on the simulated data .....	95
Table 27: Results of PCA and agglomerative hierarchical clustering on the mouse cortex data .....	95
Table 28: Benchmarking data alternative clustering algorithms results for PCA .....	96
Table 29: Benchmarking data alternative clustering algorithms results for ICA.....	96
Table 30: Benchmarking data alternative clustering algorithms results for NMF.....	97
Table 31: Benchmarking data alternative clustering algorithms results for PCA and t-SNE .....	97
Table 32: Simulated data alternative clustering algorithms results for PCA .....	97
Table 33: Simulated data alternative clustering algorithms results for ICA.....	98
Table 34: Simulated data alternative clustering algorithms results for NMF.....	98
Table 35: Simulated data alternative clustering algorithms results for PCA and t-SNE.....	98

## Table of Equations

Equation 1: Formation of a simplicial complex [42].....	21
Equation 2: Autoencoder identity function [57] .....	29
Equation 3: Adam weight update [59].....	29
Equation 4: Mean squared error [62] .....	30
Equation 5: Standardization or Z-score normalisation [65] .....	31
Equation 6: Sum of Squared Error (SSE) [66].....	31
Equation 7: Rand index [69] .....	32
Equation 8: Adjusted Rand index [70].....	32

## Terms, Abbreviations and Notation

This section provides an explanation of biological terms, acronyms, abbreviations, and mathematical notation that is used throughout the report.

### Terms

Term	Explanation
10x Chromium	A single-cell RNA sequencing protocol
Adenocarcinoma	A type of cancer that forms in mucus-secreting glands
Cell line	A population of cells maintained in a culture that will proliferate indefinitely
DNA	A molecule made from two polynucleotide chains that encode the genetic information of living organisms
Gene	A section of DNA that encodes a protein
High-throughput sequencing	A sequencing technology able to read the nucleic acid sequence of multiple DNA molecules in parallel
Phenotype	The observable physical characteristics of an organism
Protein	A large biomolecule made up of a chain of amino acids
RNA	A single-stranded molecule, structurally similar to DNA, involved in the synthesis of proteins
Single-cell RNA sequencing	The sequencing of RNA from individual cells of the same type

Table 1: Terms

### Acronyms and Abbreviations

Abbreviation	Definition	Abbreviation	Definition
Adam	Adaptive moment estimation	PCR	Polymerase chain reaction
AE	Autoencoder	PReLU	Parametric ReLU
ARI	Adjusted Rand index	ReLU	Rectified linear unit
BCV	Biological Coefficient of Variation	RI	Rand index
cDNA	Complementary DNA	RNA	Ribonucleic acid
DEGs	Differentially expressed genes	Rprop	Resilient backpropagation
DNA	Deoxyribonucleic acid	rRNA	Ribosomal RNA
ELU	Exponential linear unit	SAGE-AR	Self-Assessment Governance and Ethics form for Animal Research
HTS	High-throughput sequencing	SAGE-HDR	Self-Assessment Governance and Ethics form for Humans and Data Research
ICA	Independent Component Analysis	scRNA-seq	Single-cell RNA sequencing
LR	Learning rate	SGD	Stochastic gradient descent
MAE	Mean absolute error	SIMLR	Single-cell Interpretation via Multi-kernel LeaRning
ML	Machine learning	SSE	Sum of the squared error
mRNA	Messenger RNA	Tanh	Hyperbolic tangent function
MSE	Mean squared error	TDA	Topological data analysis
NGS	Next-generation sequencing	tRNA	Transfer RNA

NMF	Non-Negative Matrix Factorization
PC	Principal component
PCA	Principal component analysis

t-SNE	t-Distributed Stochastic Neighbour Embedding
VAE	Variational autoencoder

Table 2: Acronyms and Abbreviations

## Mathematical Notation

Notation	Explanation
$O(n)$	Big O notation indicating an algorithm with input size $n$ has linear time complexity
$O(n^2)$	Big O notation indicating an algorithm with input size $n$ has quadratic time complexity
$a \approx b$	$a$ is approximately equal to $b$
$a \leq b$	$a$ is less than or equal to $b$
$a \geq b$	$a$ is greater than or equal to $b$
$a \in X$	$a$ is an element of the set $X$ , for example $a \in \{a, b, c\}$
$X \subset Y$	$X$ is a subset of $Y$
$\{a \mid a > 0\}$	The set of all values of $a$ such that $a$ is greater than zero
$X \cap Y$	Intersection of $X$ and $Y$ is the set of all elements in $X$ and all elements in $Y$
$f : X \rightarrow Y$	A function from domain $X$ to domain $Y$
$f^{-1}(B)$	For a function $f : X \rightarrow Y$ where $A \subset X$ and $B \subset Y$ , $f(A) = \{f(x) \in Y : x \in A\}$ is the image of $A$ , while $f^{-1}(B) = \{x \in X : f(x) \in B\}$ is the preimage
$\epsilon$	Epsilon symbol used in mathematics as a variable
$\infty$	Infinity
$1e-1$	Equivalent to $10^{-1} = 0.1$
$A^T$	Transpose of matrix $A$ , in which if $A$ is an $m \times n$ matrix, then $A^T$ is an $n \times m$ matrix
$ a $	The absolute value of $a$ , i.e., a non-negative value ignoring its sign
$\sum_{i=1}^j a_i$	Sum of $a_i$ for all $i = 1, 2, \dots, j$

Table 3: Mathematical Notation

## Statement of Ethics

Throughout this project, it is my responsibility to ensure that the work follows a legal, ethical, social, and professional code of conduct in alignment with the British Computing Society [1]. Considerations include:

- ♦ **Public interest** – This project has been devised to avoid adverse effects to the public, environment, and reputation of the University. While it does make use of animal biological data, all data is secondary and is from experiments conducted by other researchers. No animals were harmed or physically present. Furthermore, the project complies with legislation such as the Computer Misuse Act 1990 [2] as no computer material has been accessed without permission, nor has other's data been altered.
- ♦ **Informed consent** – There was no participation from human subjects or the collection of personal data and therefore this is not applicable.
- ♦ **Confidentiality of data** – This project does not have a client nor financial data. All datasets used are available in the public domain and so no data is involved must be kept confidential. It should be noted that while genetic data is defined as special category data under the Data Protection Act (2018) [3], use of this data is authorised under the condition of explicit consent [4]. Therefore, the project complies with both the ACM Code of Ethics and Professional Conduct [3] and the DPA (2018) [3] as there is no sensitive data to protect.
- ♦ **Honesty and integrity** – It is my responsibility to be clear and truthful about the capabilities and limits of implemented systems. I understand that I must not make false claims, fabricate results, offer nor accept bribes [5] and that I could be held accountable if these principles are not followed.
- ♦ **Social responsibility** – The creation of any new research contributes to the shared knowledge available to society. The results of this project may provide future researchers with inspiration or understanding of new approaches. It is intended that anyone who uses the findings use them in socially responsible way.
- ♦ **Professional competence** – This project has been conducted in a professional manner under the guidance of a supervisor. This includes recognising ethical challenges, reflection of technical feasibility, planning and controlling risks.
- ♦ **Intellectual property** – Copyright, patents, and trademarks, have been respected and credit has been given where applicable for the use of IP of others. All datasets are open-source and freely available to the public and so the publisher has given consent. If a publisher revoked permission and requested the removal of the data, I would be ethically obliged to delete copies of the dataset and all test data that is traceable back to the source. Though, I would not be obliged to delete the overall results as they are not traceable back to the source data. Additionally, this report is the intellectual property of the creator and anyone passing the research as their own is committing plagiarism. If others wish to use the findings, they should provide a citation.

## Ethics Review

In compliance with university policy, a SAGE-HDR form was completed at the beginning of the research to assess whether an ethics review from a committee was required. After completion of the form, it was concluded that an ethics review would not be required. The full SAGE-HDR has been included in Appendix A.

As the research involves biological data from animals, a SAGE-AR form was completed in addition. No ethical review of the study was required as all testing and training uses secondary data originally collected for other research. The full SAGE-AR form can be seen in Appendix A.



# 1. Introduction

This section details the problem background, project description, aims and objectives, success criteria and report structure.

## 1.1 Problem Background

In molecular biology, proteins are produced in cells through two steps: transcription and translation. Transcription is when sections of DNA are copied to produce a complementary molecule called messenger RNA (mRNA), while translation is the process in which mRNA is used to produce a protein. This means that cellular gene expression can be inferred through analysing RNA.

Improvements in high-throughput sequencing technologies (HTS), also known as next-generation sequencing (NGS), have provided new methods for scientists to study the inner workings of cells at a molecular level [6]. The sequencing of RNA for individual cells of the same type using HTS is known as single-cell RNA-seq.

To sequence RNA, it must first be converted to DNA [6]. Multiple DNA molecules can be sequenced in parallel using HTS to rapidly produce large volumes of data. This is unlike techniques used in the past, such as Sanger sequencing, which sequence one molecule at a time [7]. With the ability to create larger and more complex datasets, scientists must rely on newly developed computational techniques for evaluation.

## 1.2 Project Description

Within the project, experiments are conducted comparing dimensionality reduction techniques and clustering methods to interpret single-cell RNA-seq biological datasets with the aim of revealing hidden variation in gene expression of cells of the same type. In addition, Mapper is used to visualise the high-dimensional data, and this is evaluated against dimensionality reduction.

This is a type of secondary research as all experiments were conducted on previously collected or generated data. While similar research projects have been conducted before, I aim to create fresh results on which new conclusions can be made.

This will be achieved in three main sections: a detailed literature review, substantial implementation of the numerous experimentations, and an evaluation of methods based upon performance, accuracy, and efficiency.

## 1.3 Aims and Objectives

The aim of the project is to experiment with clustering and topological data analysis to detect hidden gene expression in three single-cell RNA-seq datasets. The following are a list of objectives:

1. Research and understand single-cell RNA-seq and the resulting datasets
2. Review literature for data encoding techniques, neural network optimisation methods, clustering methods and topological data analysis
3. Implement an autoencoder that can extract a lower dimensional representation of biological data
4. Test and experiment with different neural network architectures and apply optimisation techniques to the autoencoder to determine how these change the quality of the encoding
5. Apply clustering methods to the encoded data
6. Apply a topological data analysis method to the data
7. Evaluate the performance of topological data analysis in comparison to clustering methods
8. Review the results of the project and recommend future improvements

## 1.4 Success Criteria

To determine if the project has been successful, it will be assessed against the aims and objectives listed above. The criteria required to meet these objectives is outlined in the Requirements section, while evaluation is documented in Conclusions and Future Work.

## 1.5 Report Structure

Section	Description
1. Introduction	An introduction to the report briefly explaining the project's topic, a high-level overview, the objectives, success criteria and the report structure.
2. Literature Review	This part includes a more detailed explanation of the project topic and research conducted by others. Data encoding techniques, methods to optimise neural networks, clustering algorithms and topological data analysis are explored in depth. Additionally, an overview is given of suitable datasets and a comparison between machine learning libraries.
3. Requirements and Specification	This section explores the project's scope in more detail than the Introduction. Objectives are prioritised and split into sub-objectives to identify steps required for them to be met. Hardware and software specifications are explained, and the project's feasibility is investigated. The project plan is displayed in a Gantt chart and milestones, dependencies and risks are identified.
4. System and Experiment Design	Details the design of the system before implementation. This includes the initial architecture of the autoencoder, further exploration of clustering encoded data, how topological data analysis will be implemented and the plan to create of a simulated dataset. Design choices are justified, and challenges are highlighted. Also contains a table of experiments to be carried out in Testing and Validation.
5. Implementation	This part contains documentation of the development process, implemented code and challenges. This includes importing the datasets, creating the autoencoder, clustering, topological data analysis and the creation of the simulated dataset.
6. Testing and Validation	In this section, the technical details of the datasets are explained, and the results of conducted experiments are documented in the form of tables and graphs. This includes autoencoder optimisation, training the optimised models, applying clustering techniques and topological data analysis with Mapper. Additional results from this section are available in Appendix B.
7. Conclusions and Future Work	The final part of the project consists of a general overview, evaluation against objectives, explanation of the results and implications, suggestions for future research and the final statement.

Table 4: Report structure

## 2. Literature Review

This part of the report explores single-cell RNA sequencing, data encoding techniques, optimisation of neural network architectures, clustering techniques and topological data analysis in depth. This is essential to meet objectives 1 and 2.

### 2.1 Single-Cell RNA Sequencing

RNA sequencing experiments provide an abundance of big data suitable for evaluation using computational methods. These datasets are large and complex, so it is important that a researcher understands the data before they can evaluate it. Therefore, this section explores the technical details of single-cell RNA sequencing, benefits over other techniques and challenges.

#### 2.1.1 Purpose

Single-cell RNA-seq enables researchers to measure gene expression across a population of heterogeneous cells [8]. This includes identifying which genes are active, when genes are active, and their level of expression.

Since the first study in 2009, this technique has helped scientists make many discoveries and breakthroughs in medicine. For example, analysing genomic differences between individual cells in a tumour mass has previously revealed malignant tumour cells [9]. Similarly, researchers have been able to examine unique individual cells, such as cells within early stages of a developing embryo.

#### 2.1.2 How It Works

Figure 1 shows the typical steps of RNA sequencing. First viable, individual cells are isolated from a tissue sample. For each cell, the membrane is broken down in a process called lysis as this improves the capture of RNA molecules [9]. Then the process of reverse transcription is used to create tagged complementary DNA (cDNA) synthesized from messenger RNA (mRNA). This cDNA is known as a cDNA library [10].

Small volumes of cDNA are amplified through a technique such as polymerase chain reaction (PCR) which facilitates rapid creation of millions of copies. Each cell's cDNA library is then pooled and sequenced by next-generation sequencing (NGS) to produce a dataset [9].

#### 2.1.3 Benefit Over Alternative Techniques

The standard method of RNA-sequencing, bulk RNA-seq, can only determine average gene expression levels over a large population of cells. In comparison, scRNA-seq provides a more accurate representation at a cell-to-cell level (see Figure 2).

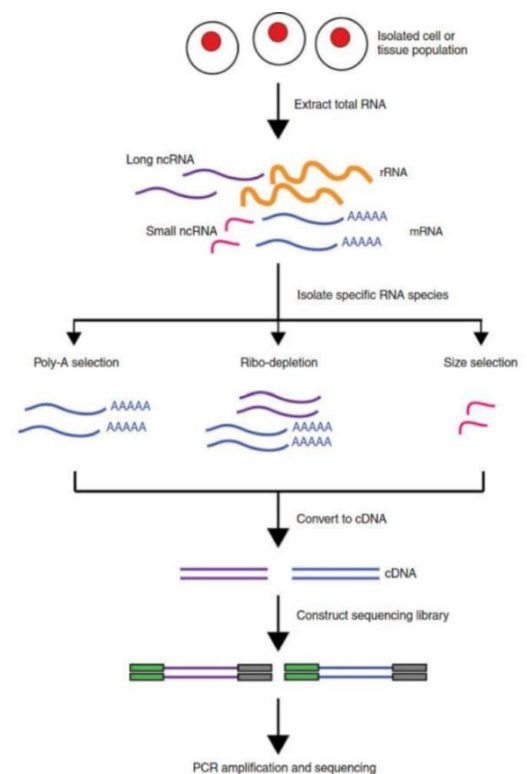


Figure 1: RNA sequencing steps [97]

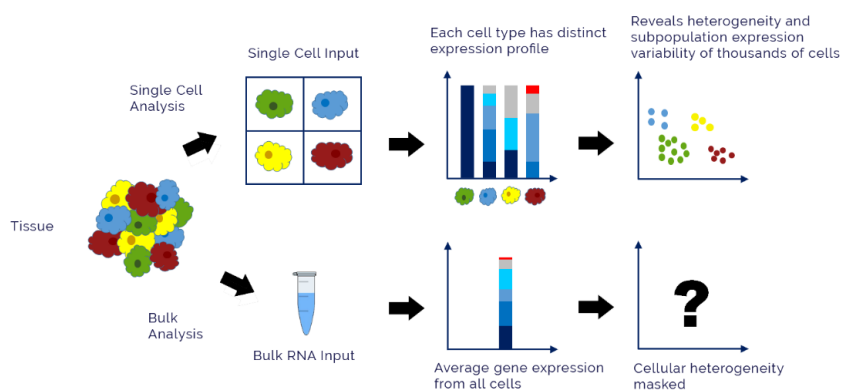


Figure 2: Single-cell vs bulk RNA sequencing [11]

### 2.1.4 Challenges

During RNA-sequencing, scientists often read gene expression for thousands if not millions of cells across thousands of different genes. This produces large volumes of data requiring high storage capacity and computationally expensive analysis [12].

Noise and unexpected variation can be introduced due to a variety of reasons including the local environment of cells, variety in the rate of RNA processing or changes in the cell cycle. Similarly, for many genes, no expression is detected within a cell producing a zero in the data. Either the gene is not present, or it has been missed producing a false negative. This is common in methods such as droplet-based scRNA-seq which frequently produces sparse datasets [5]. However, redundant values and noise can be reduced through data encoding. This can also greatly decrease dataset size and improve performance during computational evaluation.

## 2.2 Data Encoding Techniques

Analysing a scRNA-seq dataset typically involves normalisation, feature extraction and dimension reduction. Normalisation is the adjustment of data to account for differences in experimental conditions [13]. However, this step cannot be included as the project will only use secondary or generated data. Feature extraction attempts to discover single or combinations of genes which best represent a cell sample. As a result, genes that do not exhibit any biologically significant variation are disregarded [13]. Dimension reduction aims to learn a lower representation of the data which reflects the gene expression of each cell sample. This means that data can be compressed without loss of significant characteristics.

Two popular methods used to perform feature extraction and dimension reduction include principal component analysis and autoencoders. Therefore, their suitability will be evaluated in this section.

### 2.2.1 Principal Component Analysis (PCA)

Principal component analysis is one of the most widely used techniques to reduce the dimensionality of datasets [14]. It operates by transforming data into principal components and selecting the top k that represent a high percentage of variation within the data. All other components are removed as they do not significantly benefit the model [15].

A lower dimensional hyperplane is constructed that best maximises the amount of variance within the data. Figure 3 shows an example in which PCA selects two dimensions, PC1 and PC2, that provide the highest variance and second highest variance respectively.

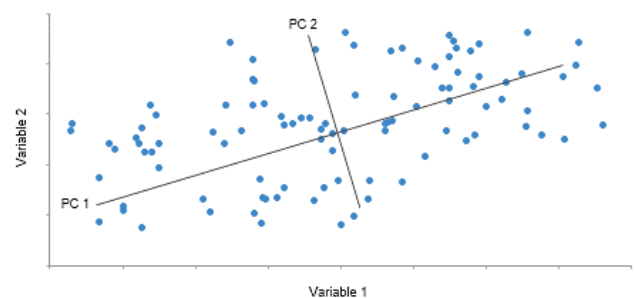


Figure 3: Example of PCA example showing two principal components [16]

#### 2.2.1.1 Previous Work

Many previous studies have applied PCA to the analysis of scRNA-seq data. For example, research by Sato et al. [15] compared the performance of 10 PCA algorithms with four clustering methods on large datasets. Each dataset contained millions of samples consisting of human cells from the blood or pancreas, or mouse cells from the brain or spinal cord. The results show that some implementations of PCA can be fast, accurate and memory efficient. Furthermore, this study is useful as it provides guidelines for selecting an appropriate PCA algorithm.

### 2.2.2 Autoencoders

#### 2.2.2.1 Deterministic Autoencoder

Autoencoders are an unsupervised learning technique created using a neural network [16]. A deterministic autoencoder will take an input vector and attempt to reconstruct it so that the output vector is as similar as possible [17]. Often, it is trained over multiple iterations to minimise mean squared error using gradient descent [15]. An autoencoder's goal is to find a function that maps features to itself. This is achieved using an encoder and decoder as shown in Figure 4. First the encoder learns to represent data in a lower-dimensional space by extracting important features. Then the decoder learns to reconstruct the original data using the encoded feature representation [18].

### 2.2.2.2 Variational Autoencoder

A variational autoencoder (VAE), such as that shown in Figure 4, is a type of autoencoder that returns a probability distribution rather than a single value for each attribute. Unlike a standard autoencoder, the VAE encoder outputs a vector of means  $\mu(x)$  and vector of variance  $\sigma(x)^2$ . These are used to create a Gaussian distribution that allow a range of encodings to be generated as opposed to fixed values [19].

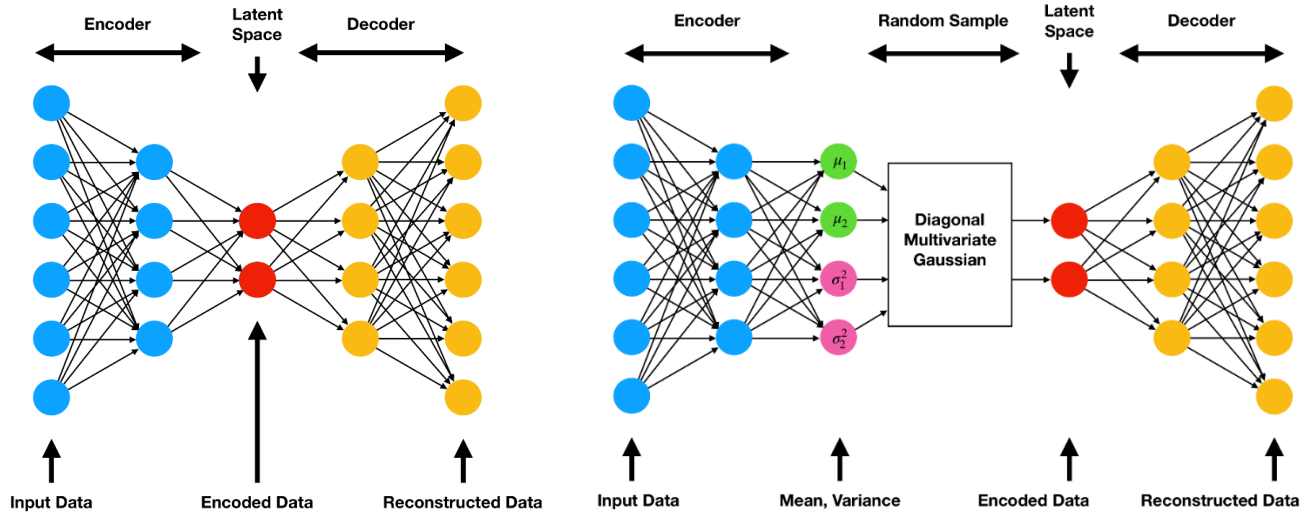


Figure 4: Standard deterministic autoencoder architecture (left) and variational autoencoder architecture (right) [19]

### 2.1.1.1 Previous Work

In a study conducted by Kim et al. [15], a VAE was trained by unsupervised learning on 20 scRNA-seq datasets from The Cancer Genome Atlas [20]. The learnt weights were then transferred to a survival prediction model they named VAECox. Transferring and fine-tuning weights was shown to be more effective than when VAECox's weights had been randomly initialised. This indicates that using the autoencoder improved the model's ability to identify gene expression useful for predicting patient survival for a range of cancers.

### 2.1.2 PCA vs Autoencoders

- ◆ PCA is a linear transformation, while autoencoders can learn non-linear functions to represent data through using neural networks. This allows autoencoders to learn a more complex representation of the data.
- ◆ Autoencoders combine features into a compressed layer of size  $k$ , while PCA select the top  $k$  features. This means that autoencoders are better at retaining the structure of the original data [15].
- ◆ If a small  $k$  is selected, autoencoders tend to produce a lower reconstruction error than PCA at the same  $k$  value. Therefore, smaller datasets can be used by an autoencoders to produce the same error. This is advantageous when dealing with big data. However, when  $k$  is increased, error converges [15].
- ◆ As an autoencoder learns to extract features from training data, they are often only capable of reconstructing data of a similar class as the training set [14]. In comparison, PCA produces a more generalised model that can be used for datasets with greater variation. However, the robustness of PCA is not required for this project as testing will be limited to data with a similar structure.

### 2.1.3 Conclusion

PCA and autoencoders are both suitable methods for encoding scRNA-seq data. However, for the reasons above, I decided to focus upon deterministic autoencoders as the main method of dimensionality reduction. Consequently, the technical details of autoencoders are explored further in the System Design section.

## 2.2 Neural Network Optimisation

Neural networks can be optimised through tuning parameters and hyperparameters. A model's parameters are those that are learnt during training, such as weights and biases, while hyperparameters are those set beforehand, such as number of hidden layers and nodes, learning rate, and activation function [21].

According to a guide by G. Liu [22], learning rate is the most important hyperparameter, followed by batch size, optimiser, and number of epochs. Therefore, this section explores how these hyperparameters can be set to improve performance.

## 2.2.1 Optimisers and Learning Rate

Common optimisers include resilient backpropagation (Rprop), stochastic gradient descent (SGD) and Adam. These algorithms update model weights to minimise a loss function. Changing the optimiser can affect a model's accuracy, speed of convergence to an optimum, and run time. For example, Figure 5 demonstrates the influence of various optimisers on validation accuracy.

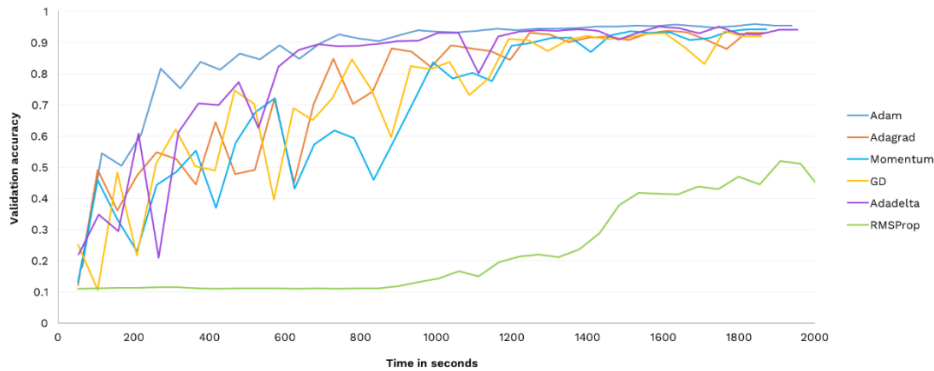


Figure 5: Example showing validation accuracy over time for different optimisers [23]

The magnitude an optimiser's weights are updated is determined by learning rate [24]. If learning rate is too low, the optimiser will make small changes to the weights and training will be slow. Similarly, if it is too high, training may be unstable. Therefore, optimising learning rate is a key step to improving speed and performance as exhibited in Figure 7.

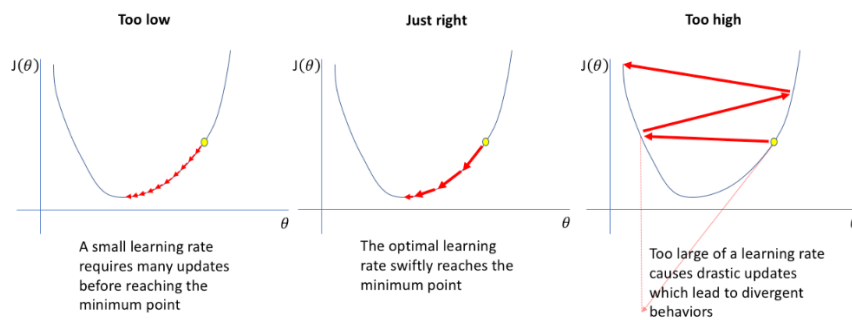


Figure 7: Learning rate comparison [24]

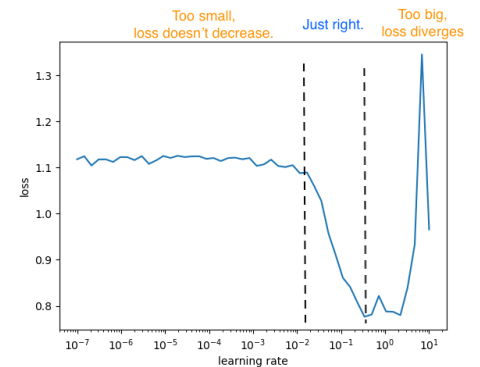


Figure 6: Example of learning rate optimisation using LR Range Test [26]

### 2.2.1.1 LR Range Test

The LR Range Test [25] is a trial-and-error technique for learning rate optimisation. Learning rate is initialised at a low value and the model is run for several iterations with learning rate increasing linearly or exponentially [26]. Loss at each learning rate is plotted (see Figure 6) and a new learning rate is selected from the region where loss converges fastest.

## 2.2.2 Batch Size

Batch size is the number of training samples used to estimate error gradient before a model's weights are updated. Batch gradient descent is when batch size equals the number of training samples. Stochastic gradient descent is when batch size is set to one. If batch size is anywhere between one and the total training set, this is mini-batch gradient descent [27].



### 2.2.2.1 Selecting Batch Size

Batch gradient descent is most suited to small datasets with less than 2000 training samples [28], and becomes inefficient when using lots of training samples. In this case, a smaller batch size between one and several hundred is recommended as this can improve training stability and performance. It is also advised to shuffle samples as changing the order of the batches per epoch regularly leads to faster convergence [29]. To find a suitable batch size, the model can be run over a set number of epochs, such as 30, with multiple batch sizes. The batch size with the lowest loss is then selected [22].

### 2.2.3 Epochs

In machine learning, epochs are the number of complete passes through the training data. A single epoch consists of one or more batches, depending on the type of gradient descent. Once an epoch is complete, this indicates each training sample within every batch contributed towards updating the model's parameters [30].

#### 2.2.3.1 Overfitting and Underfitting

Overfitting is when a network learns to model training data too well. This can be a problem as the model picks up on noise meaning it does not generalise well when applied to test data. A network may start to overfit if trained over too many epochs. In comparison, underfitting is when a network does not model the training data, nor generalise on test data. This often occurs if the network has not been trained for long enough or if the training data does not contain enough samples. Both underfitting and overfitting should be avoided as they lead to poor performance on testing data [31]. A visual example can be seen in Figure 9.

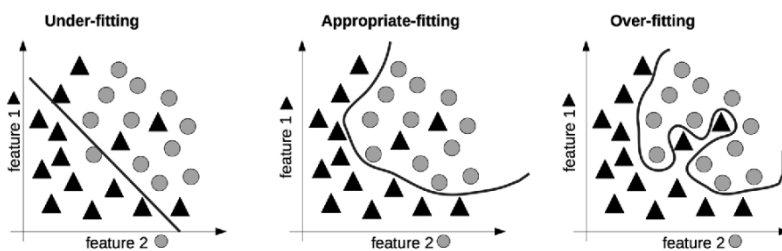


Figure 9: Comparison between underfitting and overfitting

#### 2.2.3.2 Early Stopping

The ideal number of epochs can be determined as the point before testing or validation accuracy decreases, and the model starts to overfit. This is called early stopping and can be visualised by the line in Figure 8. Generally, this is achieved through counting a set number of times in which no improvement is observed [32]. However, in practise, early stopping may lead to sub-optimal performance due to fluctuations in accuracy.

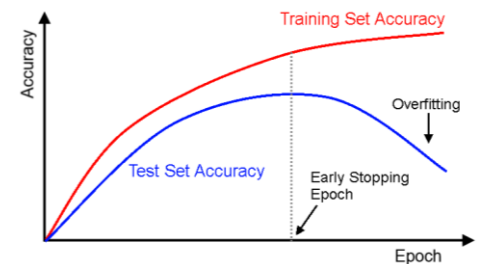


Figure 8: Trade-off between training and testing accuracy [32]

### 2.2.4 Conclusion

To conclude, this part considers the most important hyperparameters to tune to reduce loss and subsequently improve a network's accuracy. Optimisation of hyperparameters is investigated further in the System and Experiment Design section.

## 2.3 Clustering Techniques

Clustering is an unsupervised learning method in which data points are divided into groups with similar traits [33]. When analysing scRNA-seq gene expression data, the goal is to group similar genes by adding them to the same cluster [34]. Two effective and widely used algorithms are k-means and hierarchical clustering, and so they are investigated in this section.

### 2.3.1 K-Means Clustering

K-means is a type of partitional clustering as data points are divided into non-overlapping groups as demonstrated in Figure 10. It uses an iterative, centroid based algorithm. Before starting, the number of clusters,  $k$ , must be specified. At initialisation, every data point is randomly assigned a cluster and centroids are computed. For each iteration, data points are reassigned a cluster based upon its closest centroid. The centroids are re-computed, and the process repeated until no improvement is observed for two consecutive repeats or after the maximum number of iterations [33].

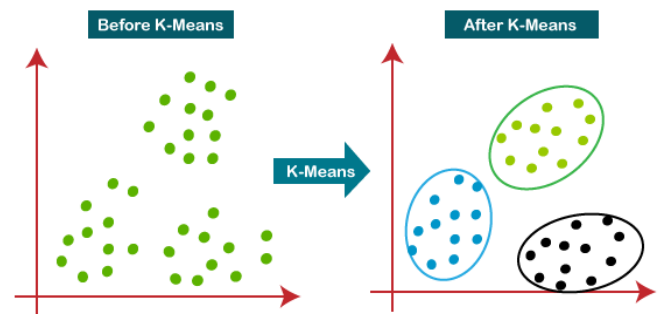


Figure 10: K-means clustering example [102]

#### 2.3.1.1 Previous Work

Ensemble clustering is a technique in which multiple clustering models are combined to improve performance. A study by Geddes et al. [35] conducted experiments applying ensemble learning to k-means clustering, and separately with SIMLR, a clustering method designed for RNA sequencing data. Before clustering, random subspace projections were taken from eight sc-RNA-seq datasets and compressed using an autoencoder. Ensemble learning was then applied over all encoded data to create cell clusters. The results suggest that applying ensemble clustering improved the accuracy of k-means by 30% and SIMLR between 50% and 100%. Although ensemble clustering is outside of the scope of this project, this study highlights that k-means is a valuable technique for the analysis of scRNA-seq data.

### 2.3.2 Hierarchical Clustering

Hierarchical clustering is a connectivity-based algorithm implemented using either a bottom-up or less frequently a top-down approach. Agglomerative clustering is a bottom-up approach that starts by assigning each data point to its own cluster. The nearest two clusters are then merged to form the same cluster. This is repeated and the algorithm terminates when there is one cluster remaining. By comparison, divisive clustering is a top-down approach that starts with one cluster which is repeatedly split into the least related clusters until only single data points remain [36].

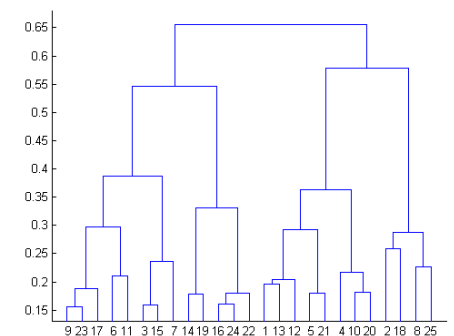


Figure 11: Example of agglomerative hierarchical clustering dendrogram [33]

The results are displayed in a dendrogram (Figure 11). The height at which two clusters merge represent the distance between them in the data space [33]. The optimal number of clusters is that which best distinguishes each group. By examining the dendrogram, this can be determined by finding the largest difference in distance on the y-axis in which the number of clusters does not change [33]. For example, in Figure 11, the best number of clusters is four as the largest distance is  $0.55 - 0.4 = 0.15$ .

#### 2.3.2.1 Previous Work

When gene expression is inconsistent across samples of the same cell type from separate populations, these are known as differentially expressed genes (DEGs). It is assumed that DEGs are the main cause of multiple phenotypes [37]. A paper by Koch et al. [38] applied hierarchical clustering and k-means clustering to DEGs from three populations of macrophage cells. Although some clusters assigned by the two techniques were not identical, they appear to show a similar pattern.

### 2.3.3 K-Means vs Hierarchical Clustering

- ♦ The time complexity of k-means is linear  $O(n)$ , while hierarchical is quadratic  $O(n^2)$  meaning k-means is better suited to handling big data [33].
- ♦ K-means is non-deterministic because the algorithm starts by randomly assigning clusters. As a result, different outcomes can be produced on separate runs using the same input. By contrast, hierarchical clustering is deterministic and so the results will always be the same for a given input.
- ♦ K-means works best when clusters have a spherical shape but does not perform well for clusters with complex shapes and different sizes. Hierarchical clustering can be sensitive to outliers [36].



### 2.3.4 Conclusion

Both k-means and hierarchical clustering are appropriate for clustering scRNA-seq data. Therefore, I decided the two algorithms will be implemented, and their performance compared. Further research on these techniques is conducted in the System Design section.

## 2.4 Topological Data Analysis

In mathematics, topology is the study of non-numeric geometric properties such as smoothness and connectedness. Topological data analysis (TDA) is the application of topology to understand the geometric nature of datasets [39].

### 2.4.1 Purpose

TDA is a technique for the investigation of high-dimensional data, such as scRNA-seq datasets. The problem with these datasets is that points cannot be properly plotted as they have too many coordinates to be visualised. However, TDA can resolve this by creating a structure to visually represent the full shape of the data [40]. Additionally, TDA is resistant to noise and missing data by retaining only significant features [41].

A more popular method for visualising scRNA-seq data is t-Distributed Stochastic Neighbour Embedding (t-SNE). It works particularly well at identifying cell type clusters. However, sometimes a population of cells may convey a spectrum of gene expression meaning that they cannot easily be separated into groups. In this situation, t-SNE would not be suitable [35].

### 2.4.2 Topological Signatures

A topological signature is a simplified representation of topology in a space that summarises a dataset. A common representation of a signature is a simplicial complex. This is a set of n-dimensional simplices, in which 0-simplex is a vertex, 1-simplex is a line, 2-simplex is a triangle, and 3-simplex is a tetrahedron [42]. The first four simplices are presented in Figure 12.

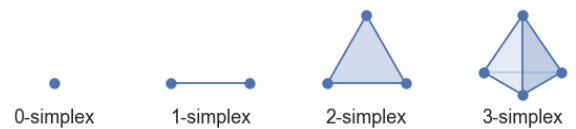


Figure 12: Low-dimensional simplices [105]

### 2.4.3 The Vietoris-Rips Complex

The Vietoris-Rips complex, often known as Rips complex, is an example of a topological signature. A simplicial complex  $VR_\epsilon(X)$  is constructed using data points as a vertex set  $X$  and parameter  $\epsilon \geq 0$ . For each pair of points in the vertex set with distance less than  $\epsilon$ , a 1-simplex edge is formed between them [42]. The higher the value of  $\epsilon$ , the more simplices that can be formed. This can be visualised using balls with radius  $\epsilon/2$ , although note  $\epsilon$ -balls are not actually part of the complex [37]. For example, Figure 13 demonstrates how more simplices are formed when  $\epsilon$  is increased as more of the ball intersect.

However, increasing this threshold too much can also result in the loss of topological features. This is because if the balls overlap too much, the higher dimensional shape will become too complicated and distinctive features will no longer be visible. For example, increasing  $\epsilon$  further in the last simplicial complex in Figure 13 would cause the ring shape to disappear.

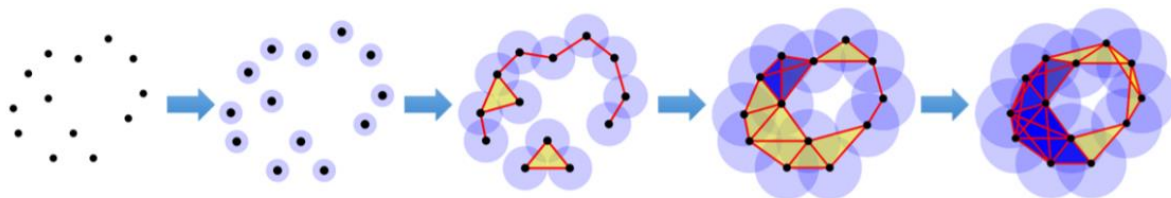


Figure 13: Example in which a simplicial complex is generated by increasing  $\epsilon$  [43]

When forming a higher dimensional simplex across multiple vertices, it can only be included if all lower dimensional simplices are also included between them [40]. For example, in the third image in Figure 13, a 2-simplex triangle is formed between three vertices only if each is also connected by an edge. The formation of a Rips complex from non-empty intersections can be generalised by Equation 1.

$$B(X_1, \epsilon) \cap B(X_2, \epsilon) \cap \dots \cap B(X_n, \epsilon)$$

$$\text{Where } B(X_i, \epsilon) = \{x \in X \mid d(X_i, x) \leq \epsilon\}$$

Equation 1: Formation of a simplicial complex [42]

Where:

- ◆  $X_i$  is the  $i^{\text{th}}$  vertex
- ◆  $B(X_1, \epsilon) \cap \dots \cap B(X_n, \epsilon)$  is the intersection of  $\epsilon$ -balls for vertices 1, ..., n
- ◆  $d(X_i, x)$  is the distance between vertex  $X_i$  and another vertex  $x$

## 2.4.4 Persistent Homology

The idea of persistent homology is that for a space represented as a simplicial complex, features retained over a large parameter range are statistically significant [42]. For example, with a Rips complex, it is advisable to investigate a range of possible values for  $\epsilon$ . This is because persistent features are more likely to represent the true features of the dataspace rather than those caused by noise [44].

## 2.5 Datasets

Multiple datasets are to be used to thoroughly test and evaluate the performance of the dimensionality reduction, clustering methods and topological data analysis. Initially I will experiment with a benchmark dataset, then after completing implementation I plan on evaluating it against both simulated and real data. Therefore, this component investigates the three selected scRNA-seq datasets.

### 2.5.1 Human Lung Adenocarcinoma Benchmark Dataset

For testing during development, the tabular dataset sc\_10x, has been selected. This is a small set of genetic data composed from a mixture of three cell lines. All data has passed quality control checks and contains no gene filtering [45]. It is part of a larger dataset sc\_mixology created by Luyi Tian that additionally contains results from clustering experiments. This is useful to provide a comparison to evaluate the performance of clustering algorithms implemented in this project.

The dataset was created to test scPipe, a pre-processing pipeline for scRNA-seq data proposed by Tian et al. [46]. The aim of the research was to create a new R package compatible with data generated from a variety of scRNA-seq protocols. To generate sc\_10x, human lung adenocarcinoma cells were cultivated from cell lines: HCC827, H1975 and H2228. These cells were processed by 10x Genomics' Chromium technology and sequenced using Illumina Nextseq 500. A gene expression matrix was then produced through scPipe.

### 2.5.2 Splat Simulated Evaluation Dataset

The benefit of simulated data is that it can be quickly generated at low cost. This is useful to provide more testing samples and can also provide a ground truth to compare against when evaluating a model's performance and limitations [47]. Splatter [48] is an R package proposed by Zappia et al. [47] that provides six different models to create simulated sc-RNA-seq data. Although other simulations tools exist, the motivation behind Splatter is to create data that is provably accurate and reproducible.

In this paper, each model is evaluated, including their own proposed model Splat. The results suggest that performance highly depends on the data's source. For example, Splat and BASiCS were both able to create the closest matches to real data for induced pluripotent stem cells. However, Splat achieved the worst performance when simulating cerebral organoid cells [47]. Therefore, it is a good idea to experiment with several simulators.

### 2.5.3 Mouse Cortex Evaluation Dataset

In an article published in Science by Zeisel et al. [49], an mRNA dataset [50] was created by large-scale scRNA-seq experiments from mouse cortex and hippocampus cells. They attempted hierarchical clustering, though found that this alone resulted in fragmented clusters as many of the genes were not related to every sample. To solve this, they created a new biclustering algorithm BackSPIN. This algorithm was then used to cluster the cells into nine classes and was applied again to these classes revealing 47 molecularly distinct subclasses. Together,

these comprise all known major cell types in the cortex [49].

## 2.6 Machine Learning Libraries

PyTorch and TensorFlow are two popular machine learning libraries considered for implementing the autoencoder.

### 2.6.1 PyTorch

PyTorch is a free, open-source library for Python or C++ programming [51]. It consists of many modules suitable for creating neural networks [52]. Deep learning is reliant on tensors, which are multi-dimensional matrices. PyTorch is built upon the Torch library, so provides a large library of operations on tensors. A benefit of PyTorch tensors compare to tensors used by other frameworks, such as NumPy arrays, is that they can be operated on by CPU or GPU [52]. Using GPU can improve performance which is particularly useful when training networks on large datasets. The layer-by-layer construction of networks in PyTorch makes them highly customisable. This is an important property for experimentation with the autoencoder's architecture. This includes testing different activation functions and loss functions to optimise the network.

### 2.6.2 TensorFlow with Keras

TensorFlow is an open source, deep learning library while Keras is a high-level API that sits on top. Keras has a simple architecture while TensorFlow is more complex and often difficult to use [53]. Therefore, the Keras API has been integrated into TensorFlow, making the TensorFlow library more user friendly [54]. Like PyTorch, a network is constructed by joining layers, with each layer configured according to the number of nodes and activation function.

### 2.6.3 PyTorch vs TensorFlow

- ◆ TensorFlow is older and so currently has a wider range of documentation and tutorials.
- ◆ PyTorch code is simpler making it suitable for newer developers, while TensorFlow is only recommended to intermediate-level developers.
- ◆ PyTorch has strong support for GPU acceleration, while TensorFlow only supports NVIDIA GPUs [55].
- ◆ TensorBoard is a logger provided by TensorFlow useful for visualising graphs. However, integration is provided for PyTorch [56].
- ◆ It is easier to deploy machine learning models on TensorFlow. Though, this is out of scope of this project.

### 2.6.4 Conclusion

PyTorch and TensorFlow provide similar features and either could be used to implement the autoencoder. However, I have selected PyTorch because code is cleaner, and has better support for GPUs.

### 3. Requirements and Specification

This section identifies the project requirements, hardware and software specifications, analysis of project feasibility and details the project plan.

#### 3.1 Requirements

This part specifies the requirements necessary to evaluate the success of the project during Conclusions and Future Work.

##### 3.1.1 Sub-Objectives and Prioritisation

For every objective outlined in the Introduction, sub-objective(s) are identified detailing the specific steps required to achieve it. Each is given a priority between 1 – 5, in which 5 is the highest priority and 1 means it is optional.

ID	Objective	Sub-objective	Priority (0 – 5)
1.1	Research and understand single-cell RNA-seq and the resulting datasets	Review multiple sources, including papers and articles on single-cell RNA-seq	4
1.2		Write a summary of the technical details of RNA-seq and challenges with the type of data in the Literature Review	3
1.3		Select datasets for benchmarking and evaluation	5
2.1	Review literature for data encoding techniques, neural network optimisation methods, clustering methods and topological data analysis	Research and compare at least two data encoding methods in the Literature Review	4
2.2		Research neural network optimisation techniques and write a summary in Literature Review	4
2.3		Research and compare at least two clustering methods in the Literature Review	4
2.4		Research topological data analysis and write a summary in the Literature Review	2
3.1	Implement an autoencoder that can extract a lower dimensional representation of biological data	Split dataset into testing and training data	5
3.2		Implement an encoder that can produce a compressed version of input data with less features	5
3.3		Implement a decoder that can reconstruct the compressed data so that it has the same number of features as the input data	4
4.1	Test and experiment with different neural network architectures and apply optimisation techniques to the autoencoder to determine how these change the quality of the encoding	Experiment with different learning rates and update the autoencoder to use the optimum	4
4.2		Experiment with different batch sizes and update the autoencoder to use the optimum	4
4.3		Experiment with different optimisers and select the best for the autoencoder	3
4.4		Experiment with different numbers of hidden layers and features to optimise the structure of the autoencoder	3
4.5		Improve decoded output so it produces 50% or less error	1
4.6		Train the optimised autoencoder	5
4.7		Convert autoencoder to VAE autoencoder	1
5.1	Apply clustering methods to the encoded data	Run the trained autoencoder on test data to retrieve the encoding	5
5.2		Perform k-means clustering on encoded data	5
5.3		Perform agglomerative hierarchical clustering on encoded data	5
5.4		Perform clustering on secondary datasets	2

6.1	Apply a topological data analysis method to the data	Create a simplicial complex from the first dataset	3
6.2		Plot the simplicial complex	3
6.3		Perform TDA on secondary datasets	2
7.1	Evaluate the performance of topological data analysis in comparison to clustering methods	Document clustering results in report	5
7.2		Document TDA results in report	3
7.3		Compare the results from the two methods and explain findings	2
8.1	Review the results of the project and recommend future improvements	Evaluate project against aims and objectives	5
8.2		Suggest how the project could have been improved with hindsight and how it could be continued in the future	5

Table 5: Sub-objectives and prioritisation

## 3.2 Hardware and Software Specification

### 3.2.1 Hardware Specification

No specific hardware is required as code can be executed through a browser on Google Colab. It is recommended to use a GPU if training a new autoencoder. In Colab, this can be done by pressing “Runtime” → “Change runtime type” → select GPU from “Hardware accelerator”.

### 3.2.2 Software Specification

Code is written in Python 3.6.9 and R 4.0.3 across two Jupyter notebooks. During development, they were executed through Colab via the Google Chrome browser. The main libraries are listed below:

#### Python libraries:

- ◆ Torch
- ◆ PyTorch Lightning
- ◆ Pandas
- ◆ Numpy
- ◆ Scikit-learn
- ◆ Plotly
- ◆ KeplerMapper

#### R libraries:

- ◆ Bioconductor
- ◆ Splatter

## 3.3 Feasibility Analysis

Before development, it is crucial that the project is determined to be achievable and within the required scope. In particular, the project will be assessed against technical, legal, economic, and scheduling feasibility.

### 3.3.1 Technical Feasibility

Regular discussions with my supervisor allowed me to develop a project that is both challenging and yet within the scope of my technical capabilities as a computer science student. Before starting, I had several years’ programming experience with Python and have used Pytorch in a third-year module. However, this project is far more difficult technically than I have experienced before and will allow me to explore the framework in more depth. Performing a comprehensive literature and technology review prior to implementation was vital so that I could explore relevant topics to gain the understanding required during development.

Additionally, programming in Google Colab notebooks enables code to be executed through a browser with no set up, no hardware constraints and access to GPUs. This means I will not be reliant on any specific device and only require an Internet connection.

### 3.3.2 Legal Feasibility

The project has been developed in line with relevant legislation as specified earlier in the Statement of Ethics. Furthermore, this project is intended for research purposes only.

### 3.3.3 Economic Feasibility

This project is for research purposes only and has no development costs. In addition, there are no ongoing costs such as maintenance fees. Therefore, it is economically viable.

### 3.3.4 Scheduling Feasibility

During the early stages of the project, a workplan was developed. Since creation, it has been updated accordingly with the release of new deadlines. Following this plan should enable key objectives to be completed before the deadline. This is explored in further detail in the Planning section below.

## 3.4 Planning

This section details the planning to ensure all necessary milestones are met so that the project's aim is fulfilled. This consists of a workplan and risk identification.

### 3.4.1 Workplan

The project timetable is illustrated below in Figure 14 as a Gantt chart. Deliverables are categorised into stages "Concept and Research", "Implementation", "Testing", "Evaluation and Closure" and are colour coded accordingly. Each deliverable has an estimated start date, end date and ideal estimated length of completion represented by the size and position of the bar. In addition, an orange error bar indicates the acceptable margin of freedom. If the deliverable is not completed within this time, this may be detrimental to the overall completion of the project.

Key milestones are of high priority and are represented by grey diamonds along with their dates. These include submission deadlines and target completion dates for stages. Deliverables marked with an asterisk have a low priority and could potentially be dropped as further explained in the risk section.

Dependencies are represented by dashed arrows between deliverables. A deliverable with many arrows is of high priority as this means there are other tasks reliant on prior progress before they can begin. For example, report write-up must start well in advance of draft report submission so that the report can be reviewed by the supervisor before the final report submission deadline.

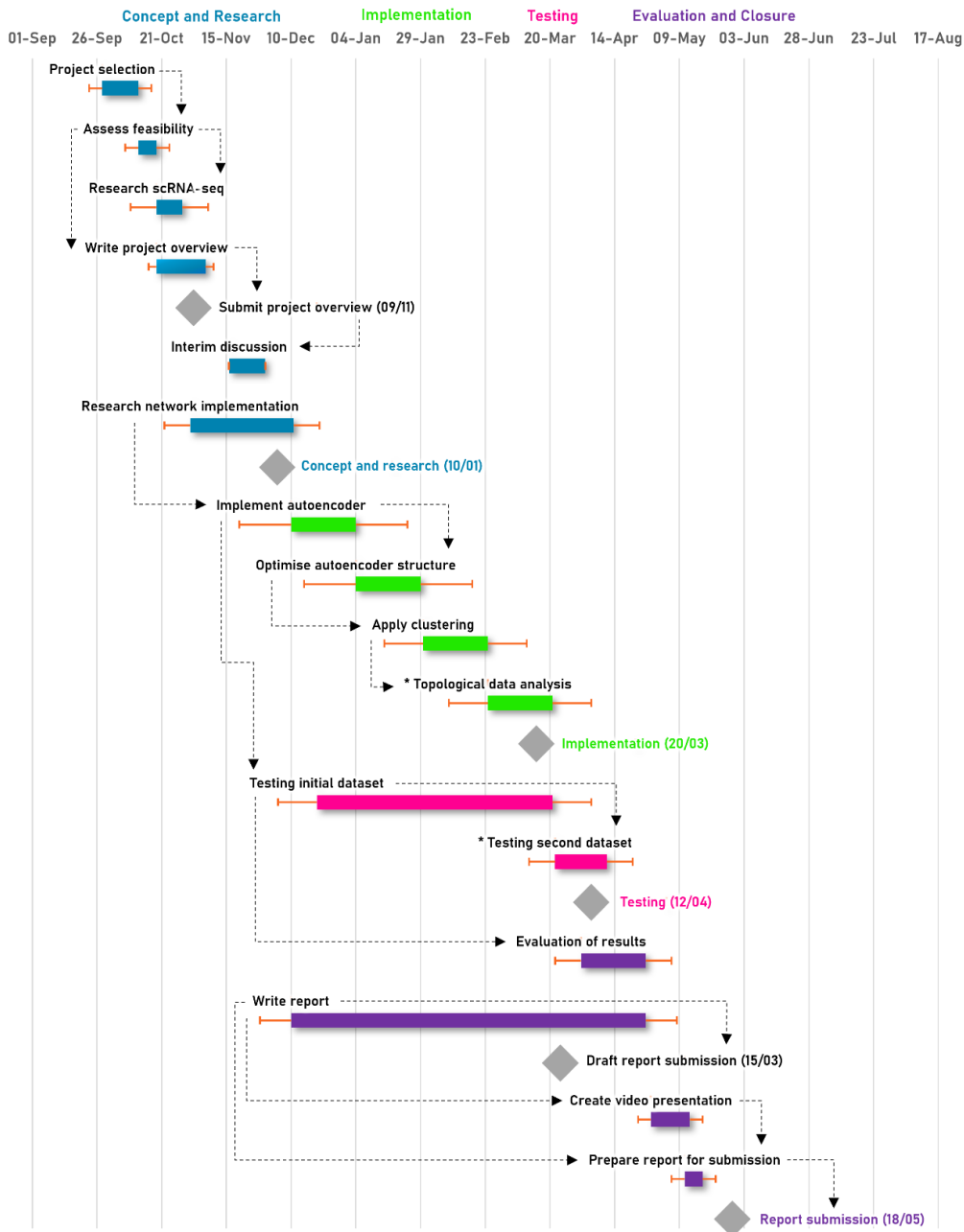


Figure 14: Project timetable

### 3.4.2 Risk Assessment

While developing the workplan, it was difficult to give an accurate estimate for the expected duration of each task as there are many factors that could influence completion. Known factors are listed in the table below. Each has a description, estimated likelihood, estimated impact, severity based upon likelihood and impact, and solution(s) to mitigate the risk.

ID	Risk	Affected Milestone(s)	Likelihood	Impact	Severity	Mitigating Action(s)
1	Project is not feasible	Concept and Research	Low	High	Medium	Assess feasibility with sponsor at start. Change project complexity to make project feasible.
2	Falling behind on timeline due to unknown other course deadlines	Any	High	Medium	High	Assigning additional time each week to work on project. Re-evaluate plan once semester 2 dates are known. In worst case, low priority

	and commitments					deliverables will not be implemented.
3	Error in estimated task completion time	Any	Medium	Medium	Medium	Overlapping bars and acceptable error in Gantt chart provide leeway for overrunning tasks. In worst case, low priority deliverables will not be implemented.
4	Illness	Any	Low	Medium	Low	Speak to supervisor if work is affected. In worst case, low priority deliverables will not be implemented.
5	Lack of clarity on the next stage project	Any	Low	Medium	Medium	Regular meetings with supervisor to discuss progress. Re-estimate remaining tasks if necessary.
6	Supervisor unavailability	Any	Low	Low	Low	Arrange regular meetings with supervisor in advance. Work on another part of the project if unsure about a task.
7	Underestimating difficulty of implementation	Implementation and Testing	Medium	High	High	Low priority deliverables will not be implemented. Assign additional time to work on project.
8	Unfamiliarity with programming framework	Implementation	Medium	Medium	Medium	Read documentation and tutorials.
9	Large dataset taking a long time to run	Implementation and Testing	Medium	Medium	Medium	Reduce dataset size during testing and / or training of models.
10	Data loss	Any	Low	High	Medium	Regularly save work and store backups.
11	Internet connection problems	Any	Low	Low	Low	Work on another part of the project, such as report write up, until connection is restored.

Table 6: Risk identification and mitigation



## 4. System and Experiment Design

System and Experiment Design outlines the plan for the initial system to be implemented, design challenges and a table of planned experiments. This is essential to meet objectives 3 – 6.

### 4.1 System Design

This part describes the design for the autoencoder, clustering algorithms, topological data analysis and creation of simulated data.

#### 4.1.1 Autoencoder

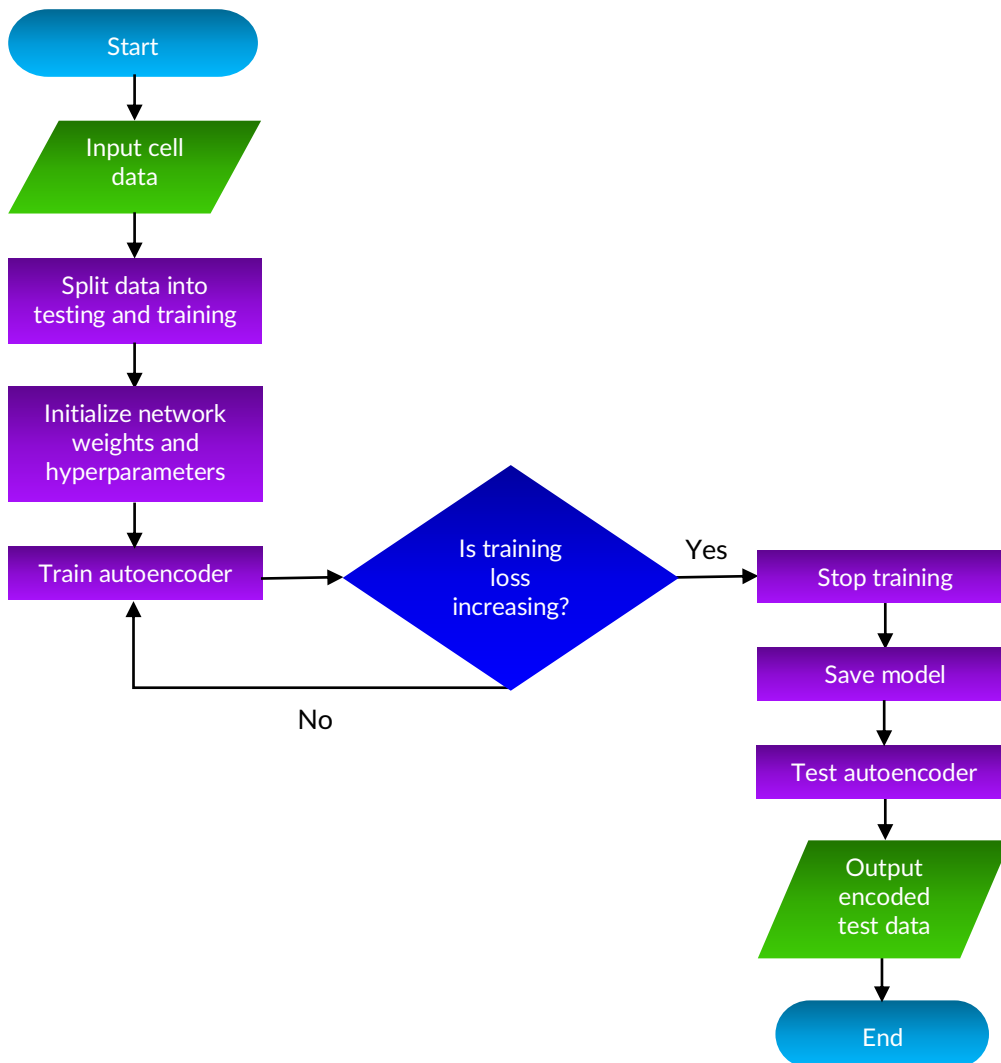


Figure 15: Flowchart for training and testing autoencoder

Figure 15 gives an overview of how the autoencoder will learn to produce an encoded output. First the inputted dataset will be divided into testing and training data. Then the autoencoder's parameters will be set and it will be trained on the training data. Once training is complete, the autoencoder's state will be saved, i.e., the model's weights. This model will then be evaluated against the testing data and the encoding for each sample outputted.

##### 4.1.1.1 Network Architecture

The design for the autoencoder's initial structure is shown below in Figure 16. The encoder will have  $N$  inputs, where  $N$  is the number of features for each sample. In a dataset, this is the number of genes per cell so the value has not been specified as it can vary.

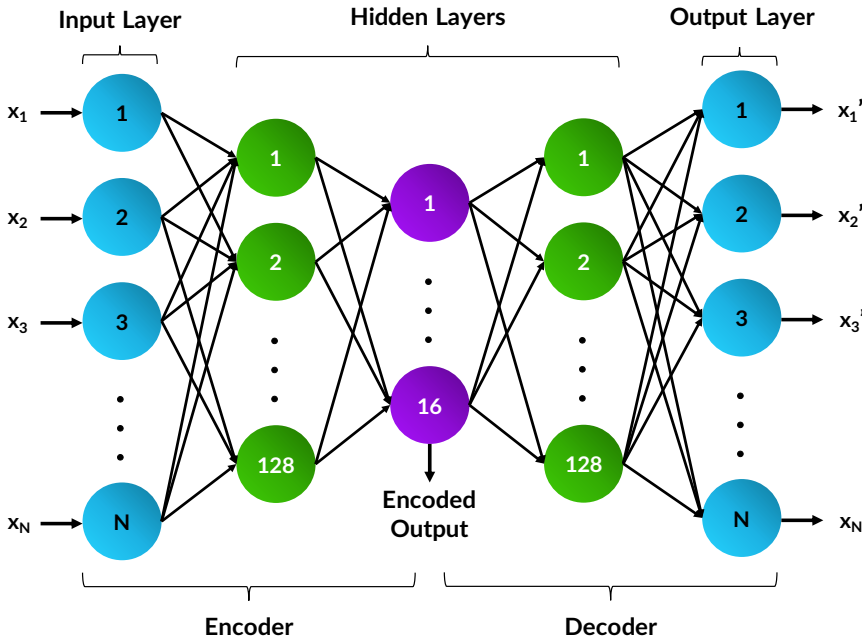


Figure 16: Autoencoder architecture diagram

The decoder will have  $N$  outputs as the goal is to find an  $x'_k$  for each  $x_k$  ( $k = 1, 2, \dots, N$ ) such that  $x_k \approx x'_k$ . In other words, each input feature  $x_k$  will be reconstructed with minimum error. To achieve this, the autoencoder attempts to learn an identity function as shown in Equation 2.

$$h_{w,w'}(x) = x$$

Equation 2: Autoencoder identity function [57]

Where:

- ◆  $w$  is the weights of the encoder
- ◆  $w'$  is the weights of the decoder
- ◆  $x$  is the vector of all inputs

A study by Geddes et al. [35] proposed that their autoencoder achieved the highest accuracy using 128 hidden features, 16 encoded features and a learning rate of 0.001. Therefore, these hyperparameters have been selected as baselines to initialise the model. Additionally, all layers will be fully connected. The ReLU activation function shall be applied to hidden layers, while linear will be used for the input, output, and encoded layer. The network will be trained through gradient descent.

#### 4.1.1.2 Optimiser

The Adam optimiser is an extension of stochastic gradient descent. It has been selected as it is computationally efficient, requires little memory and is good with noisy datasets [58]. Unlike SGD, Adam uses momentum and adaptive learning rate. This can be seen in Adam's weight update rule as shown by Equation 3.

$$w_t = w_{t-1} - \eta \frac{\hat{m}_t}{\sqrt{\hat{v}_t + \epsilon}}$$

Equation 3: Adam weight update [59]

Where:

- ◆  $w_t$  and  $w_{t-1}$  are the model weights of time instances  $t$  and  $t-1$  respectively
- ◆  $\eta$  is the learning rate
- ◆  $\hat{m}_t$  is the bias-corrected first moment estimate
- ◆  $\hat{v}_t$  is the bias-corrected second raw moment estimate
- ◆  $\epsilon$  is a small term preventing division by zero

Adam is a descendent of Adadelta, which was created as an extension of Adagrad optimiser. A more recent variant is AdamW, which aims to fix weight decay in Adam [60]. These variants will be tested along with two other popular optimisers, Rprop and SGD to see if they can reduce loss.

#### 4.1.1.3 Loss Function

Mean squared error (MSE), alternatively known as squared L2 norm, is frequently used for optimisation problems. It is more suitable than alternatives, such as mean absolute error (MAE), for data with many outliers [61] and therefore, has been chosen as the loss function. MSE calculates the average squared difference between each output's actual value and predicted value. As a result, values lie within the range  $[0, \infty]$ . For the autoencoder, the predicted value is input  $x_k$  and its actual output is  $x'_k$  as shown in Equation 4. This hyperparameter must not be changed throughout testing and evaluation so that loss is calculated consistently.

$$MSE = \frac{1}{n} \sum_{k=1}^n (x_k - x'_k)^2$$

Equation 4: Mean squared error [62]

Where:

- ◆  $n$  is the batch size
- ◆  $x_k$  is the  $k^{\text{th}}$  input
- ◆  $x'_k$  is the  $k^{\text{th}}$  output
- ◆  $\sum_{k=1}^n y_k$  is the sum of all  $y$  for  $k = 1, 2, \dots, n$
- ◆  $(x_k - x'_k)^2$  is the difference between  $k^{\text{th}}$  input and  $k^{\text{th}}$  output squared

#### 4.1.1.4 Activation Function

Activation functions are applied to the output of a neuron. If a threshold value is reached then the neuron fires and it is said to have been activated. While if the activation function of a neuron is not specified, it is linear. This is the function  $f(x) = cx$ , in which  $x$  is the neuron's output and  $c$  is a constant. This means that activation is directly proportionate to input.

If a network only uses linear activation functions, this can cause an issue because the gradient is always constant with no relationship to  $x$ . Any changes made by backpropagation will not depend on the change of input [63]. Therefore it is good practise to use both linear and non-linear functions (see Figure 17). ReLU is recommended when the shape of the function you wish to approximate is unknown [63]. For this reason, it has been selected as hidden layer activation function for the autoencoder. However, other activation functions will be tested during Testing and Validation.

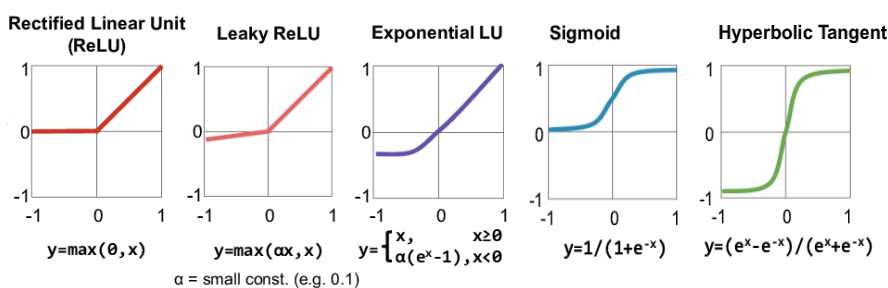


Figure 17: Non-linear activation functions [107]

#### 4.1.1.5 K-Fold Cross Validation

Cross validation is a statistical method to approximate performance of a model on unseen data. This is useful to improve accuracy of a model with few training samples [64]. For this project, it will be used to split data into training and validation sets.

First data is shuffled, then split into  $k$  groups. Over  $k$  iterations, each group is held out as the validation set, while remaining groups are training sets. The model is trained and evaluated on the validation data. The best dataset split is then selected to train the final model.

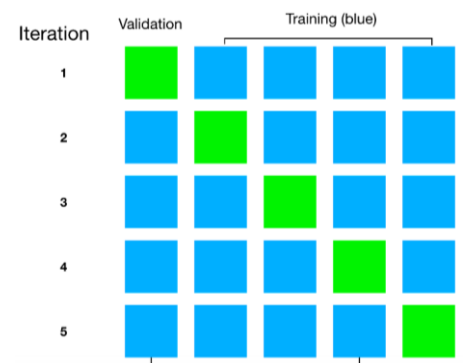


Figure 18: 5-fold cross validation [108]

### 4.1.2 Clustering

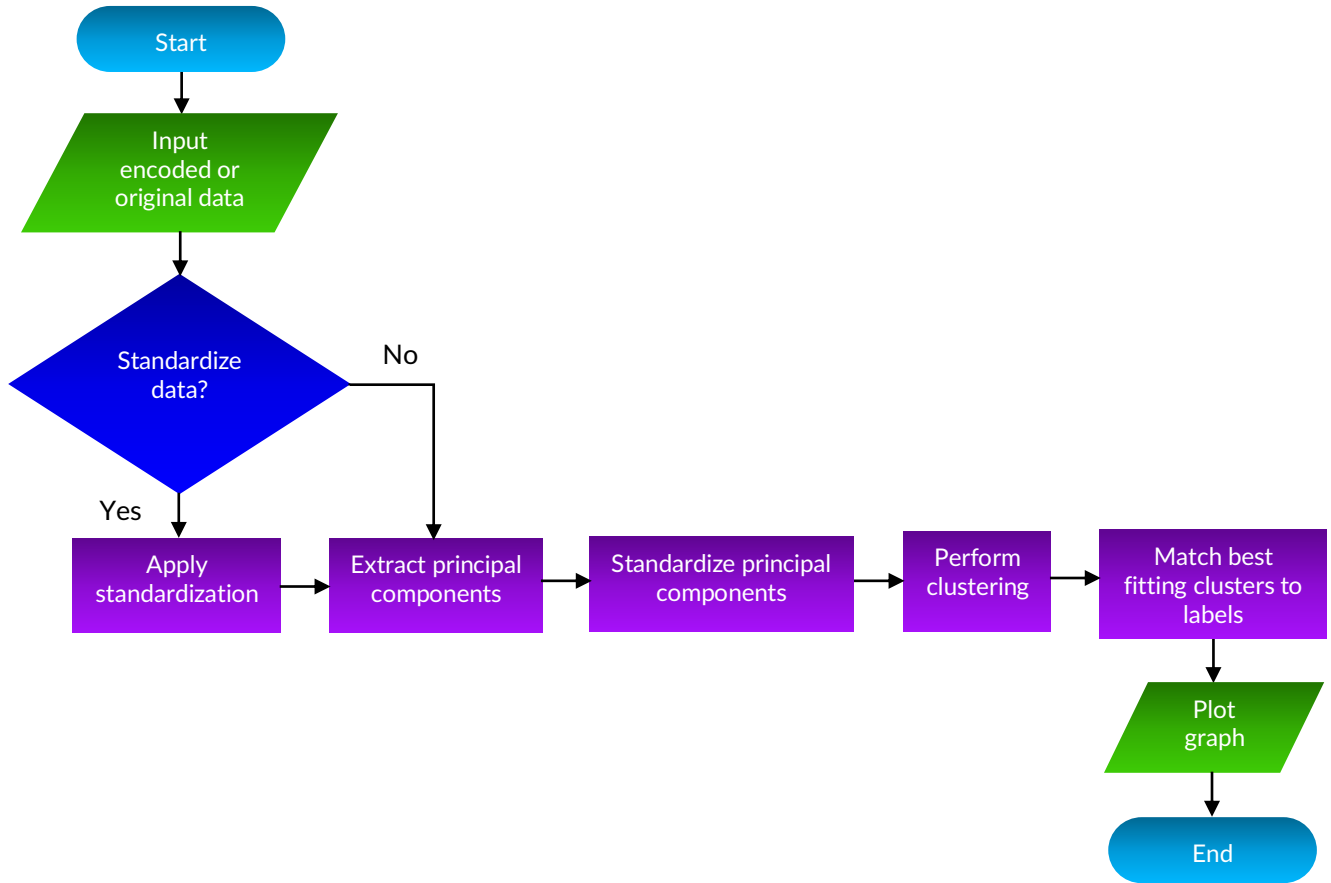


Figure 19: Flowchart of clustering steps

The figure above displays the steps required to cluster the encoding or the original gene count data. Before a clustering algorithm is applied, standardization can be optionally used before running PCA to extract the top components. After the principal components will be standardized. This process, shown by Equation 5, shifts and scales features so that they have a mean of 0 and standard deviation 1. Feature scaling is useful for algorithms such as k-means to ensure that all features are within the same range when calculating distance [65]. Once clusters have been assigned, data points will be plotted, and colour coded accordingly.

$$X_{\text{stand}} = \frac{x - \text{mean}(x)}{\text{standard deviation}(x)}$$

Equation 5: Standardization or Z-score normalisation [65]

#### 4.1.2.1 K-Means

Assigning a cluster to a data point is decided by computing the sum of the squared error (SSE) displayed in Equation 6. This is the sum of the squared Euclidean distance between a data point and its nearest centroid. The aim of the k-means algorithm is to minimise this error [36]. Due to the non-deterministic nature of k-means, often the algorithm is run multiple times on the same input and the cluster assignments with the lowest SSE are selected [36].

$$SSE = \sum_{i=1}^k \sum_{x \in C_i} d^2(x, m_i)$$

Equation 6: Sum of Squared Error (SSE) [66]

Where:

- ◆  $x$  is a datapoint in cluster  $C_i$
- ◆  $m_i$  is the representative point for  $C_i$ , i.e., the centre of the cluster
- ◆  $d$  is the Euclidean distance between  $x$  and  $m_i$

The elbow method and the silhouette coefficient are two popular methods to find the optimal number of clusters  $k$ . To implement the elbow method, k-means is run for several values of  $k$  and plotted against SSE. The

point at which the curve starts to bend down is known as the elbow point. This is the best trade-off between error and number of clusters [36]. For example, in Figure 20, this value is 3. The silhouette coefficient studies the clarity of clusters and the distance between them. It has a range of [-1, 1] and the average is plotted against a range of k values. The best k is the one that scores the highest [36].

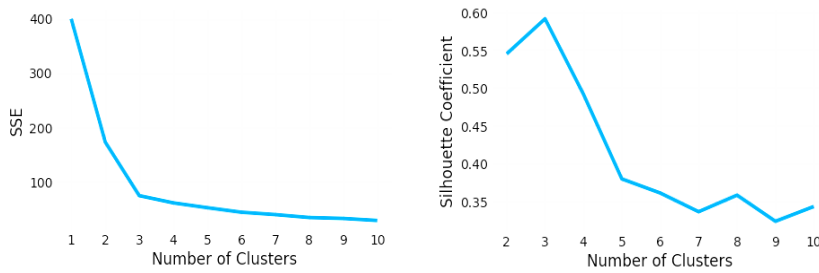


Figure 20: Elbow method (left) and silhouette coefficient (right) [36]

#### 4.1.2.1 Measuring Cluster Accuracy

The metadata of each dataset contains the cell line or group of each cell. These can be converted to numeric representations and used as labels to determine a sample's true cluster. However, as k-means assigns an arbitrary value between 0 and  $k - 1$  to each cluster, predicted and actual clusters cannot be directly compared. This means a standard accuracy score will not be reliable as the true and predicted labels may not match even if all clusters are the same.

To overcome this problem, it is necessary to figure out which mapping between predicted and actual labels produce the highest accuracy. A bipartite graph is a type of graph in which each edge connects vertices from two sets [67] and so one method is to construct a bipartite graph between predicted and actual labels (Figure 21) [68]. The graph can then be solved using the Hungarian matching algorithm to find the maximal-weight matchings [68]. This is the mapping between predicted and actual labels and once obtained, accuracy can be calculated.

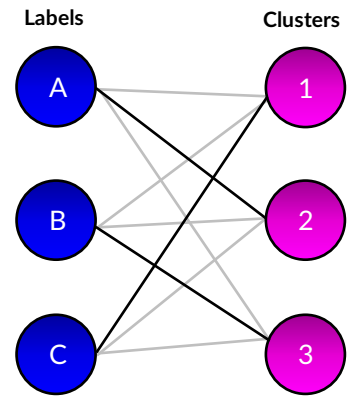


Figure 21: Bipartite graph matching labels to predicted clusters

#### 4.1.2.2 Adjusted Rand Index (ARI)

The Rand index (Equation 7) is another method to measure the similarity between a set of clusters and their true labels. It returns a value between 0 to 1 in which 0 means that no points in the clusters match their true label, while 1 indicates they all match [69].

$$RI = \frac{TP + TN}{TP + FP + FN + TN}$$

Equation 7: Rand index [69]

Where:

- ◆ TP is the number of true positives
- ◆ TN is the number of true negatives
- ◆ FP is the number of false positives
- ◆ FN is the number of false negatives

The adjusted Rand index (Equation 8) is the RI score adjusted for chance. This is used because the number of clusters or size distribution between them can vary depending on the clustering technique [69]. Unlike the RI, it is possible for ARI to return a negative value.

$$ARI = \frac{RI - \text{Expected RI}}{\text{Maximum RI} - \text{Expected RI}}$$

Equation 8: Adjusted Rand index [70]

For the benchmarking dataset, scores from experiments using many different clustering evaluation methods are available. I decided upon ARI as it allows me to quantitatively compare the results of this project with the results of others. However, it is worth mentioning that the benchmarking set contains the results from many different

clustering algorithms, normalisation techniques and imputation methods which I will not be able to test due to the limited scope of this project.

### 4.1.3 Topological Data Analysis

#### 4.1.2.3 Mapper

The Mapper algorithm is a TDA technique that combines dimensionality reduction, clustering, and graph networks to produce a simplicial complex from point cloud data [71]. Kepler Mapper [41] is a Python implementation of Mapper that will be used for this project.

The algorithm consists of the following steps:

1. Given a dataset  $X$ , project it to a lower dimensional space using a filter function  $f : X \rightarrow Z$ .
2. Create a cover  $\mathcal{U} = (U_i)_{i \in I}$  of the projected data, in which  $I$  represents the set of all indexes and  $U_i$  is an interval for index  $i$ . In Kepler, this cover contains overlapping intervals constructed from  $n$ -dimensional hypercubes.
3. For each interval  $U_i$ , apply a clustering algorithm  $C$  to points within its preimage  $f^{-1}(U_i)$  to form clusters  $C_{i,1}, \dots, C_{i,k_i}$ .
4. Initialise a graph by setting each cluster as a node. An edge is then added between each node if their clusters have points in common to create the simplicial complex.

The right of Figure 22 shows an example of a graph produced by Mapper from the 2D datapoints on the left. Height was used as the filter  $f$ , while nearest neighbours was used to cluster the cover  $\mathcal{U}$ . For this project, I plan on using PCA and k-means instead.

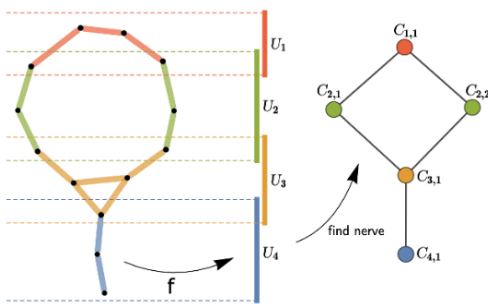


Figure 22: Mapper applied to 2D data

### 4.1.4 Creating Simulated Data

#### 4.1.2.4 Splatter

Simulated data will be generated through Splatter using the Splat simulation model. The aim is to create mock data that imitates real biological data, such as the benchmarking dataset, to test the implemented system. This means that the benchmarking dataset will be inputted to seed the simulation model.

First the Splat model simulates mean expression levels from all genes in the dataset using a gamma distribution with shape  $\alpha$  and rate  $\beta$ . The probability that a gene is an outlier is given by  $\pi^0$ . As the gamma distribution is unable to capture genes with extreme expression, outliers must be created by multiplication with a log-normal factor. Next, the means are adjusted using the Biological Coefficient of Variation (BCV). Finally, a new matrix of cells and gene counts is produced from a Poisson distribution [72].

Splat has four variations:

- ◆ **Single:** creates a single population of cells
- ◆ **Groups:** produces a population of mixed cell types
- ◆ **Paths:** generates a population in which one cell type is differentiating into another
- ◆ **Batches:** simulate technical variation that may result when samples are processed at the same time due to flaws using different equipment

For this project, I will be generating groups (see Figure 23) as I aim to investigate variation within gene expression. Additionally, it should be noted when generating the simulated data, the number of gene counts per cell must remain the same as the original, though the number of samples may vary.

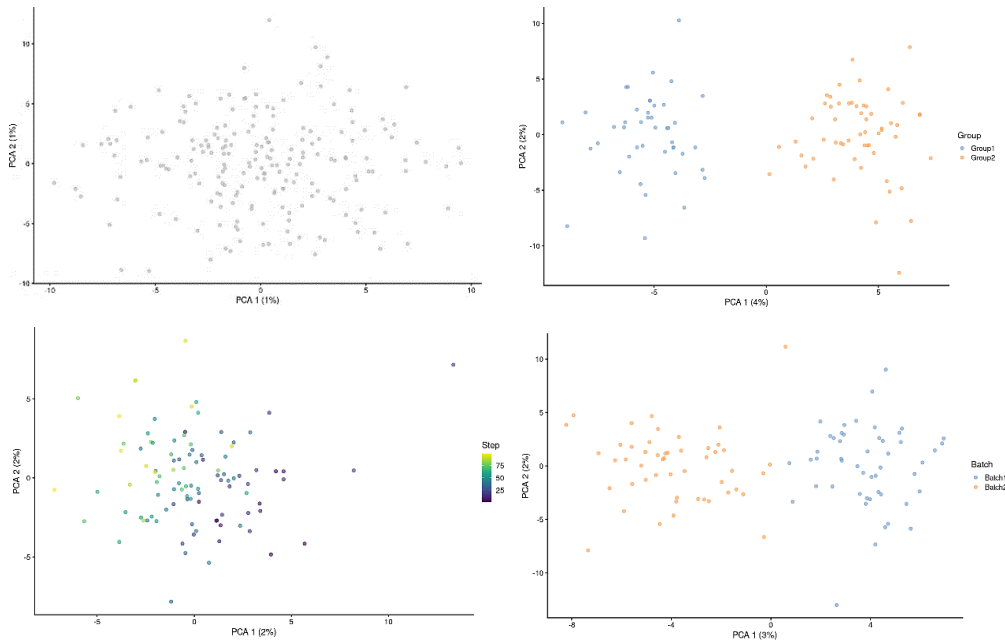


Figure 23: Top left single, top right groups, lower left paths and lower right batches [72]

## 4.1 Experiment Design

Planned experiments are listed in the table below. These will be carried out during and after development and will be documented in Testing and Validation.

ID	Experiment Description	Parameters	Outcome
1.1	Test the autoencoder's performance for the benchmarking dataset over 40 epochs using the parameters outlined in the System Design section.	<ul style="list-style-type: none"> <li>Optimiser = Adam</li> <li>Learning rate = 0.001</li> <li>Loss function = MSE</li> <li>Batch size = 32</li> <li>Activation = ReLU</li> <li>Epochs = 40</li> </ul>	Record the model's initial training and testing accuracy.
1.2	Test autoencoder performance across different learning rates.	Five different learning rates between 0.1 and 0.00001	Record training and testing loss, determine which value performed best and set this as the autoencoder's new LR.
1.3	Test autoencoder accuracy and training time for different batch sizes.	10 different batch sizes between 1 and the number of training samples	Decide upon the most suitable batch size in terms of loss and run time and set this for the autoencoder.
1.4	Test autoencoder's loss for six different optimisers.	<ul style="list-style-type: none"> <li>Rprop</li> <li>SGD</li> <li>Adadelta</li> <li>AdaGrad</li> <li>Adam</li> <li>AdamW</li> </ul>	Deduce the optimiser that achieves the lowest loss and set this for the autoencoder.
1.5	Test autoencoder's accuracy and run time when increasing or decreasing the number of hidden layers.	Set number of hidden layers to 1, 3 and 5 respectively	Determine the best number of hidden layers with regards to time and performance and set this for the autoencoder.
1.6	Test autoencoder performance for seven different activation functions.	<ul style="list-style-type: none"> <li>Linear</li> <li>ReLU</li> <li>Leaky ReLU</li> <li>PReLU</li> <li>ELU</li> <li>Sigmoid</li> </ul>	Document testing and training loss for each and select the best for the autoencoder.

		♦ Tanh	
1.7	Run autoencoder training over many epochs and determine when the model starts overfitting.	Epochs = 400	Produce a graph of training and validation loss and decide the optimal number of epochs.
2.1	Use 5-fold cross validation to split data into training and validation.	♦ k = 5 ♦ Epochs = 20	The split with the lowest validation loss will be selected.
2.2	Train autoencoders on each dataset.	Set autoencoder to use optimised hyperparameters found in prior experiments	Record training and testing accuracy and save each model's weights to file.
3.1.1	Use the Elbow method and silhouette coefficient to find the best number of clusters for k-means.	Set number of clusters between 2 to 10	Plot graphs for each, establish the best k and set this as the number of clusters for k-means.
3.1.2	Perform k-means clustering on each dataset.	♦ Encoded data ♦ Original data ♦ Pre-standardized encoded data ♦ Pre-standardized original data	Produce graphs of clusters, calculate accuracy and ARI and document in table. Conclude the best result.
3.2.1	Plot a dendrogram to determine the best number of clusters for agglomerative hierarchical clustering.	Ward linkage criterion	Document dendrogram and conclude the best number of clusters for hierarchical clustering.
3.2.2	Perform agglomerative hierarchical clustering on each dataset.	♦ Encoded data ♦ Original data ♦ Pre-standardized encoded data ♦ Pre-standardized original data	Create graphs of clusters, calculate accuracy and ARI and document in table. Conclude the best result and compare against k-means.
4.1	Run Kepler Mapper to produce a simplicial complex for each dataset.	♦ PCA with 2 principal components ♦ K-means clustering algorithm ♦ $\epsilon = 0.15$ ♦ 10 hypercubes	Record the graphs, write a conclusion and compare against clustering.

Table 7: Planned experiments



## 5. Implementation

In this part development of the Jupyter notebooks is explained. This is a necessary requirement to meet objectives 3 - 6. Although most of the code has been omitted, it is available in notebooks. The last section, Creating Simulated Data, explains Splat\_Simulator.ipynb, while all other parts detail Clustering\_and\_TDA.ipynb.

### 5.1 Setting up the Datasets

#### 5.1.1 Opening the Data

In the main notebook, each dataset and its metadata are automatically downloaded from URLs in either CSV or text file format and converted to a DataFrame using pandas. As the simulated dataset has been created for this project, it is hosted from my GitHub. For the benchmarking and simulated datasets, the gene counts and metadata are already separated. However, for the evaluation data, pre-processing is required to extract them from the tab-delimited text file.

The code will only perform experiments on one dataset, specified by setting the variable `dataset` as either `benchmark_dataset`, `sim_dataset`, or `eval_dataset`. The `metadata` variable will be set automatically as the corresponding cell line or group labels if using one of these datasets. The selected data is then split into training data (`x_train`) and testing data (`x_test`). The number of cell samples in `x_test` can be adjusted by changing `test_size`, although this has been set at 150 for the benchmarking (Figure 24) and mouse cortex evaluation data, and 500 for the simulated data.

	CELL_000572	CELL_000233	CELL_000098	CELL_000200	CELL_000677	CELL_000147	CELL_000246	CELL_000384	CELL_000481	CELL_000842	CELL_000850
ENSG00000272758	0	1	1	0	1	0	2	1	0	2	0
ENSG00000154678	0	0	0	2	0	2	2	0	0	0	0
ENSG00000148737	1	4	4	1	2	4	2	3	0	0	1
ENSG00000196968	0	2	1	1	2	0	0	3	0	1	2
ENSG00000134297	2	1	1	1	0	0	1	1	0	0	1
...	...	...	...	...	...	...	...	...	...	...	...
ENSG00000237289	0	0	2	0	0	0	0	0	0	0	0
ENSG00000238098	0	3	0	0	0	0	0	0	0	0	0
ENSG00000133433	1	0	1	0	1	0	0	0	0	0	0
ENSG00000054219	0	0	0	0	0	0	0	0	0	0	0
ENSG00000137691	0	0	2	3	0	2	6	0	0	0	0

16468 rows × 150 columns

Figure 24: DataFrame showing 150 test samples from the benchmarking dataset

As the selected datasets use rows for genes and columns for cells, it is necessary to transpose `cell_data`. This prevents the genes incorrectly being treated as samples instead of the cells. If a new dataset is used, then it is important to check that samples have been correctly identified as the notebook may still run but will produce unexpected results.

#### 5.1.2 Splitting Data into Batches

Before testing and training data can be passed into the `LitAutoencoder`, detailed in a later section, it must first be converted to a custom PyTorch Dataset. The implemented class `DatasetRNASeq` is known as a map-style dataset as it provides a mapping between indices and samples [73].

A `DataLoader` is an object that iterates over a dataset and allows samples to be split into batches [74]. After changing training and testing data into `DatasetRNASeq`, they are then converted to dataloaders. Batch size was initially set to 32 but can be changed by updating the `batch_size` global variable.

### 5.2 Autoencoder

#### 5.2.1 LitAutoencoder Class

The autoencoder was implemented as a PyTorch Lightning class `LitAutoencoder`. This class takes input three dataloaders and the number of input / output features. Other hyperparameters can be set by changing the keyword arguments `autoencoder_kwargs`. These include: the number of hidden features and layers, loss function, optimiser, learning rate and activation function. Once an autoencoder has been initialised, the

hyperparameters are saved temporarily to a YAML file so that the model state that reaches the lowest validation loss can be loaded again from a checkpoint.

Within the model, the encoder and decoder are separated as `Sequential` containers consisting of `Linear` layer(s) and activation function(s), though the exact structure depends on the hyperparameters. As well as the model layers, the class also encapsulates the training, validation and testing steps and logs the results to TensorBoard.

### 5.2.2 K-Fold Cross Validation

Before fully training the model, k-fold cross validation is used to split the data assigned for training into train and validation sets. The autoencoder is run over a low number of epochs, such as 20, and the final validation loss of each fold is recorded. Once all folds have been tested, the training and validation data that achieves the lowest loss is selected and converted to dataloaders.

### 5.2.3 Autoencoder Training

When training a new model, PyTorch Lightning requires the model to be fit to a `Trainer`. This class puts the model into training mode, iterates through the batches in the train data loader, calculates the loss and performs backpropagation to update the model's weights [75]. Additionally, the `Trainer` will place the model onto GPU if available as this can decrease training time considerably.

The PyTorch learning rate finder can optionally be used before training. This will test a range of learning rates and then set the best as the learning rate of the optimiser. However, I did not find this improved performance as explained during Testing and Validation.

### 5.2.4 Saving Model State to File

Once an autoencoder has been trained, it can be saved to file by setting `download_model` to `True`. This will save lists of the selected for train, test and validation to text files and the best model state is saved as a checkpoint file. These files may then be reloaded to restore the model's state and continue training if required. Saving the model was useful for this project as it prevented me from having to train a new model each time and allows the results of the experiments to be reproducible.

#### 5.2.4.1 Solving the File Size Problem

While developing the notebook, I initially saved and loaded the train, test and validation dataloaders and checkpoint file from Google Drive to restore the states of the trained model. Although this was fine for performing experiments, it became a problem when I tried to upload these files to GitHub as most of the files (see Figure 25) exceeded the limit of 25,000 KB.

Name	Date modified	Type	Size
autoencoder-epoch=247-val_loss=56.86...	26/02/2021 17:47	CKPT File	49,659 KB
test_dataloader.pth	26/02/2021 17:42	PTH File	19,722 KB
train_dataloader.pth	26/02/2021 17:44	PTH File	97,257 KB
validation_dataloader.pth	26/02/2021 17:44	PTH File	97,253 KB

Figure 25: Checkpoint (CKPT) and test, train and validation DataLoader (PTH) files for the benchmarking autoencoder

After some research, I discovered the size of checkpoint files could be reduced by saving only the model's weights to the `ModelCheckpoint` rather than the entire model state [76].

```
1 checkpoint_callback = ModelCheckpoint(monitor='val_loss', mode='min',
2                                     save_weights_only = True,
3                                     filename='autoencoder-{epoch:02d}-{val_loss:.2f}')
```

However, the models states had already been saved, and so I was required to write a custom function `convert_to_weights_only` to remove the optimiser state, learning rate scheduler information and callbacks. While removing this information would impact the autoencoder if it were trained further as the optimiser's state would be reset, this was not an issue as I had completed training and only need to restore the model's best state to encode the data.

```

1. def convert_to_weights_only(checkpoint_file, new_file_name="checkpoint_weights.ckpt"):
2.     key_to_remove = {'optimizer_states', 'lr_schedulers', 'callbacks'}
3.     # Load the checkpoint from file
4.     checkpoint_dict = torch.load(checkpoint_file, map_location=torch.device('cpu'))
5.     # Remove the keys not found in save_weights_only
6.     for key in key_to_remove:
7.         checkpoint_dict.pop(key, None)
8.     # Save the checkpoint
9.     if not new_file_name.endswith('.ckpt'):
10.         new_file_name = new_file_name + '.ckpt'
11.     torch.save(checkpoint_dict, new_file_name)
12.     print("Saved checkpoint to " + str(new_file_name))

```

Instead of saving dataloaders to file, I saved the names of the cells in the test, train and validation splits to text files. These can then be used to reconstruct the dataloaders and the notebook will automatically now save to text file if run again. As demonstrated in Figure 26, updating the checkpoint and dataloader files resulted in a massive reduction to the file sizes of the model state. These files were then as zipped to reduce size further and uploaded to GitHub. This means that a user can now run the notebooks using the pre-trained models without being required to upload any files.





Name	Date modified	Type	Size
 benchmark_autoencoder_weights.ckpt	05/05/2021 12:21	CKPT File	16,554 KB
 test_data	05/05/2021 12:58	Text Document	2 KB
 train_data	05/05/2021 14:09	Text Document	8 KB
 validation_data	05/05/2021 14:05	Text Document	2 KB

Figure 26: Reduced size checkpoint (CKPT) and test, train and validation tex files for the benchmarking autoencoder

## 5.2.5 Autoencoder Testing

Once an autoencoder is trained, it is evaluated on test data and the average loss displayed. Graphs of training and testing performance can be viewed using TensorBoard. Then an encoding is produced from the test data to be used during clustering.

## 5.3 Clustering

### 5.3.1 Set up Data and Labels

Standardized versions of the both the encoded and original test data are produced using scikit-learn's `StandardScaler`. Along with the unstandardized versions, this provides four variations of the data to test during the clustering experiments.

As each dataset's `metadata` contains either group or cell line labels, these are used to provide a ground truth to compare against the cluster assignment of each cell. First though, each unique label is assigned a numerical representation and a list `test_cell_labels` is produced containing the target labels of each cell in the test data as demonstrated in Figure 27.

Mapping between cell lines and numerical value:  
{'H1975': 0, 'H2228': 1, 'HCC827': 2}

Numerical values of target clusters:  
[0, 0, 0, 1, 0, 1, 1, 2, 2, 0, 0, 1, 1, 1, 0, 1, 1, 0, 1, 1, 0, 2, 1, 0, 0, 2, 1, 2, 1, 0, 2, 1, 1, 2, 1, 2, 2, 1, 1, 2, 2, 1, 1, 0, 2,

Figure 27: Print screen of assigning numerical values to the cell lines labels of the benchmarking dataset

## 5.3.2 K-Means

### 5.3.2.1 Elbow Method and Silhouette Coefficient

Before applying k-means clustering, the best number of clusters `k` is determined using the elbow method and silhouette coefficient. The custom function `elbow_method` will test `KMeans` with `k` clusters for `k` between 2 and a value `max_k` set by the user. For each attempt, the sum of the squared error is added to a list and plotted against the number of clusters, then the function `KneeLocator` is used to mark the elbow point on the graph. The function `silhouette_coefficient` is similar, though instead plots the silhouette score against cluster

number and marks the best k on the graph as the value that achieves the highest score.

### 5.3.2.2 K-Means Clustering

Before a clustering algorithm is applied, a dimensionality reduction technique is used to extract representational features from the data and then standardization is applied. Initially PCA was used, although later I extended the functionality to support Independent Component Analysis (ICA), Non-Negative Matrix Factorization (NMF) and t-SNE. Then k-means is run on a version of the data specified by the user and interactive graphs are returned.

Graphs of the predicted clusters are plotted as either a 2D or 3D scatter chart using Plotly [77]. This library was selected as it allows custom text to appear when a user hovers their cursor over a data point. For the project, this was set to display the cell names and the cell lines or groups if provided.

Additionally, if the ground truth is available, a new trace 'Actual' displaying the target labels is added to the plot (see Figure 28), as well as a trace 'Difference' that highlights the incorrect predictions in red. To calculate this difference, first a bipartite graph is constructed between the predicted clusters and their target labels. Then the function `linear_sum_assignment` is run to find the best match between the clusters and labels according to the Hungarian matching algorithm. The numbers assigned to the predicted cluster are then updated to match their most likely label and the accuracy are ARI between the two calculated.

The silhouette coefficient is also calculated, though this does not require the ground truth. Then the three metrics are printed below the graph as demonstrated in Figure 28. Optionally, a user may save the graph as an HTML file.

K-Means Clustering on 2 Components Using PCA

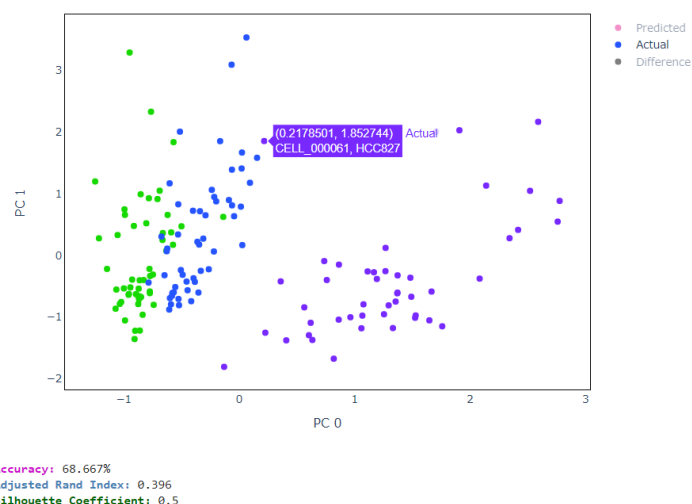


Figure 28: Example graph showing cell lines of the benchmark dataset and its accuracy, ARI and silhouette coefficient

### 5.3.2.3 Fixing the Issue of Matching Cell Names to Values

When a dataset is split for testing and training, the order of the samples is shuffled resulting in different test-train splits each time the notebook is run. When setting up clustering after loading the optimised autoencoder for the benchmarking dataset, I was running k-means on the samples in the `test_data_loader` loaded from file yet getting cells names from the columns in `DataFrame x_test` (shown before in Figure 24) to display on the graph. Although the code ran, it meant the cell names did not match their values, and this was not apparent until I was trying to match the cells to their cell line labels. This issue has since been resolved through reading the cell names from the test data saved to file when loading a pre-trained model.

### 5.3.2.4 Finding the Best Numbers of Components

After first running k-means with a set dimensionality reduction technique, such as PCA, on two and three components, a custom function `find_best_components` is used to test higher number of components against a given measure. This can be either: accuracy, ARI or silhouette coefficient. Internally this function will run the clustering algorithm multiple times to test a range of components between 2 and a set value `max_components`. The average score of the measure for each number of components is returned and displayed as a graph through function `plot_components`, and scores presented as a DataFrame (Figure 29).

The optimal number of components is then retrieved, and k-means is once again run to produce 2D and 3D graphs of the clusters with the best component number. If PCA was used for dimensionality-reduction, then additional graphs are displayed showing how much each principal component contributes to the variation of the data.

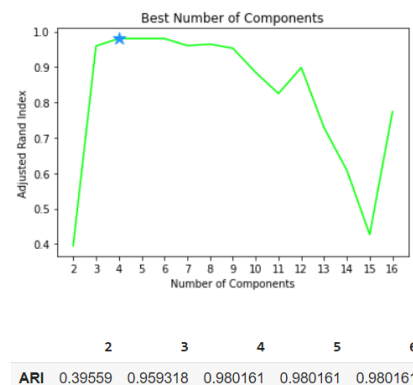


Figure 29: Example showing the effect of increasing principal components on ARI for the encoded benchmark dataset

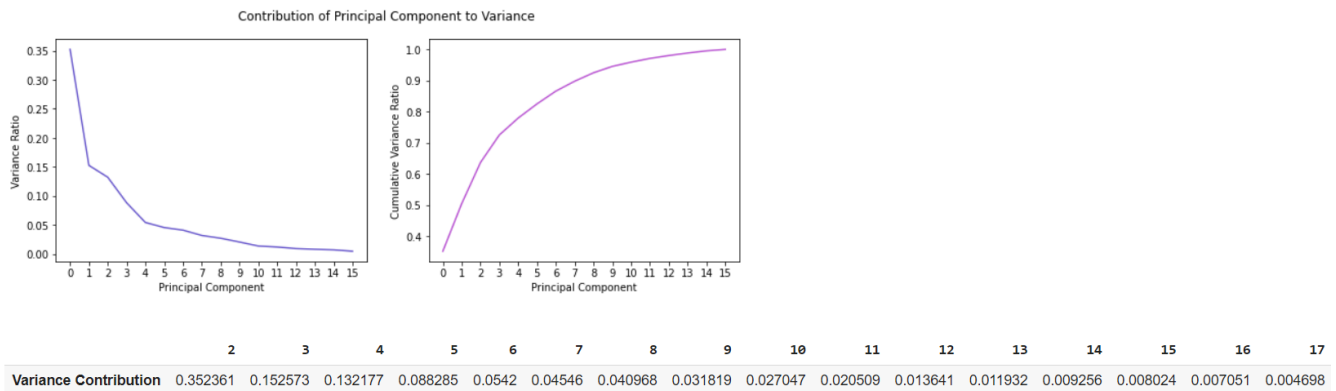


Figure 30: Example graphs revealing the gain in variation with the addition of each principal component for the encoded benchmarking data

### 5.3.3 Agglomerative Hierarchical Clustering

The graph to determine the best number of clusters is plotted using SciPy's `dendrogram` function. The clustering algorithm can be changed from k-means by setting variable `algorithm`. Then as before, clustering can be run in combination with a dimensionality reduction technique and 2D and 3D graphs displayed.

### 5.3.4 Alternative Clustering Algorithms

Along with k-means and agglomerative hierarchical, four new clustering algorithms are tested including: BIRCH, mini-batch k-means, spectral clustering, and Gaussian mixture. These are explained in more detail in Testing and Validation as they were not originally included in the project's design. A `for` loop is run that iterates through each combination of clustering algorithm and components for dimensionality reduction on both the encoded and original data with and without pre-standardization. The combination for each clustering algorithm that achieves either the highest accuracy or silhouette coefficient is then presented in a DataFrame, such as in Figure 31, and plotted as a bar chart. Then the algorithm that performs best overall is run and the clusters plotted.

	Accuracy	Number Incorrect	Components	Encoded	Standardized
K-Means	0.997783	2	3	False	True
Hierarchical	0.998891	1	3	False	True
Birch	0.998891	1	3	False	True
Mini Batch K-Means	0.997783	2	3	False	True
Spectral Clustering	0.998891	1	4	False	True
Gaussian Mixture	0.997783	2	3	False	True

Figure 31: Example of testing ICA with different numbers of components on the benchmarking data

## 5.4 Topological Data Analysis

### 5.4.1 Kepler Mapper

Kepler Mapper is run on all samples in each dataset individually. First, data is projected to a lower dimensional space using PCA with two principal components. Then the Mapper algorithm is run to produce a simplicial complex which can be downloaded as an HTML file.

#### 5.4.1.1 Setting Parameters

Some trial and error was required to find suitable values for  $\epsilon$ , known as `perc_overlap`, and the number of hypercubes `n_cubes` on the first dataset. When  $\epsilon$  was small, many of the nodes were separated, while increasing it too much resulted in most nodes being connected losing the topological features (see Figure 32). Similarly, using only a few hypercubes created a simple graph, while increasing this revealed more complexity (see Figure 33). As a compromise between the two, I selected 10 hypercubes and  $\epsilon$  as 0.15. Note clicking the figures will open hyperlinks to the graphs.



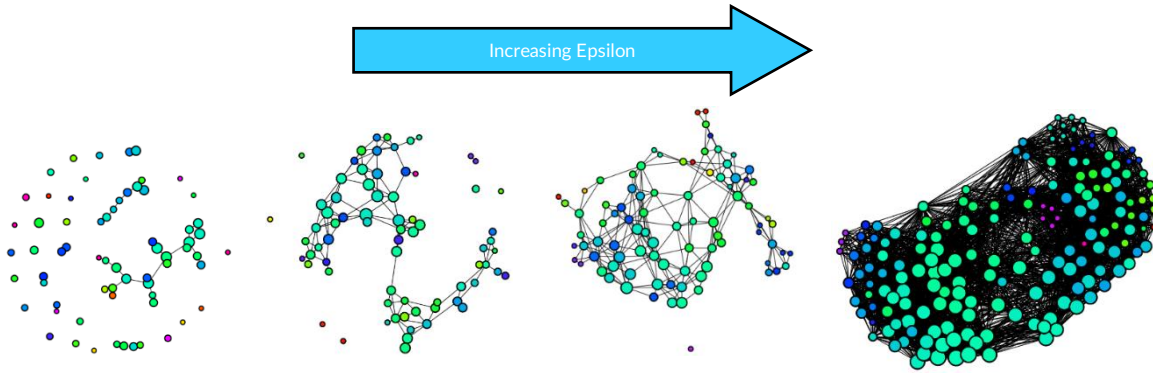


Figure 32: (Clickable) Testing epsilon values 0.05, 0.2, 0.4 and 0.8 for the benchmarking dataset

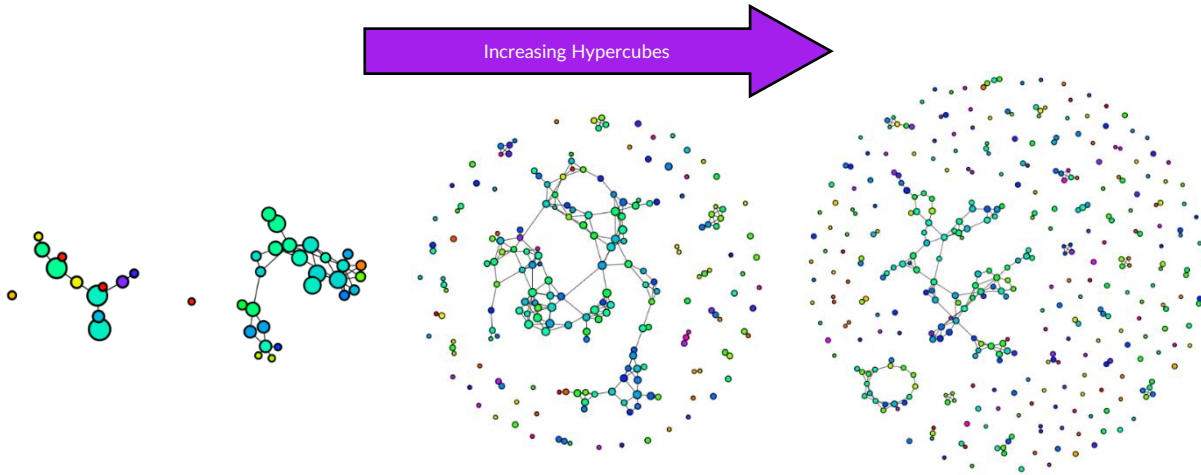


Figure 33: (Clickable) Testing 2, 5, 25 and 50 cubes for the benchmarking dataset

## 5.5 Creating Simulated Data

The simulated data was created in a separate notebook `Splat_Simulator.ipynb`. This notebook contains both Python and inline R. Although Python implementations of Bioconductor and Splatter exist, they were created as R frameworks and so I decided to use R so I could follow the official documentation.

### 5.5.1 Opening the Benchmarking Dataset

The benchmarking data is downloaded from GitHub as `dataset`, the number of features per sample is recorded as `total_genes`, while the matrix of gene reads, in which each row is a gene, and each column is a cell, is referred to as `counts`.

### 5.5.2 Creating a New Dataset

First, `splatEstimate` creates parameters mimicking the original dataset which are then used by `splatSimulate` to create new data. The counts are obtained, and the first two principal components extracted by PCA to plot the new cells (see Figure 35). As no method was given to `splatSimulate`, Splat generates single data in which all cells are of the same type. However, as I wish to investigate variation in gene expression, a second simulation is created where cells are split into four groups in a ratio of 40% : 15% : 20% : 25%. Average gene expression between the single group of simulated data (Splat\_Single), two group of simulated data

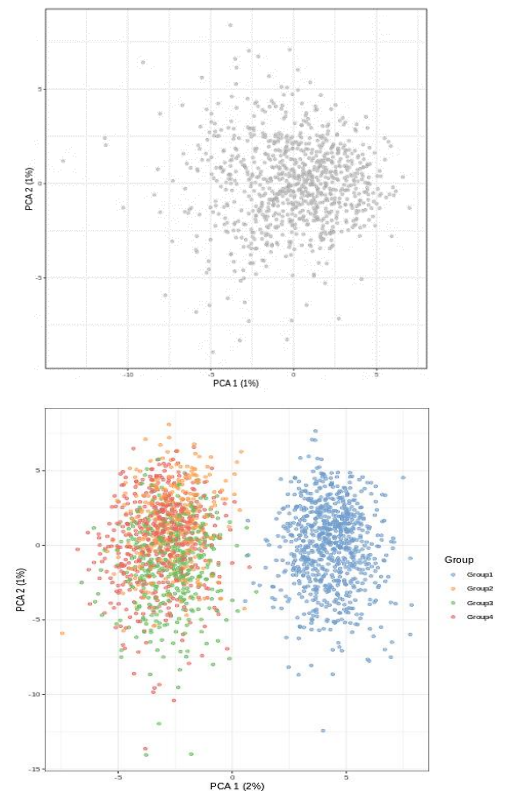


Figure 34: Single group of simulated cells (top) and four groups of simulated cells (lower)

(Splat\_Groups) and the original dataset (sc\_10x) is compared through plotting the distribution of mean gene expression (Figure 35). Although there is visible difference between the simulated data and the original, the similar shapes suggest Splat has performed well at mimicking the gene expression of the real biological data. Finally, the new dataset and its group labels were saved as CSV files and uploaded to GitHub.

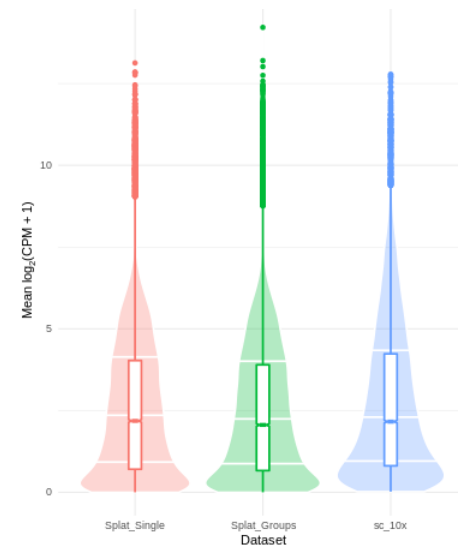


Figure 35: Comparing distribution of mean gene expression between Splat and real data

## 6. Testing and Validation

Testing and Validation explains the technical details of datasets, documents experiments and evaluates results. For a list of all planned experiments, see Experiment Design.

### 6.1 Datasets

This part explains the contents of each dataset, their dimensions, metadata and how each was used during training, testing and validation.

#### 6.1.1 Benchmarking Data

##### 6.1.1.1 Count Data

The first dataset comprises 16468 genes  $\times$  902 cells. This means there are 902 samples, each with 16468 features as shown in Figure 36. The name of each gene is an Ensembl ID, in which the ENSG tag indicates it represents a genomic region and the number is a unique identifier [78].

Benchmarking Dataset:

	CELL_000001	CELL_000002	CELL_000003	CELL_000004	CELL_000005	CELL_000006	CELL_000007	CELL_000008	CELL_000009
ENSG00000272758	0	0	0	2	0	0	0	1	1
ENSG00000154678	0	0	0	1	0	1	0	0	1
ENSG00000148737	0	0	0	1	3	2	0	2	1
ENSG00000196968	0	0	0	1	2	2	5	0	4
ENSG00000134297	0	0	0	1	1	1	2	1	3
...	...	...	...	...	...	...	...	...	...
ENSG00000237289	0	1	0	0	0	0	0	0	0
ENSG00000238098	0	0	0	0	1	0	0	0	0
ENSG00000133433	0	0	0	0	0	1	0	0	0
ENSG00000054219	1	0	0	0	0	0	0	0	0
ENSG00000137691	0	3	0	0	0	0	0	0	0

16468 rows  $\times$  902 columns

Figure 36: Print screen of sc\_10x

As explained during the Literature Review, this dataset was selected for testing during development. Therefore, Clustering and Topological Data Analysis experiments are first conducted on this data.

During Autoencoder Optimisation, the set is shuffled and split into 602 training samples, 150 validation samples and 150 testing samples, although the samples remain the same for each test. In Training the Optimised Network, the set is shuffled and initially split into 752 training and 150 testing samples. K-fold cross validation is performed to separate the training samples into 80% train and 20% validation.

##### 6.1.1.2 Metadata

In the count data above, biological variation is present because cells were cultivated from three different cell lines. As a result, metadata contains the cell line of each sample as demonstrated in Figure 37. This has been used to provide a ground truth during Clustering.

Additionally, the metadata includes the adjusted Rand index from experiments by L. Tian et al. [79]. These experiments explore applying a range of normalisation techniques, imputation methods and clustering algorithms to a lower-dimensional representation of sc\_10x extracted through t-SNE and the first two principal components of PCA [79].

The results and a graph comparing ARI for each clustering method are displayed in Figure 39 and Figure 40 respectively. As evident in the graph, some clustering algorithms over-clustered resulting in poor ARI, while those closer to the true number of clusters, represented by the dotted line, performed better. The best ARI is 0.742,

Cell Lines in Benchmarking Dataset:

	cell_line
CELL_000001	HCC827
CELL_000002	H1975
CELL_000003	HCC827
CELL_000004	HCC827
CELL_000005	HCC827
...	...
CELL_000939	H1975
CELL_000943	H1975
CELL_000946	H2228
CELL_000955	HCC827
CELL_000965	HCC827

902 rows  $\times$  1 columns

Figure 37: Print screen of benchmarking data cell lines



the worst is 0.095 while overall, there is a mean average of 0.436. These provide a second comparison to evaluate the performance of my own clustering experiments.

ARI for Clustering Experiments on Benchmarking Dataset:

	norm_method	impute_method	clustering_method	result
9701	TMM	DrImpute	SC3	0.741936
10373	scone	DrImpute	SC3	0.741936
9477	DESeq2	DrImpute	SC3	0.741291
9253	scran	DrImpute	SC3	0.741291
10157	Linnorm	SAVER	SC3	0.741144
...	...	...	...	...
4293	scone	knn_smooth2	Seurat_1.6	0.228908
4357	SCnorm	knn_smooth2	Seurat_1.6	0.226527
4165	logCPM	knn_smooth2	Seurat_1.6	0.225011
4229	Linnorm	knn_smooth2	Seurat_1.6	0.216451
10381	scone	SAVER	SC3	0.095372

127 rows × 4 columns

Figure 39: Print screen of benchmarking ARI

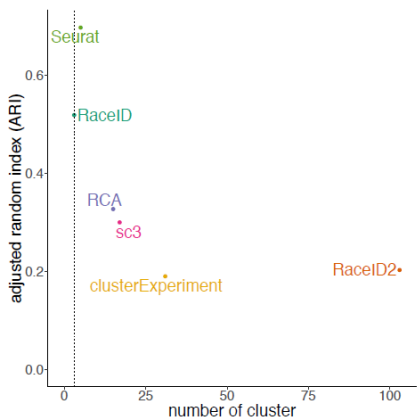


Figure 38: ARI for a range of clustering techniques [79]

## 6.1.2 Simulated Data

### 6.1.2.1 Count Data

The simulated dataset (left in Figure 41) was created to imitate the variation in gene expression of the benchmarking dataset. It contains 2000 samples, with the same number of genes as the original (16468). The creation of this data is explained under .

Splat Simulated Dataset:

	cell1	cell2	cell3	cell4	cell5	cell6	cell7	cell8	cell9	cell10	cell11	cell12	cell13
Gene1	2	6	1	7	4	2	0	1	1	2	0	4	6
Gene2	0	0	0	0	0	0	0	0	0	0	0	0	0
Gene3	2	4	2	5	0	7	0	2	0	6	0	1	2
Gene4	0	0	7	0	0	0	0	1	0	0	0	0	0
Gene5	2	1	0	0	0	0	0	0	0	0	0	1	0
...	...	...	...	...	...	...	...	...	...	...	...	...	...
Gene16464	0	0	1	0	0	0	0	0	0	0	0	0	0
Gene16465	1	4	0	0	2	6	2	9	0	8	1	14	6
Gene16466	5	12	1	8	5	7	0	7	1	7	2	3	12
Gene16467	0	5	0	0	0	1	0	0	5	2	5	0	0
Gene16468	3	0	0	0	1	2	1	0	0	0	0	0	0

16468 rows × 2000 columns

Figure 41: Print screen of simulated data

Groups in Simulated Dataset:

	Group
Cell1	Group4
Cell2	Group3
Cell3	Group1
Cell4	Group3
Cell5	Group2
...	...
Cell1996	Group3
Cell1997	Group1
Cell1998	Group1
Cell1999	Group3
Cell2000	Group4

2000 rows × 1 columns

Figure 40: Simulated data group labels

As the simulated count data has the number of dimensions as the benchmarking data, it could be directly encoded with the autoencoder created during Training the Optimised Network. However, as Splatter was not able to replicate the same high-dimensional shapes of real cell lines, this meant a new autoencoder had to be trained. The encoded data is then used during experiments in Clustering and Topological Data Analysis.

### 6.1.2.2 Metadata

Like the cell line labels in the benchmarking data, cells are split into four groups of gene expression as presented above to the right of Figure 40. As the data has been generated, this guarantees that each cell's group is correct. Again, this is explained further during Clustering.

## 6.1.3 Mouse Cortex Evaluation Data

### 6.1.3.1 Count Data

This dataset contains 1072 genes  $\times$  3005 cells as shown by the leftmost table in Figure 43. As there are three general class of RNA molecule, the names of some genes have been marked with tRNA (transfer RNA) or rRNA (ribosomal RNA). For this data, a new autoencoder had to be trained as it does not contain the same number of genes. 2004 samples were used for training, 501 for validation and 500 were set aside for testing. Again, this data was used to provide a comparison during Clustering and Topological Data Analysis.

Evaluation Dataset:

	1772071015_C02	1772071017_G12	1772071017_A05	1772071014_B06	1772067065_H06	1772071017_E02
r_HY1	0	1	0	1	2	1
r_HY3	4	0	0	1	5	0
r_LTR33A_	0	0	0	0	0	0
r_RLTR10A	0	1	0	3	0	5
r_IAPLTR1_Mm	50	25	45	30	5	62
...	...	...	...	...	...	...
r_U3	0	0	0	0	0	0
r_tRNA-Arg-CGY_	1	6	0	0	0	0
r_tRNA-Ala-GCY_	0	0	0	0	0	0
r_U4	0	7	0	0	0	0
r_tRNA-Ser-TCG	0	0	0	0	0	0

1072 rows  $\times$  3005 columns

Figure 43: Print screen of mouse cortex data

Groups in Evaluation Dataset:

	levelclass
1772071015_C02	interneurons
1772071017_G12	interneurons
1772071017_A05	interneurons
1772071014_B06	interneurons
1772067065_H06	interneurons
...	...
1772067059_B04	endothelial-mural
1772066097_D04	endothelial-mural
1772063068_D01	endothelial-mural
1772066098_A12	endothelial-mural
1772058148_F03	endothelial-mural

3005 rows  $\times$  1 columns

Figure 42: Mouse cortex data group labels

### 6.1.3.2 Metadata

Each cell has been assigned to one of nine groups, and then one of 47 sub-groups determined using the BackSPIN clustering algorithm [50]. This means that this dataset should contain a larger range of genetic variation than the previous two. For this project I selected the level 1 classes to use as targets for clustering (Figure 42). These cell type labels derived from the first group assignments, though groups 1 and 2 are assigned joint class “pyramidal CA1”, and 7 and 8 “astrocytes\_ependymal”. However, because the labels were created using an algorithm as opposed to human annotation, this ground truth may be less reliable.

## 6.2 Autoencoder Optimisation

The autoencoder’s architecture was implemented according to the design in section 4.1.1. Adam is set as the optimiser with default learning rate 1e-3, batch size is set to 32 and the ReLU activation function is applied to all hidden layers, except the encoded output layer. As stated by objective 4, these hyperparameters will be updated throughout the experiments to improve the model’s accuracy. All experiments are run on the benchmarking data across 40 epochs unless stated otherwise so that a fair comparison can be made.

### 6.2.1 Testing the Initial Model

The model was run using the hyperparameters specified by the original design. The final loss after 40 epochs for training and testing is recorded in Table 8, while testing and validation loss across epochs are presented in Figure 44.

Train MSE	Test MSE
189.7	80.3

Table 8: Loss of first run after 40 epochs

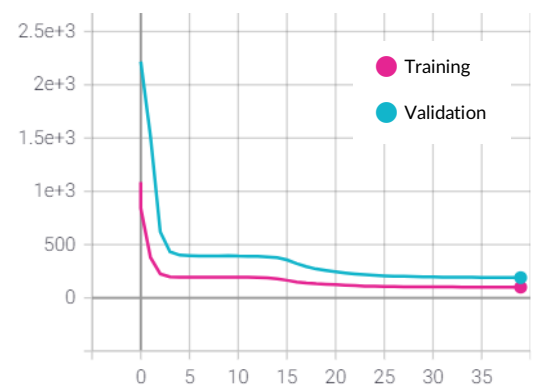


Figure 44: Loss (MSE) across 40 epochs for first run

The training loss is far higher than the testing loss implying the model is underfitting. This was because the dataset was not shuffled before splitting into training, validation, and testing. After re-arranging the columns in the dataset, the results were updated as shown in Table 9 and Figure 45.

Train MSE	Test MSE
157.5	151.6

Table 9: Loss of second run after 40 epochs

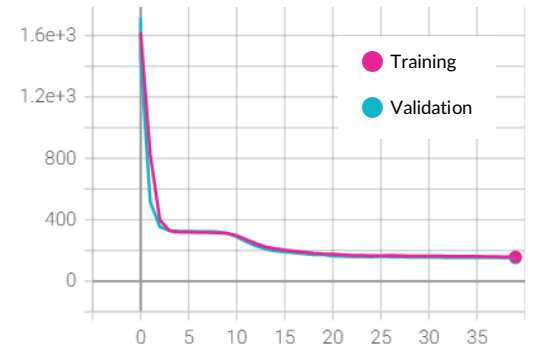


Figure 45: Loss (MSE) across 40 epochs after shuffling

Now testing and training loss is much closer indicating that the model is no longer underfitting. The final testing accuracy was 32.0%, signifying that for each sample, an average of  $16468 \times 32\% \approx 5270$  gene counts were recreated correctly. A print screen of a batch's input and recreated output can be seen in Appendix B.

## 6.2.2 Learning Rate

The network was trained separately five times with different learning rates between 0.00001 to 0.1. Each time over 40 epochs using batch size 32, Adam optimiser and 602 training samples, 150 validation samples and 150 testing samples. Loss after training and testing was recorded in the Table 10 and plotted in Figure 46.

Learning Rate	Train MSE	Test MSE
1e-1	537	603.2
1e-2	326.4	357.2
1e-3	157.7	160.4
1e-4	309.5	338.7
1e-5	775.3	819.6

Table 10: Performance for different learning rates

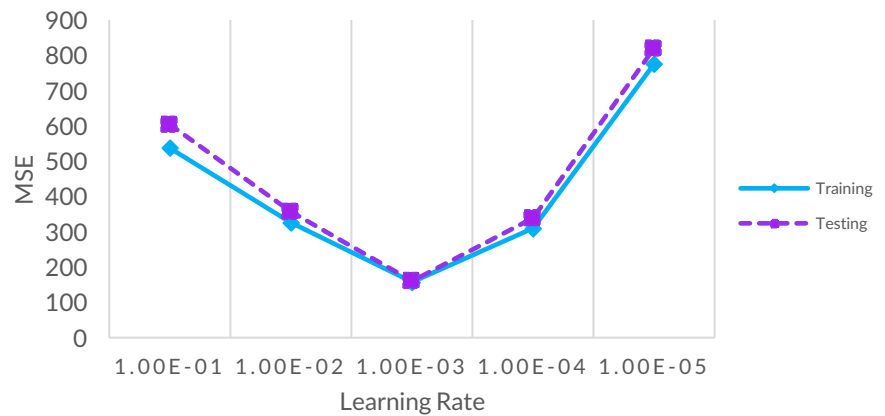


Figure 46: Loss (MSE) after 40 epochs for 5 different learning rates

The results suggest the best value is 1e-3 as it has the lowest testing and training error (highlighted red). To further tune learning rate, I ran the PyTorch Lightning learning rate finder across 100 values between  $10^{-4}$  to  $10^{-2}$  as this is the range in which the optimum can be found. The resulting graph is shown in Figure 47.

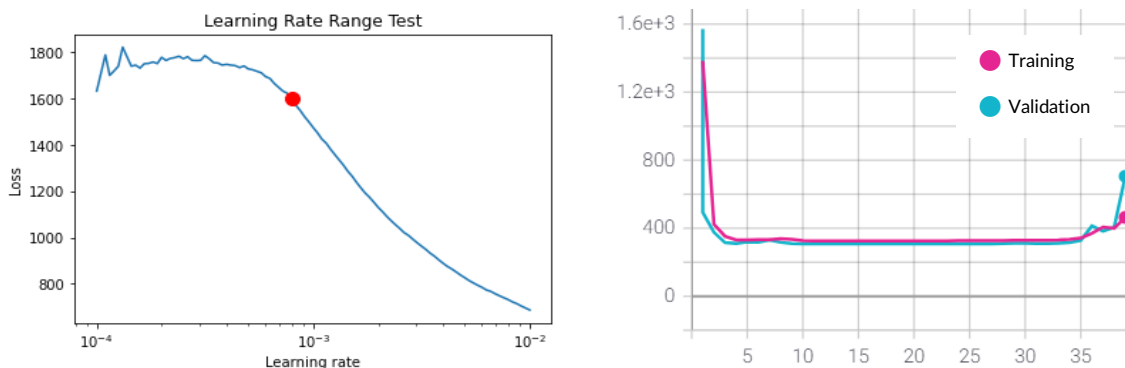


Figure 47: Learning rate finder results (left) and MSE over epochs for 0.000794 learning rate (right)

The learning rate range test suggested that 0.000794, marked by the red circle on Figure 47, would be the optimum. This value was used to train the model across 40 epochs to compare with the learning rates tested previously (see Table 11).

Learning Rate	Train MSE	Test MSE
0.000794	463.6	805.1
1e-3	157.7	160.4

Table 11: Performance of recommended range test learning rate vs 1e-3 over 40 epochs

The proposed learning rate has a far higher testing and training loss than 1e-3. While it is possible 0.000794 could outperform 1e-3 after more epochs, this is unlikely as indicated by Figure 47. This graph reveals training loss quickly converges to an optimum of 325.1 after four epochs, before increasing after 35 epochs.

### 6.2.2.1 Conclusion

Learning rate 1e-3 has both the lowest testing and training MSE and so has been selected. The value proposed by the learning rate finder is not suitable because it quickly converges to a poor loss value before the model starts overfitting.

### 6.2.3 Batch Size

To optimise batch size, the loss and run time for 10 values between 1 to 602 were recorded in Table 12. Final testing and training loss after 40 epochs is plotted in Figure 49, while Figure 48 shows change in training loss per epoch. Note that Figure 48 uses a Y-axis log scale as this helps to distinguish each bath size's performance.

Batch Size	Train MSE	Test MSE	Train Time (s)
1	96.3	128.1	167
2	94.0	125.5	113
4	94.0	119.2	85
8	117.7	157.8	72
16	153.4	201.4	64
32	150.4	194.9	60
64	194.4	234.5	59
128	301.4	349.4	58
256	302.8	377.4	59
602	438.0	534.0	59

Table 12: Performance for different batch sizes over 40 epochs

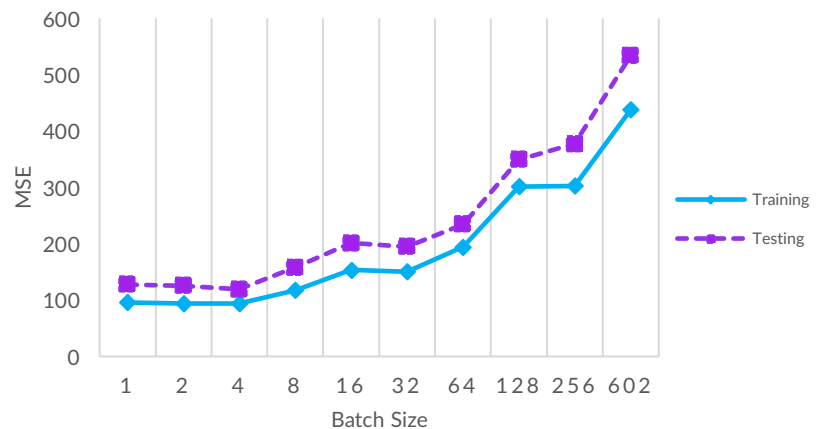


Figure 49: Testing and training loss (MSE) after 40 epochs for 10 batch sizes

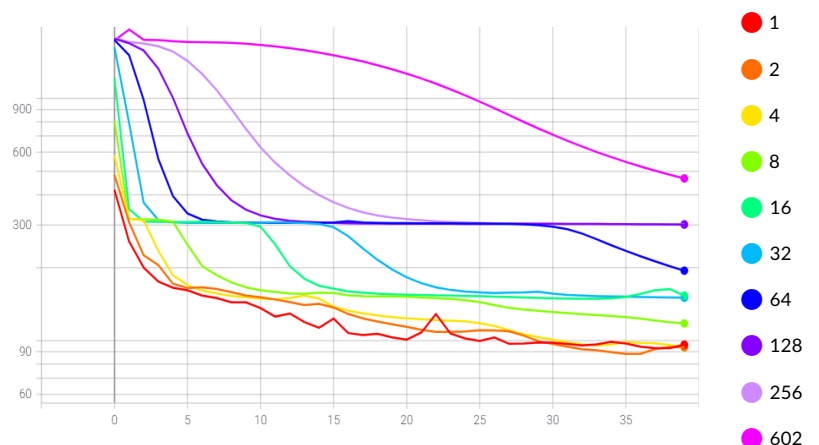


Figure 48: Training loss (MSE) over 40 epochs for 10 batch sizes

### 6.2.3.1 Conclusion

Batch size 2 and 4 have joint best final MSE in training, while 4 has the lowest in testing. Figure 48 shows there is no significant difference in trend for batch size 2 and 4, and it could be possible that 2 performs better over a higher number of epochs. However, a larger batch size significantly improves the training time of the model and so batch size 4 has been selected.

## 6.2.4 Optimiser Selection

Batch size was set as 4 and six optimisers evaluated over 40 epochs. For each, learning rate remained at  $1e-3$  and all other hyperparameters were kept at their defaults (see Table 13). As the loss for SGD was too high, a number could not be produced and so it could not be included in Figure 50 and Figure 51.

Optimiser	Train MSE	Test MSE
Rprop	227.9	222.6
SGD	NaN	NaN
Adadelta	373.1	354.7
AdaGrad	375.4	358.3
Adam	89.8	98.0
AdamW	101.8	106.5

Table 13: Performance for different optimisers over 40 epochs

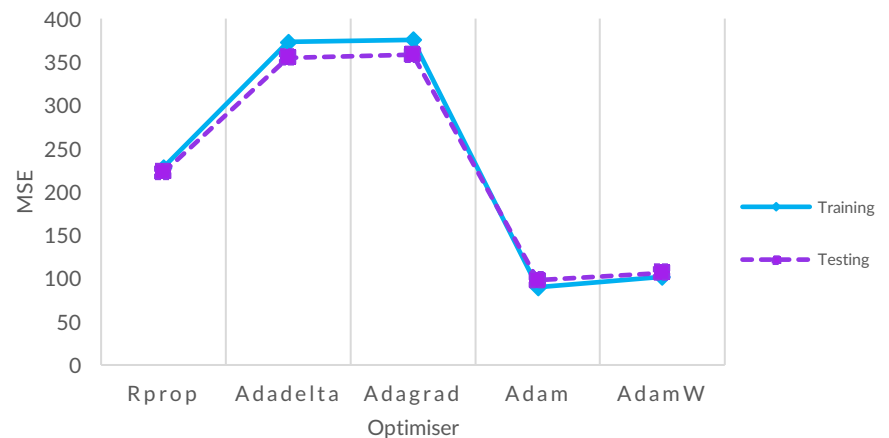


Figure 50: Testing and training loss (MSE) after 40 epochs for five optimisers

Adam was used before in part 6.2.3, with batch size 4 and achieved training loss 94.0. However, in this experiment, Adam has a lower training loss of 89.8 despite the initial model set up being the same. This is caused by stochastic nature of the network. This means that the network uses randomness, such as random weight initialisation, which leads to variation between runs [80].

Consequently, as the performance of Adam and AdamW were similar and neither fully converge to an optimum after 40 epochs, I decided to test loss of each after 100 epochs. Additionally, I tested the performance of each with the AMSGrad variant. This is a modification to the update rule that aims to prevent sub-optimal convergence [81]. The results are presented in Table 14 and Figure 52.

Optimiser	Variant	Train MSE	Test MSE
Adam	Standard	80.8	100.4
	AMSGrad	65.48	78.3
AdamW	Standard	80.5	111.7
	AMSGrad	84.9	96.6

Table 14: Performance of Adam vs AdamW over 100 epochs

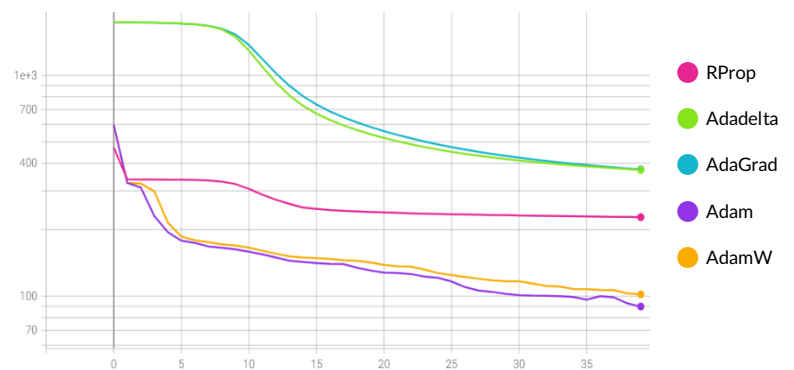


Figure 51: Training loss (MSE) over 40 epochs for five different optimisers

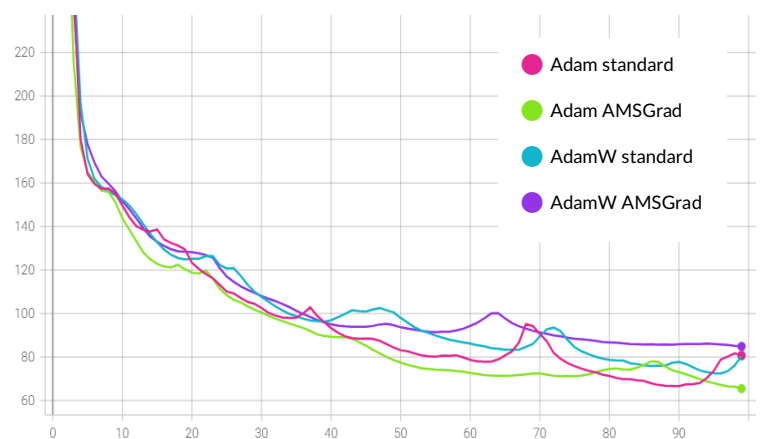


Figure 52: Training and validation loss (MSE) across 100 epochs for Adam and AdamW

### 6.2.4.1 Conclusion

Adam was the optimiser with the lowest testing and training loss after 40 epochs. After recording loss after 100 epochs with Adam, AdamW and their AMSGrad variants, the Adam AMSGrad variant performed best overall as demonstrated in Figure 52. Therefore, it has been selected as the best optimiser.

## 6.2.5 Hidden Layers

The network was tested with three variations to the number of hidden layers over 40 epochs with the updated Adam optimiser. The first variant includes only the input, encoded and output layer, the second is the original structure, while the third introduces two new hidden layers with 1024 features (see Table 15).

Hidden Layers	Layer Feature Number	Train MSE	Test MSE	Train Time (s)
1	16468 → 16 → 16468	133.8	132.4	70
3	16468 → 128 → 16 → 128 → 16468	108.1	133.9	92
5	16468 → 1024 → 128 → 16 → 128 → 1024 → 16468	102.7	150.5	510

Table 15: Performance based on number of hidden layers after 40 epochs

The lowest training loss was achieved when the network had 5 hidden layers. Figure 53 reveals that the loss remained consistently lower than with either 1 or 3 hidden layers. However, this architecture also attained the highest run time and testing loss. This high loss appears to be caused by a spike in validation loss (left of Figure 54). This likely indicates the model has started overfitting due to the increased complexity of the architecture preventing it from generalising on the test samples. The right-most graph Figure 54 demonstrates that the validation loss is more consistent with fewer layers.

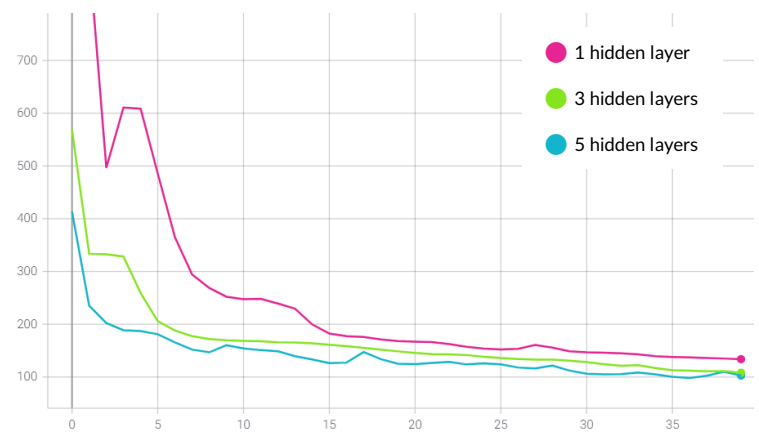


Figure 53: Training loss (MSE) over 40 epochs for 1, 3 and 5 hidden layers

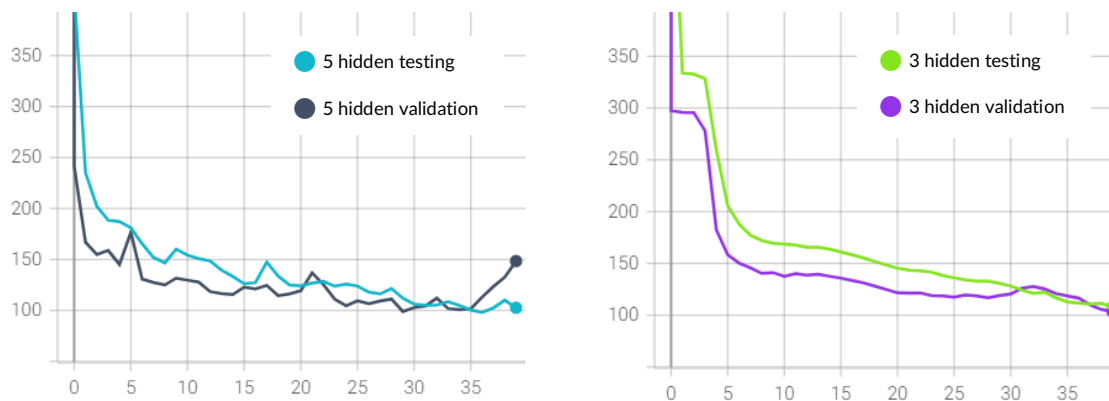


Figure 54: Training and validation loss (MSE) over 40 epochs for 5 hidden layers (left) and 3 hidden layers (right)

### 6.2.5.1 Conclusion

Although using 5 hidden layers accomplished the best training loss, the additional layers increase training time considerably. Testing loss was best overall when using 1 hidden layer, however, it was slightly lower than its respective training value. This suggests the model could be underfitting which can occur when a model is not complex enough [82]. While 3 hidden layers got neither the best training nor testing loss, it has been chosen as the best trade-off between minimising error and run time.

## 6.2.6 Activation Function

The ReLU activation function is used once within the encoder, and once within the decoder on hidden layers. Substituting ReLU for another function could reduce loss and so alternative activation functions have been tested as demonstrated in Table 16 and Figure 55.

Activation Function	Train MSE	Test MSE
Linear	65.8	66.6
ReLU	103.5	103.1
Leaky ReLU	98.7	98.7
PReLU	85.7	78.1
ELU	105.6	111.7
Sigmoid	516.3	517.0
Tanh	517.5	518.1

Table 16: Performance of different activation functions for hidden layers after 40 epochs

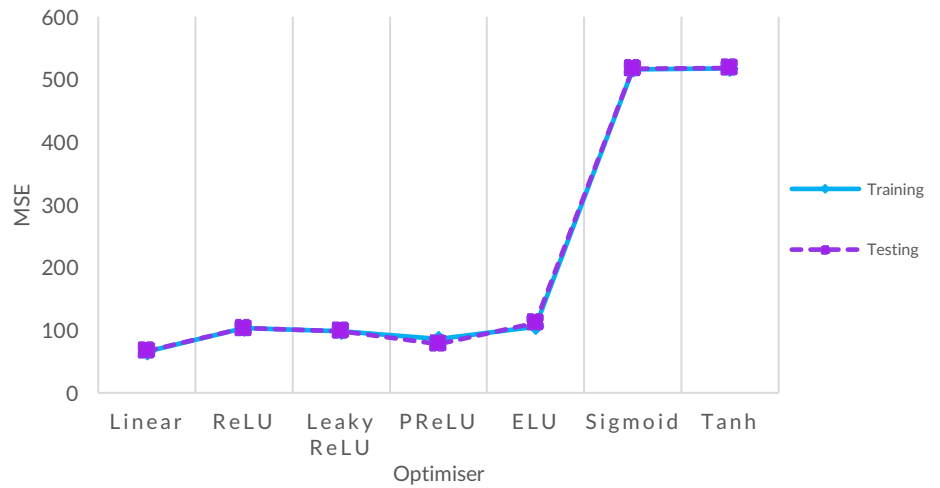


Figure 55: Testing and training loss (MSE) after 40 epochs for hidden layer activation functions

The linear activation function performed the best. Nevertheless, it can be detrimental to use only linear activation functions when training a deep neural network. This is because the model could not learn a non-linear representation of the data and so may overlook the underlying trend [83] which could be a problem when training the autoencoder on a different dataset. Therefore, in the following test, the number of epochs has been increased to 100 to see if the non-linear functions can match linear performance after longer training.

For an autoencoder outputting real numbers, it is sometimes suggested to use a non-linear activation function only in encoder layers [84]. Consequently, I have experimented using the non-linear activation function PReLU for the encoder, and either linear or non-linear for the decoder as shown in Table 17.

PReLU was selected for the encoder as it achieved the lowest testing and training loss of the non-linear functions. Additionally, percentage test accuracy is included to compare the percentage of gene counts successfully reconstructed and print screens of input tensors and decoded results are available in Appendix B.

Encoder Activation Function	Decoder Activation Function	Train MSE	Test MSE	Test Accuracy
PReLU	PReLU	47.2	72.5	35.7%
PReLU	Linear	55.3	68.2	38.8%
PReLU	Leaky ReLU	51.3	74.2	32.2%
Linear	Linear	56.0	66.7	38.3%

Table 17: Performance of combinations of activation functions after 100 epochs

PReLU + Linear achieved the highest test accuracy, although its values are not significantly different to Linear + Linear. Training loss, plotted in Figure 56, shows that Linear + Linear performs best with a low number of epochs, such as 40, as also demonstrated in the previous experiment. However, increasing the epochs causes the Linear + Linear loss to rise while the non-linear models' loss decrease.

### 6.2.6.1 Conclusion

As using PReLU for the encoder and linear activation for the decoder had the best accuracy and a good performance over a high number of epochs, they have been selected for the autoencoder.

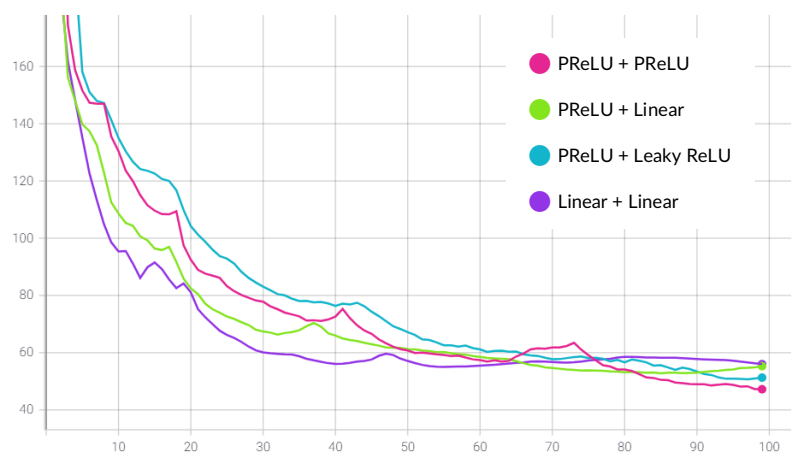


Figure 56: Training loss (MSE) after 40 epochs for combinations of activation functions



## 6.2.7 Number of Epochs

To determine a number that reduces underfitting or overfitting, the autoencoder was trained over 400 epochs (Figure 57). Training and validation loss both steadily decrease between 0 to 135 epochs. Then training loss continues to improve with further epochs, while validation loss remains consistent until 340 epochs. After this point, the model overfits.

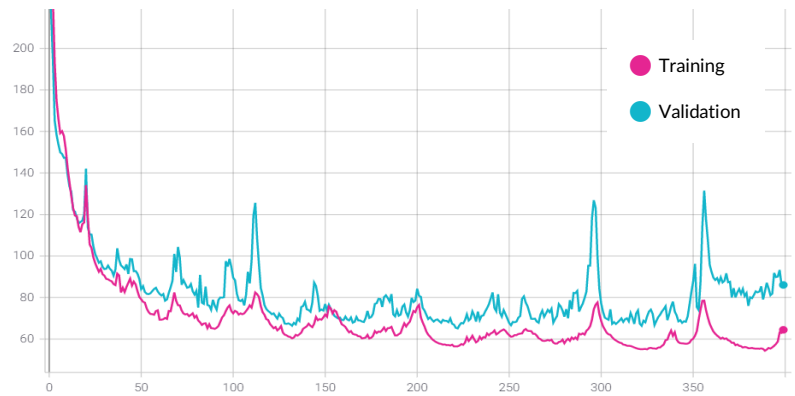


Figure 57: Training and validation loss (MSE) after 400 epochs

### 6.2.7.1 Conclusion

As performance does not significantly improve beyond 135 epochs, and since the lowest validation loss was 65.6 after 222 epochs, I have selected 250 as the maximum number of epochs. This should allow the validation loss to reach a minimum, prevent the model overfitting and ensure training time is reasonable. Note that as a maximum, this number may not be reached when training the final model due to early stopping.

## 6.2.8 Summary of Autoencoder Optimisation

The autoencoder has been optimised for the benchmarking dataset. The following changes were made:

- ◆ The optimiser was set as the AMSGrad variant of Adam with learning rate 1e-3.
- ◆ Batch size was set to 4.
- ◆ The network architecture remains the same as in the design consisting of an input layer, a hidden layer either side of the encoding layer and an output layer.
- ◆ The first activation function of the hidden layer has been set as PReLU, while the second is linear.
- ◆ The maximum number of epochs is set as 250.

## 6.3 Training the Optimised Network

After optimising the autoencoder's structure and hyperparameters, it was trained on the benchmarking data and the weights saved to file. The training of autoencoders for the simulated and mouse cortex evaluation data is also documented in this section, although they were not trained until after thoroughly testing the benchmarking dataset. These saved models can be reloaded and provide the encodings used to perform clustering.

### 6.3.1 K-Fold Cross Validation

In the tests above, samples were randomly split into testing, training, and validation. However, k-fold cross validation can be used to find a better split. Therefore, data set aside for training was split into 5 folds and trained for 20 epochs. Validation loss for both datasets is recorded in Table 18 and print screens provided in Appendix B. The best fold for each has been highlighted red. Note that loss between each dataset cannot be compared as they have different numbers of training samples and genes.

Fold	Validation Loss (MSE)		
	Benchmark	Simulated	Mouse Cortex
0	79.9	99.7	23.4
1	104.2	96.6	25.7
2	83.5	105.4	24.8
3	101.4	105.7	29.3
4	103.5	108.1	29.3

Table 18: 5-fold validation loss for benchmarking, simulated and mouse cortex evaluation data



### 6.3.1.1 Conclusion

For both the benchmarking and mouse cortex datasets, fold 0 achieved the lowest validation loss so these splits were selected respectively. For the simulated data, fold 1 was selected.

## 6.3.2 Training, Validation and Testing Results

### 6.3.2.1 Training and Validation

For each dataset, an autoencoder was trained over 250 epochs and the weights saved to file.

#### Benchmarking Data

As seen in Figure 58, the lowest validation loss of the first autoencoder was 56.9. As this was achieved after 247 epochs, the model was saved at this state.

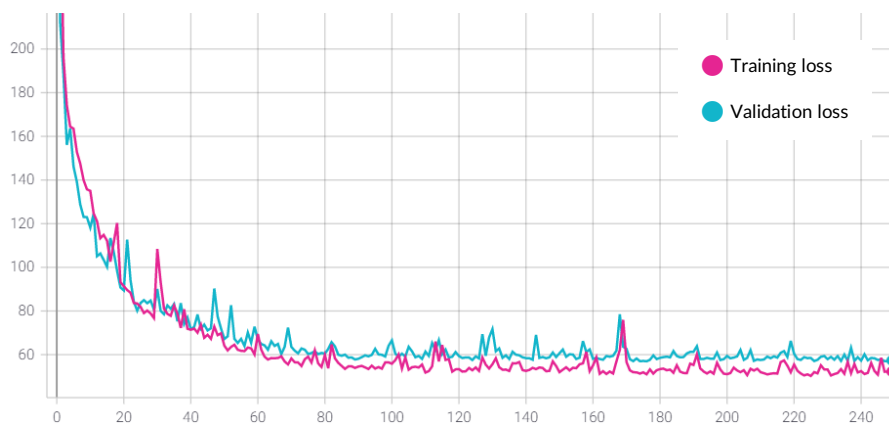


Figure 58: Benchmarking data autoencoder's training and validation loss over 250 epochs

#### Simulated Data

A second autoencoder (Figure 59) was trained reaching the best validation loss of 81.2 after 228 epochs.

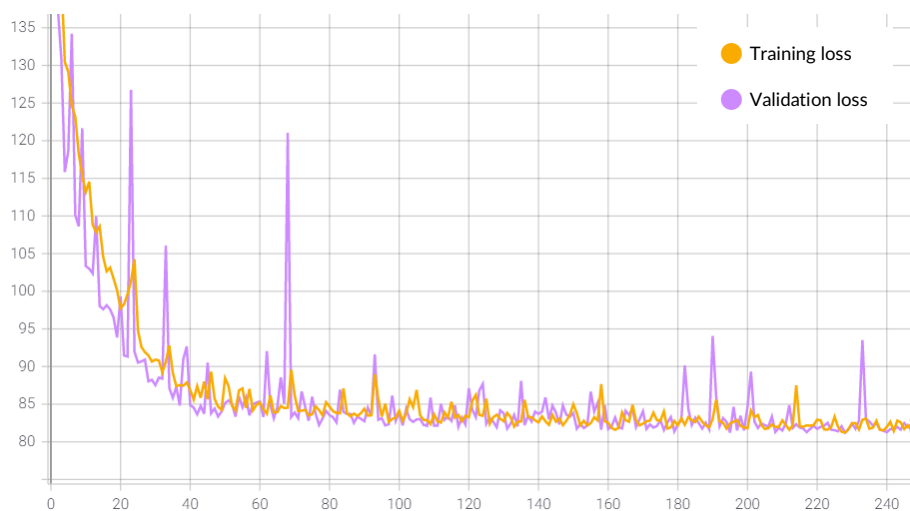


Figure 59: Simulated data autoencoder's training and validation loss over 250 epochs

#### Mouse Cortex Data

The final autoencoder's state was saved after 185 epochs when reaching the best validation loss of 16.0. See Figure 60.

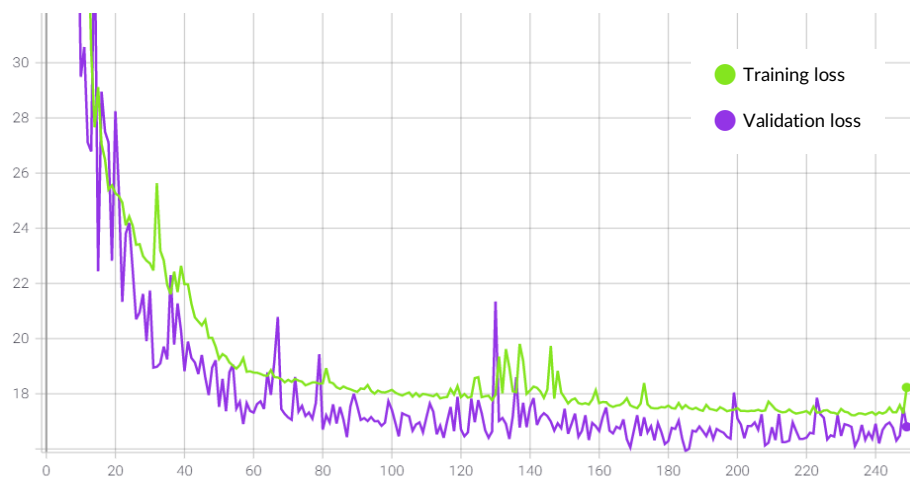


Figure 60: Mouse cortex data autoencoder's training and validation loss over 250 epochs

### 6.3.2.2 Testing

The benchmarking autoencoder was tested against 150 unseen samples, while the simulated and mouse cortex autoencoders were tested against 500. The average test loss of mini-batches in each are plotted in Figure 61. As batch size is set to 4, the benchmarking has a total of 38 mini-batches, while the others have 125 as reflected in the graphs below.

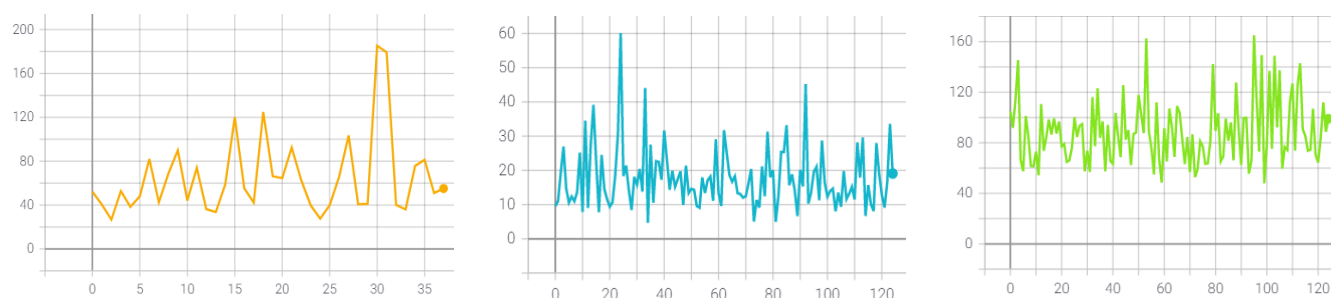


Figure 61: Average loss (MSE) per mini-batch during testing of benchmarking data autoencoder (left), simulated (middle) and mouse cortex right)

### Benchmarking Data

Overall, testing had an average loss of 65.2 and accuracy of 39.9%, in which accuracy is the percentage of gene counts correctly reconstructed. A screen shot can be seen in Appendix B.

### Simulated Data

The second autoencoder had a loss of 87.8 and the lowest accuracy of 36%. Therefore, the autoencoder does not capture the features of this dataset as well as the other models.

### Mouse Cortex Evaluation Data

This model achieved an average loss of 17.7 and the highest accuracy of 50.3% meaning the autoencoder was the best at preserving the structure of this dataset.

### 6.3.2.3 Conclusion

The final model for the benchmarking data achieved a marginally higher test accuracy than any of the optimisation experiments. Interestingly, even though the hyperparameters were tuned for the benchmarking autoencoder, the accuracy of the mouse cortex autoencoder was higher. However, this is likely because this dataset has 1072 genes while the other two have 16468 making it a lot easier for the autoencoder to retain the original features.

After the models were trained, they were saved to file. This means they can be loaded again and will produce the same test accuracy when given the same samples.

## 6.4 Clustering

As the autoencoders have now been trained, k-means and agglomerative hierarchical clustering are applied to both the encoded and original data and the performance evaluated as explained in objective 5. Though most experiments were first tested on the benchmarking data, testing of the other datasets has been included alongside so that results can be compared.

Some of the graphs are marked with “Clickable” as they include hyperlinks to interactive versions. It is worth pointing out that some experiments refer to using the data without standardization. By this I mean that it has not been applied before dimensionality reduction, such as PCA, as standardization has been consistently used on the extracted components. Refer to Clustering in System Design for further clarification.

### 6.4.1 Ground Truth of Data

The cell lines and group metadata for each dataset form clusters that can be compared against those obtained through experiments later in this report. This ground truth enables the calculation of accuracy and ARI. Therefore, the purpose of this section is to explore the target clusters of each dataset.

#### 6.4.1.1 Benchmarking Data

Each cell belongs to one of three cell lines: [H1975](#), [H2228](#), [HCC827](#). In Figure 62, the cell lines of all 902 cells are plotted against the first two principal components as transparent squares, while the 150 samples selected for testing are plotted as coloured dots. Figure 63 illustrates the testing samples against the top three components. This is because the initial experiments during this section were tested on the training data of the autoencoder, while later experiments were conducted on all samples.

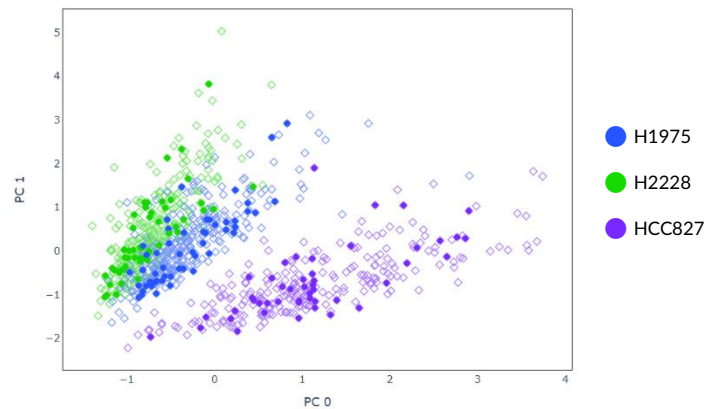


Figure 62: (Clickable) 2D plot of benchmark data's cell lines

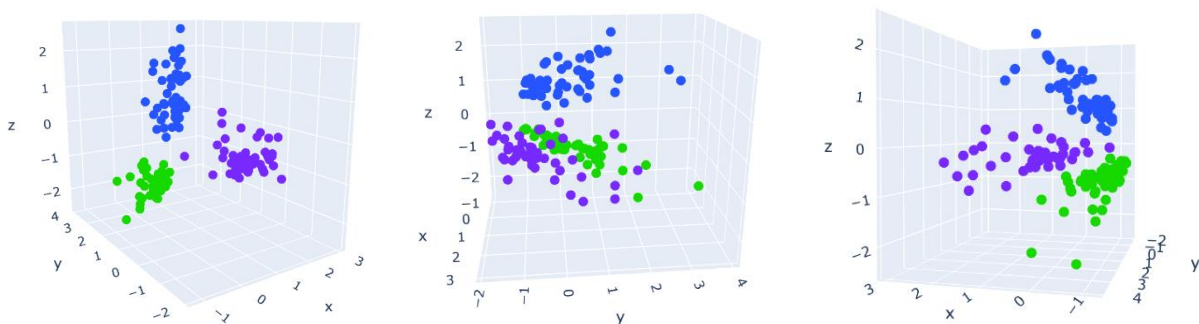


Figure 63: (Clickable) 3D plot of benchmark testing data's cell lines from three angles

### 6.4.1.2 Simulated Data

The simulated cells contains one more cluster than the benchmarking data as cells are divided into four groups: **Group1**, **Group2**, **Group3**, **Group4**. In addition, 500 samples were selected rather than 150. Refer to Figure 64 and Figure 66.

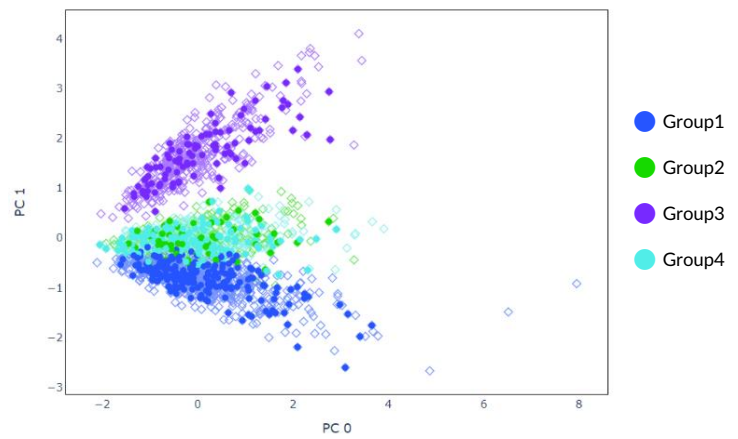


Figure 64: 2D plot of simulated data's expected groups

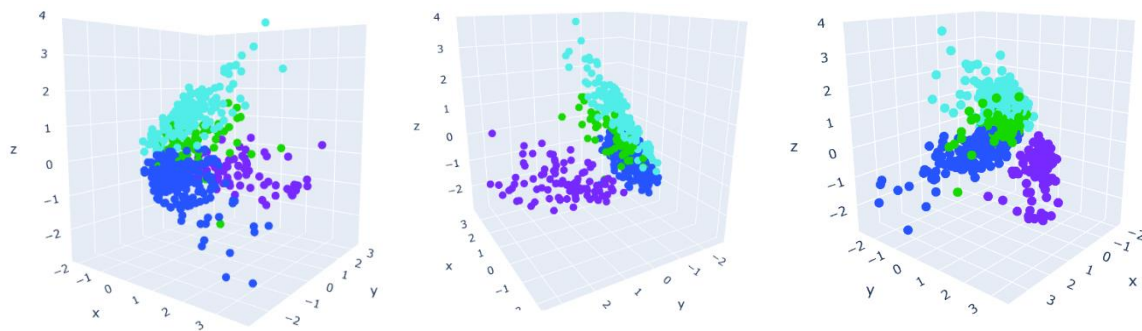


Figure 66: 3D plot of simulated test data's groups from three angles

### 6.4.1.3 Mouse Cortex Data

The mouse cortex dataset is far more complicated than the previous two both as cells have been divided into seven groups and the differences in gene expression are far more subtle. In addition, these groups were found using the BackSPIN algorithm, and so they are likely to be less reliable than the other two datasets. See Figure 65 and Figure 67.

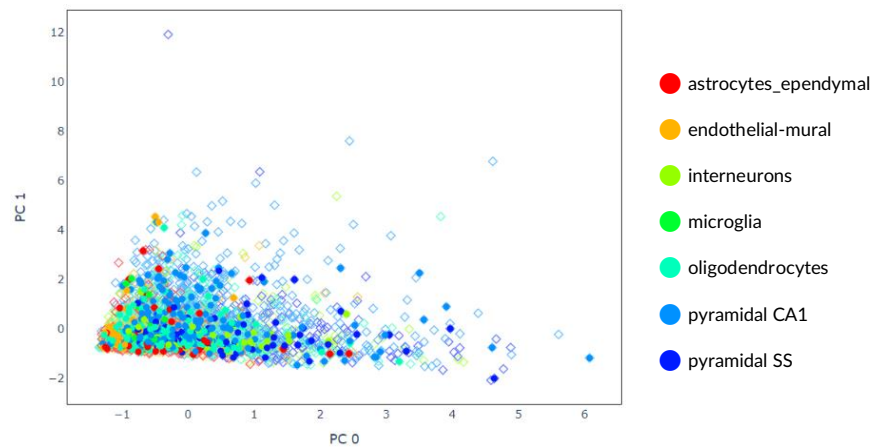


Figure 65: (Clickable) 2D plot of mouse cortex data's expected groups

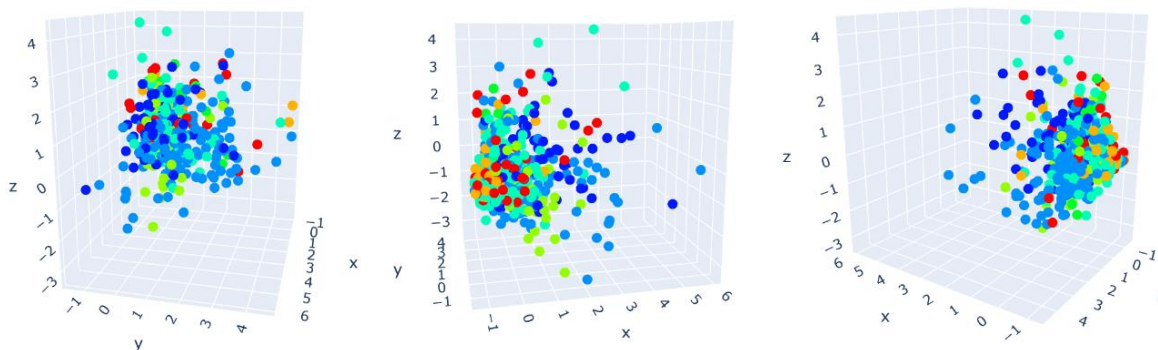


Figure 67: (Clickable) 3D plot of mouse cortex test data's expected groups from three angles

## 6.4.2 K-Means

The optimised autoencoders were loaded and encodings obtained for the test samples of each dataset. These are samples that were not used to train the autoencoder. This allows evaluation of the autoencoder's ability to generalise which is useful because some scRNA-seq datasets do not provide a label for every cell.

Before applying clustering, the elbow method and silhouette coefficient were run to find the best suggestions for k. Then k-means was applied to both the encoded and raw unencoded data and the results compared.

### 6.4.2.1 Benchmarking Data

#### Finding the Best K

For the encoded benchmarking data, the elbow method suggested three clusters while the silhouette coefficient suggested five (Figure 68). I decided upon three clusters as this matches the expected number of clusters.

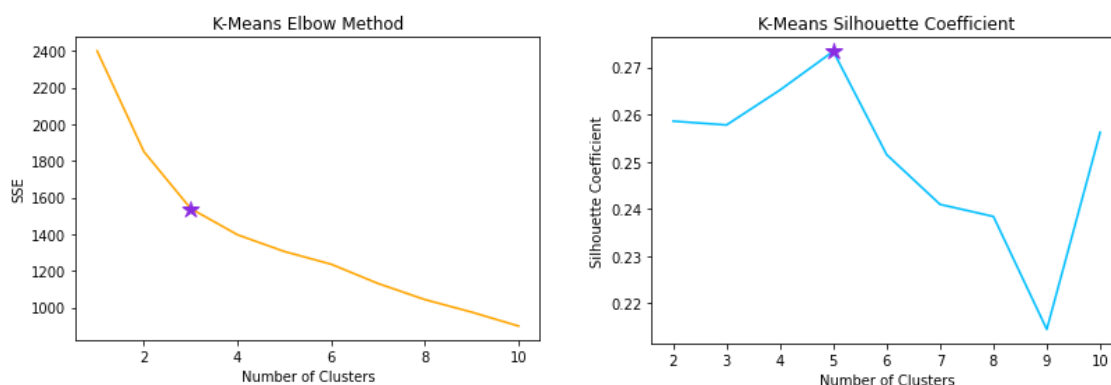


Figure 68: Elbow method (left) and silhouette coefficient (right) for benchmarking data

#### K-Means on Encoded Data with PCA

For the first experiment, k-means was applied to the first two principal components of the encoded benchmarking data. The first graph in Figure 69 shows clusters predicted by k-means and the second shows the expected clusters based upon cell lines. Figure 70 reveals which cells were not assigned to correct clusters by highlighting them in red.

This resulted in an accuracy of 68.7% as k-means was not able to assign the correct cluster to 47 out of 150 cells. The ARI was 0.396 which is lower than the average of the benchmarking metadata ( $0.396 < 0.436$ ). It is apparent that this set up did not perform well at identifying the cell lines as the shape of the left-most predicted clusters clearly do not match those of the actual clusters. One reason for this issue is because cell lines [H1975](#) and [H2228](#) overlap in the two-dimensional space.

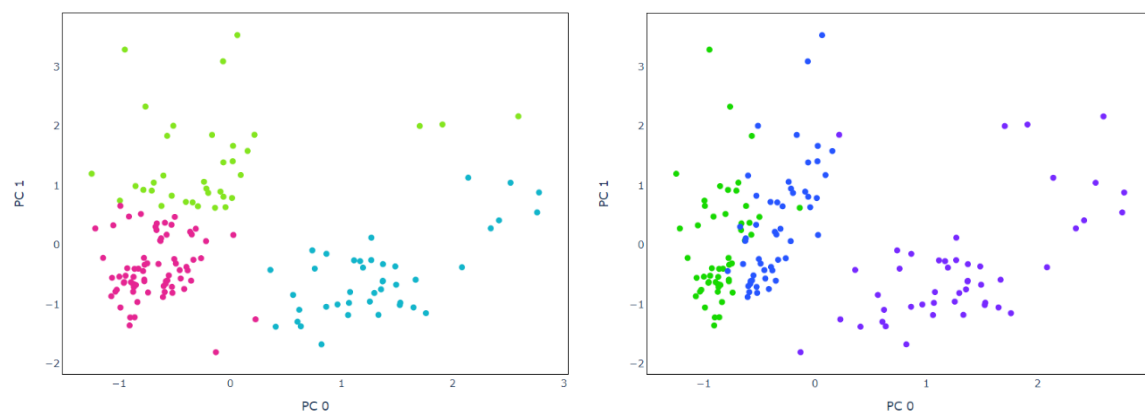


Figure 69: (Clickable) K-means prediction (top left), actual clusters (top right) for 2 PC encoded benchmarking data

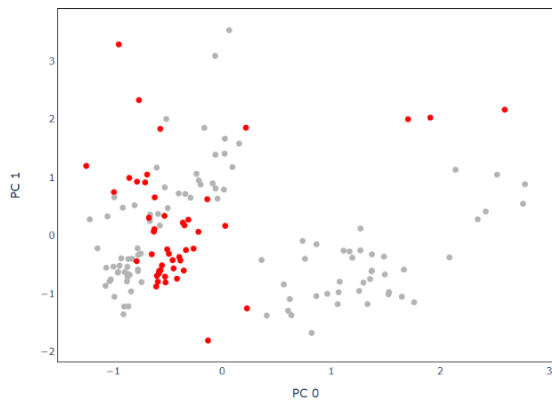


Figure 70: (Clickable) K-means prediction (top left), actual clusters (top right) and difference (bottom) for 2 PC encoded benchmarking data

To solve this problem, standardization was applied to the encoded data both before and after applying PCA with 2 components (Figure 71). This separated the cell lines and as a result, accuracy increased to 97.3%. This meant that now only 4 out of the 150 cells were incorrectly identified. The new ARI is 0.920 suggesting that this experiment has outperformed the best result in the metadata experiments ( $0.959 > 0.742$ ).

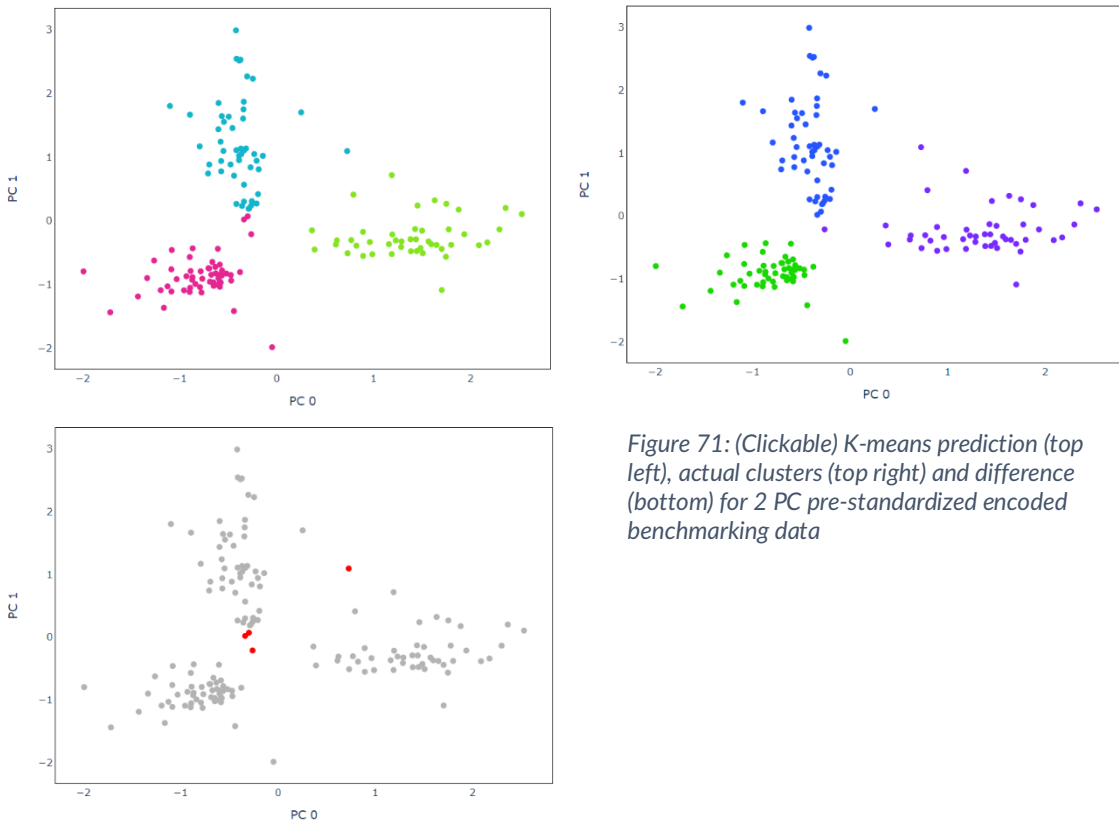


Figure 71: (Clickable) K-means prediction (top left), actual clusters (top right) and difference (bottom) for 2 PC pre-standardized encoded benchmarking data

When a third principal component was included to cluster the pre-standardized encoded data, accuracy increased to 98.0% and ARI to 0.940 (Figure 73). However, I discovered using three PCs on the unstandardized data was even better with 98.7% accuracy and ARI of 0.959 (Figure 79) suggesting that standardization may not always improve accuracy.



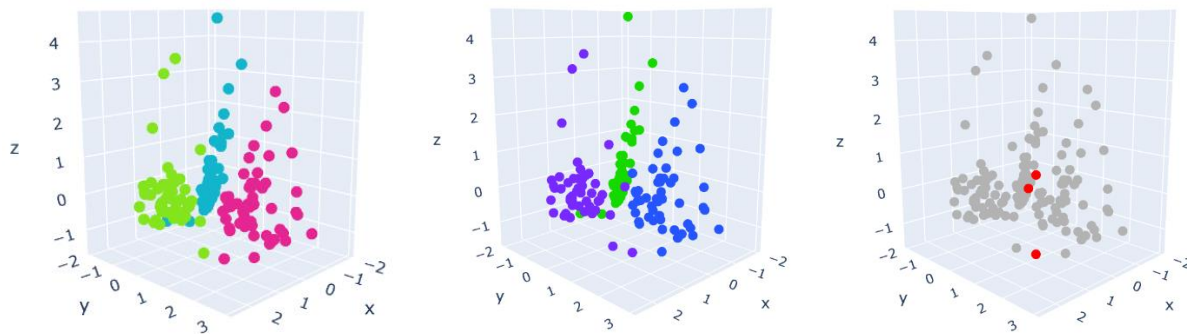


Figure 72: (Clickable) K-means prediction (left), actual clusters (middle) and difference (right) for 3 PC encoded benchmarking data with standardization

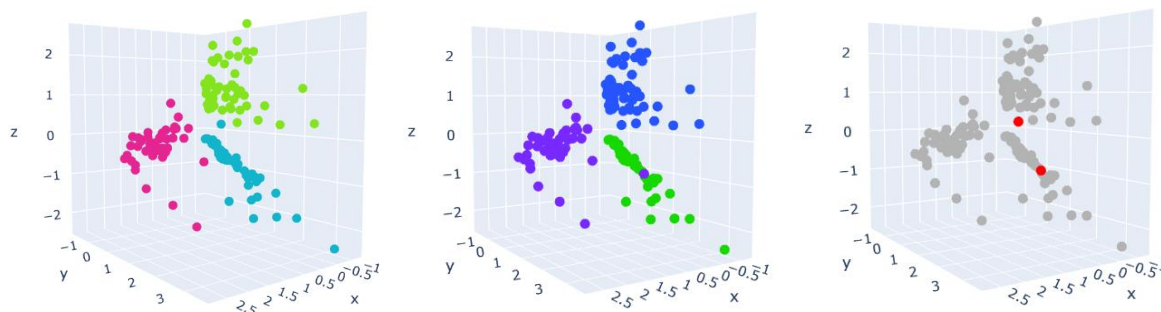


Figure 73: (Clickable) K-means prediction (left), actual clusters (middle) and difference (right) for 3 PC encoded benchmarking data without standardization

Although the improvement is not large, the experiment demonstrated that adding another principal component was an effective method to increase accuracy. To see if I could improve it any further, I tried using different number of principal components and recorded the average accuracy over 5 iterations of k-means.

The results suggest that the optimal number of principal components for the unstandardized data is 4 (Figure 93), while for the standardized data it is 7, with both attaining a near perfect 99.3% accuracy and 0.980 ARI. Although k-means was run on higher dimensional data, clusters have been plotted against the top two principal components in the graphs below. Unfortunately, neither version was able to correctly classify CELL\_000061.

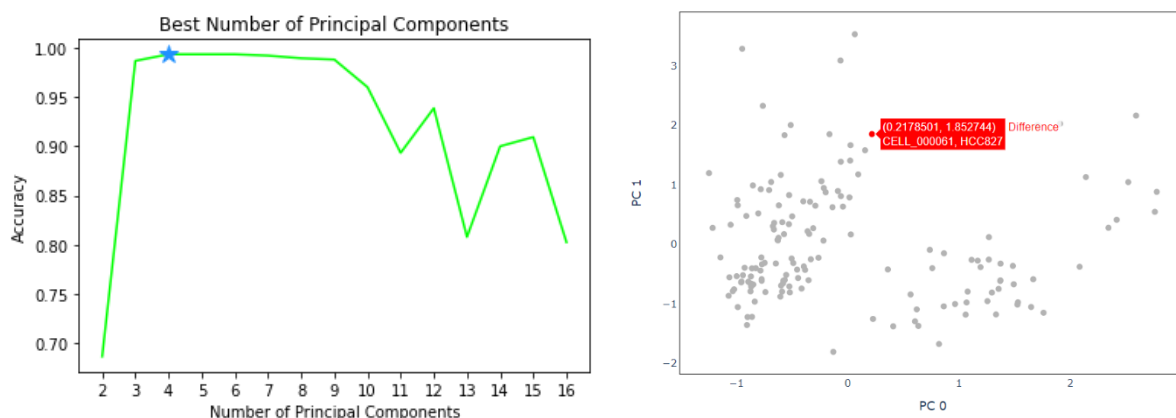


Figure 74: K-means accuracy for unstandardized encoded data when testing different numbers of principal components (left) and a plot showing the incorrectly identified cell for 4 PCs (right) (Clickable)

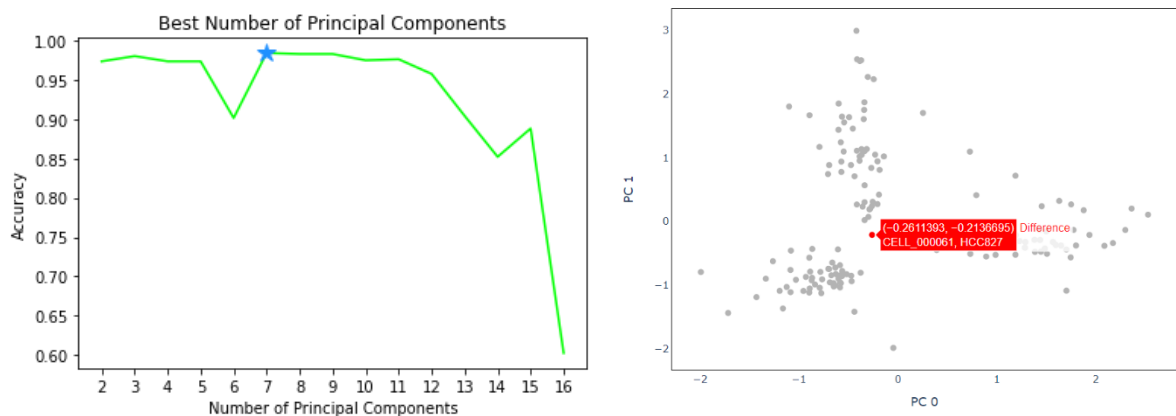


Figure 75: K-means accuracy for standardized encoded data when testing different numbers of principal components (left) and a 2D plot showing the incorrectly identified cell for 7 PCs (right) (Clickable)

### K-Means on Unencoded Data with PCA

The accuracy when using the encoded data is very good. However, to evaluate if using the autoencoder has improved performance, the experiments must be repeated on the unencoded data.

PCA was run on the unencoded data both with and without applying standardization. Again, the top two principal components were selected on which k-means was applied. The unstandardized data, shown by the left-most graphs in Figure 76 and Figure 77, got a low accuracy of 64% and ARI of 0.387. When standardization was applied, seen by the right-most graphs, this received a higher accuracy to 72.7% and ARI of 0.546. However, in both cases, the cell lines overlap in the 2D space preventing k-means from distinguishing the two. As these results are worse than when run on encoded data, this implies in this case that the autoencoder was successful at condensing important features into a lower dimensional space.

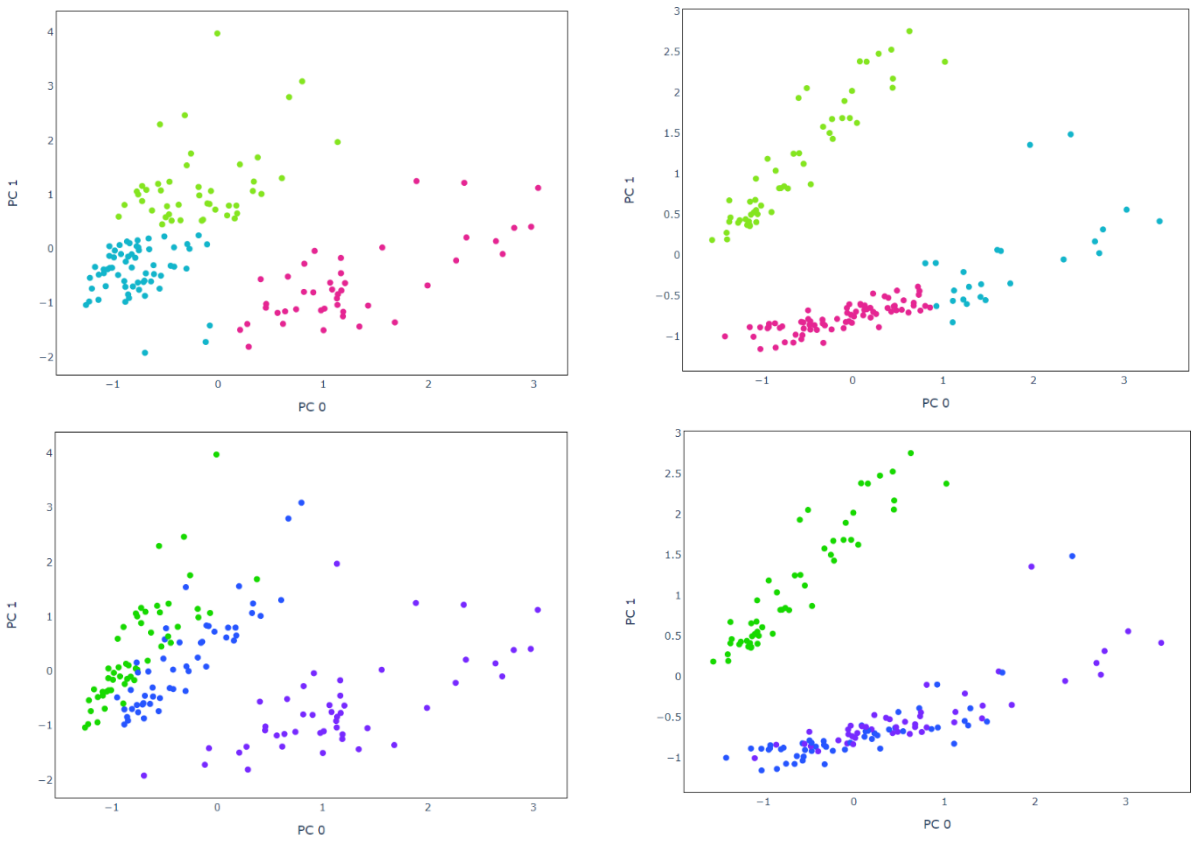


Figure 76: (Clickable) K-means using 2 PCs from the raw unencoded data (left) and standardized unencoded data (right)



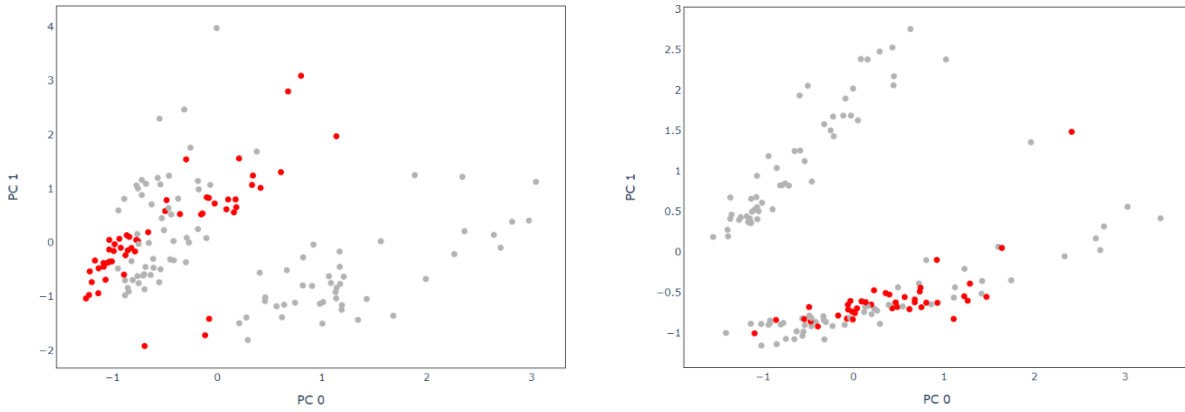


Figure 77: (Clickable) Incorrectly identified cells from k-means using 2 PCs for the raw unencoded data (left) and standardized unencoded data (right)

For both the raw and standardized unencoded data, the best number of principal components was three (Figure 78). Without standardization, accuracy was 99.3% and ARI 0.98 (Figure 79). While with standardization, 100% of the cells were correctly identified giving a perfect ARI of 1 (Figure 80).

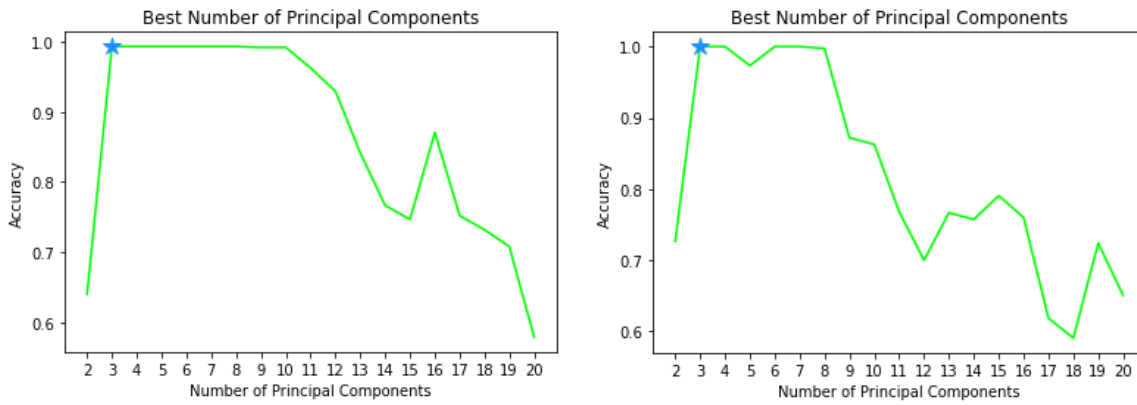


Figure 78: Comparing accuracy to number of principal components for k-means on the raw unencoded data (left) and standardized unencoded data (right)

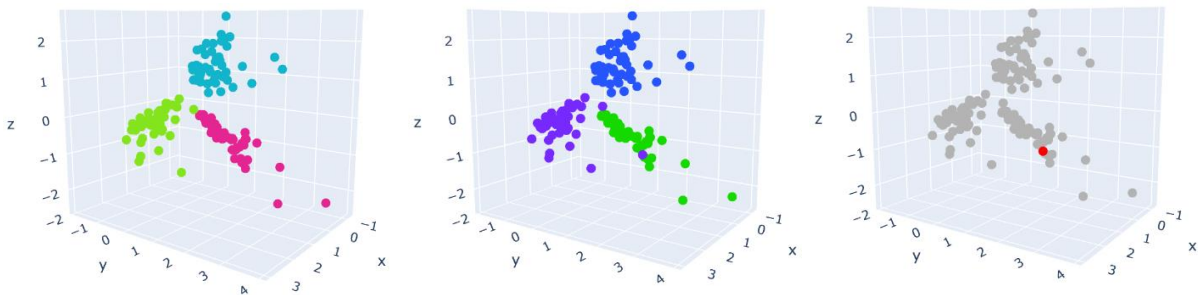


Figure 79: (Clickable) K-means prediction (left), actual clusters (middle) and difference (right) for 3 PC original benchmarking data

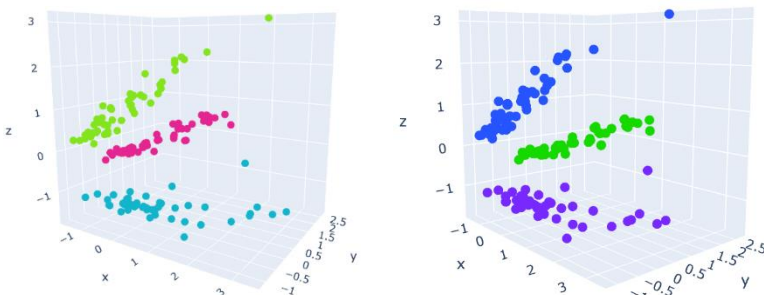


Figure 80: (Clickable) K-means prediction (left) and actual clusters (right) for 3 PC original benchmarking data with standardization

## Conclusion

The results on the unencoded using three principal components are marginally better than when encoding was applied. This means that when applying k-means to this dataset, the autoencoder's encoding did not improve performance. However, as these experiments are limited to just 150 samples it is hard to say if the difference is statistically significant. A table of results directly comparing the experiments is available in Appendix B.

### 6.4.2.2 Simulated Data

#### Finding the Best K

For the simulated data, both the elbow method and silhouette coefficient predicted four clusters which matches the true cluster number. See Figure 81.

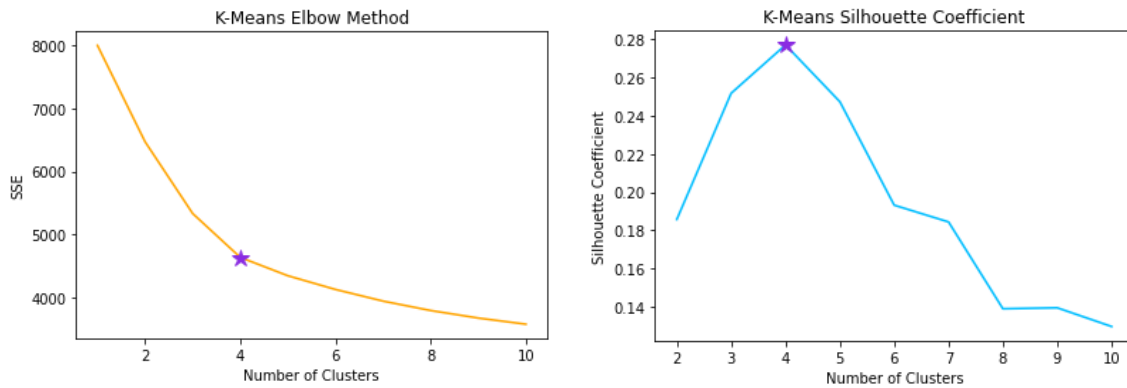


Figure 81: Elbow method (left) and silhouette coefficient (right) for simulated data

#### K-Means on 2 Principal Components

For both the encoded and unencoded data when no standardization was applied on two principal components, the performance was poor with 49.8% (Figure 82) and 48.8% (Figure 83) accuracy respectively. Despite the encoding having a slightly higher accuracy, it had a lower ARI of 0.167 compared to 0.171. This means that while the encoding was able to match more cells to their group, samples present in the clusters were less similar. Nevertheless, this difference is hard to detect in the graphs below.

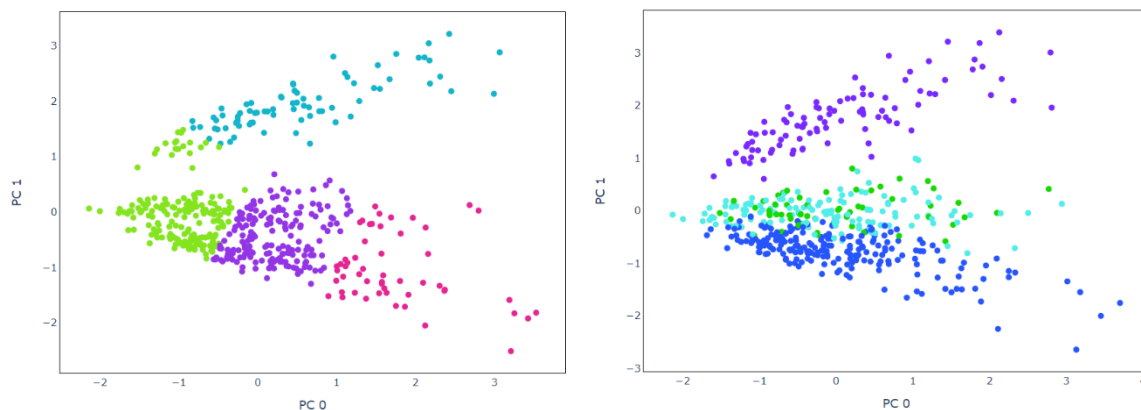


Figure 82: (Clickable) 2 PC k-means on the encoded simulated data without standardization

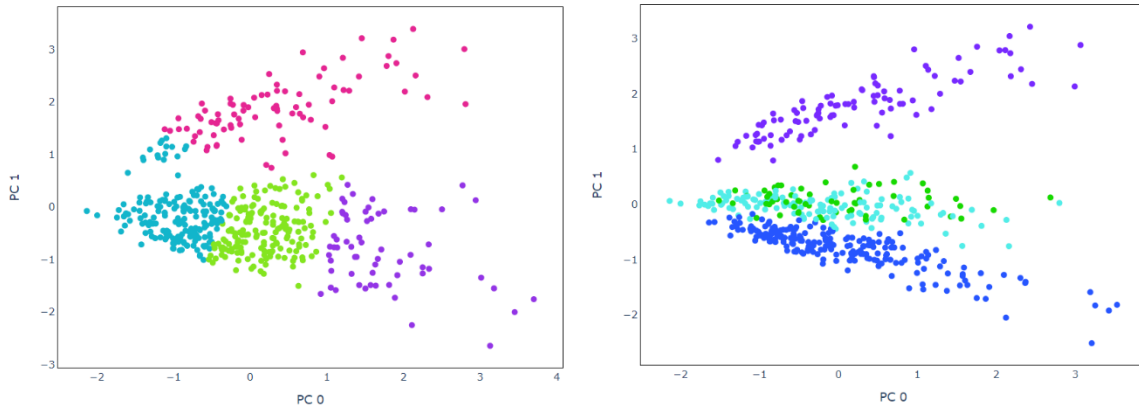


Figure 83: (Clickable) 2 PC k-means on the unencoded simulated data without standardization

Like with the benchmarking dataset, the accuracy after applying standardization with two principal components was significantly increased to 92.4% for encoded (Figure 84) but just 53.4% for unencoding (Figure 85). On inspection of the graphs, it is easy to see why as standardization was able to separate the encoded cells into distinct groups with only a small amount of overlap, while without encoding, Group2, Group3 and Group4 are placed on top of one another and only Group1 is distinct.

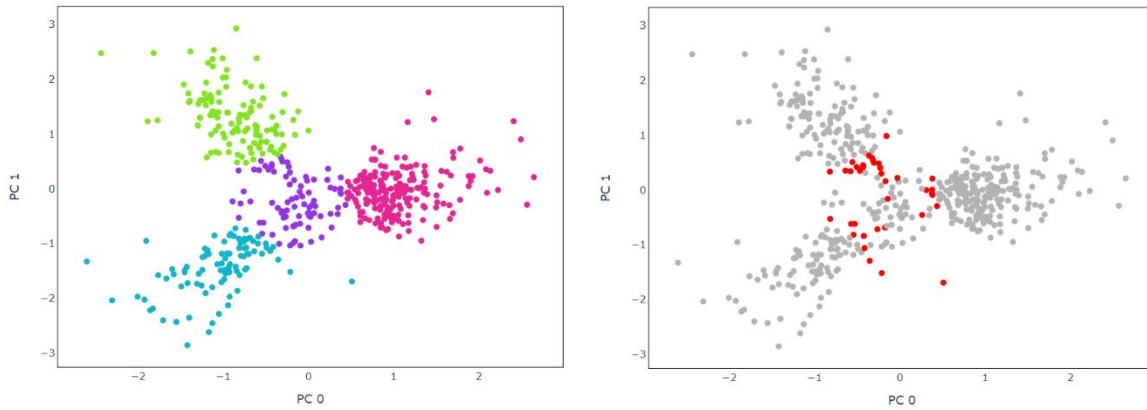


Figure 84: (Clickable) Predicted clusters (left) and incorrect cells (right) for k-means using 2 PCs on the encoded simulated data

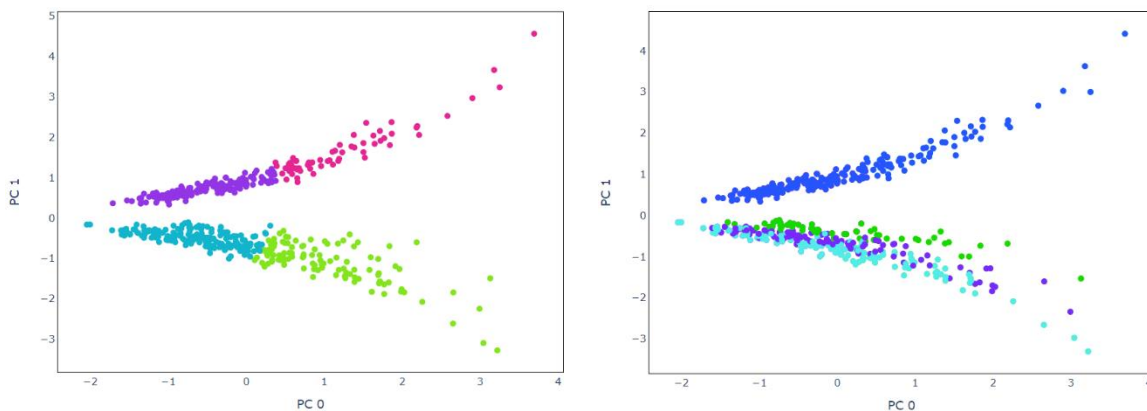


Figure 85: (Clickable) Predicted clusters (left) and expected groups (right) for k-means using 2 PCs on the unencoded simulated data

### K-Means on Optimal Number of Principal Components

When the number of principal components was optimised, three out of four experiments received 100% accuracy and ARI of 1.0. For the encoded data, this result was achieved using nine principal components with standardization and six without (see Figure 86). For the unencoded data, 19 principal components were required on the unstandardized data, while with standardization, the best result was 88.4% accuracy and 0.897 ARI on 5 principal components.

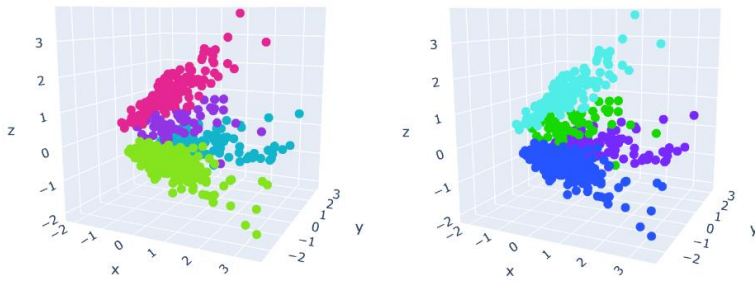


Figure 86: (Clickable) 3D plot of predicted clusters (left) expected groups (right) for k-means using 6 PCs on encoded simulated data

### Summary of K-Means on Simulated Data

K-means was applied to the first two principal components of the testing data both with and without applying encoding and standardization. Then this was repeated using the optimum principal components. Three of these experiments correctly identified all 500 of the cells, although when encoding was used, far fewer principal components were required. Therefore, for this dataset the autoencoder's encoding appears to have been beneficial as using fewer PCs reduces computational complexity. The full table of results can be seen in Appendix B.

#### 6.4.2.3 Mouse Cortex Data

##### Finding the Best K

For the mouse cortex data, the elbow method's recommendation was 4 while the silhouette coefficient's was 2 (Figure 87). Neither of these reflect the expected number of clusters which is 7. However, I selected 7 anyway so that I could compare the clusters to the BackSPIN predicted labels.

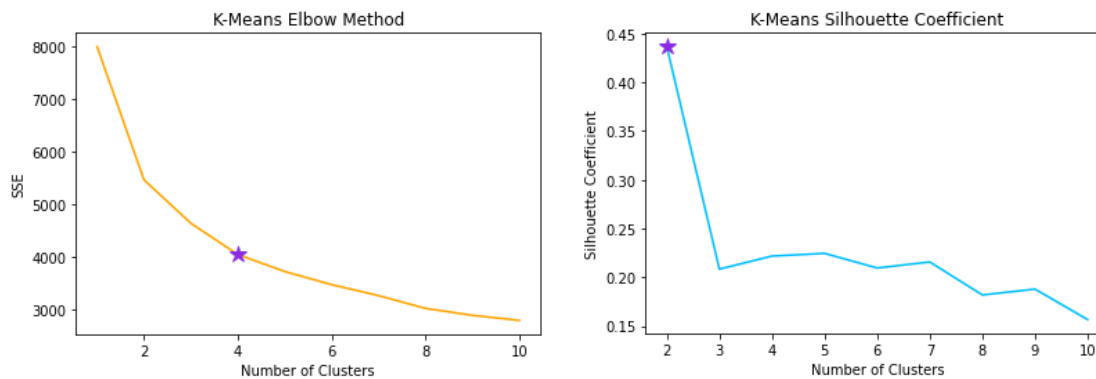


Figure 87: Elbow method (left) and silhouette coefficient (right) for mouse cortex data

### K-Means on 2 Principal Components

When the encoded data was run on two principal components without standardization, accuracy was 29.2% for the encoded version and 29.8% without (see Figure 88). Both results are poor, though this is not surprising when trying to predict the target clusters as they are not separated enough for k-means to work effectively.

Applying standardization did not improve the performance for the encoded data as accuracy was even lower at 25.4%. However, accuracy did increase for the unencoded data to 36.6% (Figure 89). Though this is a good improvement, it is clear to see that the cells still overlap too much to be reliably clustered.

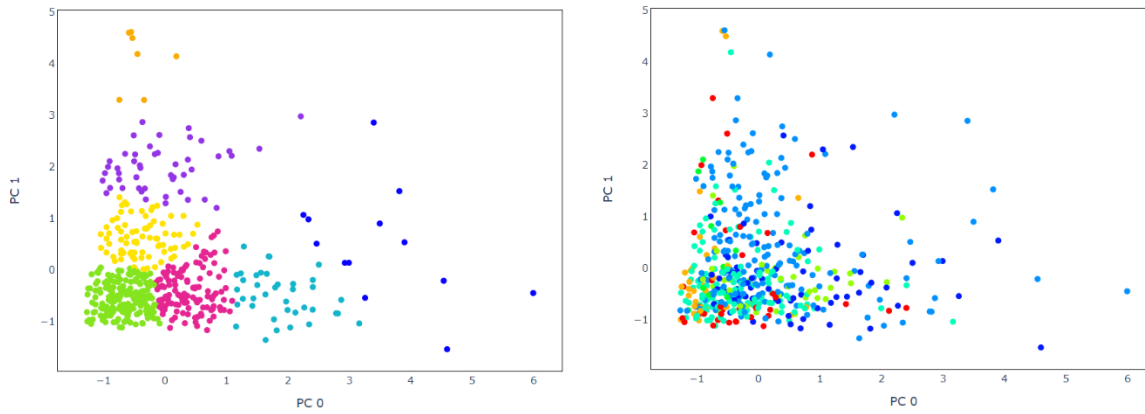


Figure 88: (Clickable) 2 PC k-means on the unencoded mouse cortex data without standardization (left) and BackSPIN labels (right)

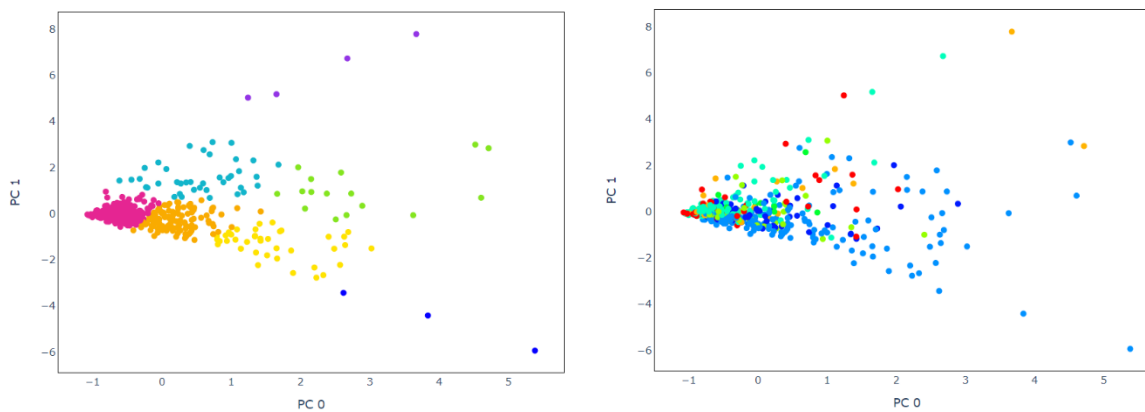


Figure 89: (Clickable) 2 PC k-means on the unencoded mouse cortex data after standardization

## K-Means on Optimal Number of Principal Components

The standardized encoded data achieved the best accuracy of 36.4% with 15 principal components (Figure 90). Nevertheless, this is still worse than using two principal components on the standardized unencoded data.

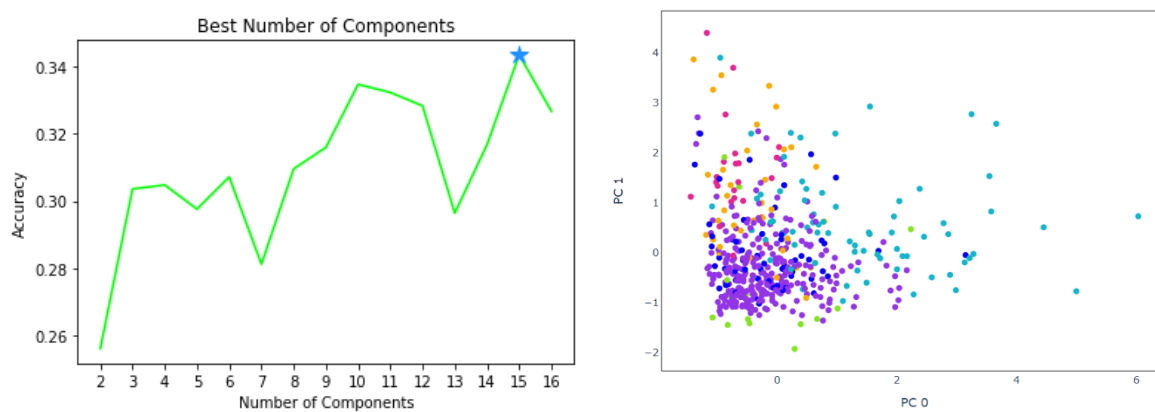


Figure 90: Finding the best number of principal components (left), and k-means predictions on the encoded mouse cortex data with standardization for 15 PCs (Clickable)

## K-Means and t-SNE

As k-means does not work well when clusters are not well separated, a new approach is needed. As briefly mentioned during the Literature Review, t-SNE is a technique that can find structure where other dimensionality reduction techniques fail [85]. Therefore, I tried applying t-SNE after dimensionality reduction with PCA. Again, different combinations of principal components were tested with the best result being the standardized, unencoded data with 33.2% accuracy (Figure 91). Unfortunately, this result was lower than without t-SNE. However, this technique does make it much clearer to view the cells in a lower dimension.

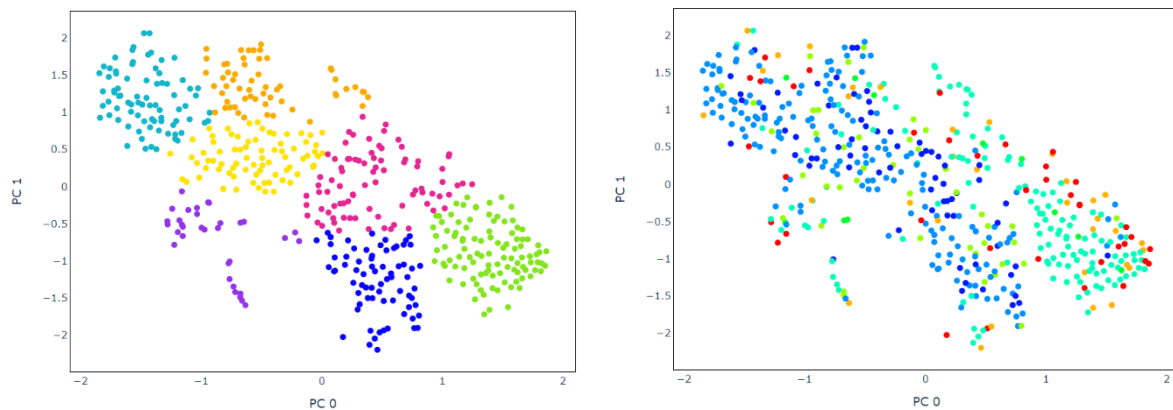


Figure 91: (Clickable) K-means predictions on the standardized, unencoded mouse cortex data with 16 principal components and t-SNE with perplexity 30

The most important parameter to tune for t-SNE is perplexity, which by default is set as 30. Decreasing this parameter will lead to fragmentation caused by local variation, while increasing it beyond the number of cells causes unanticipated behaviour [85] as demonstrated in Figure 92.

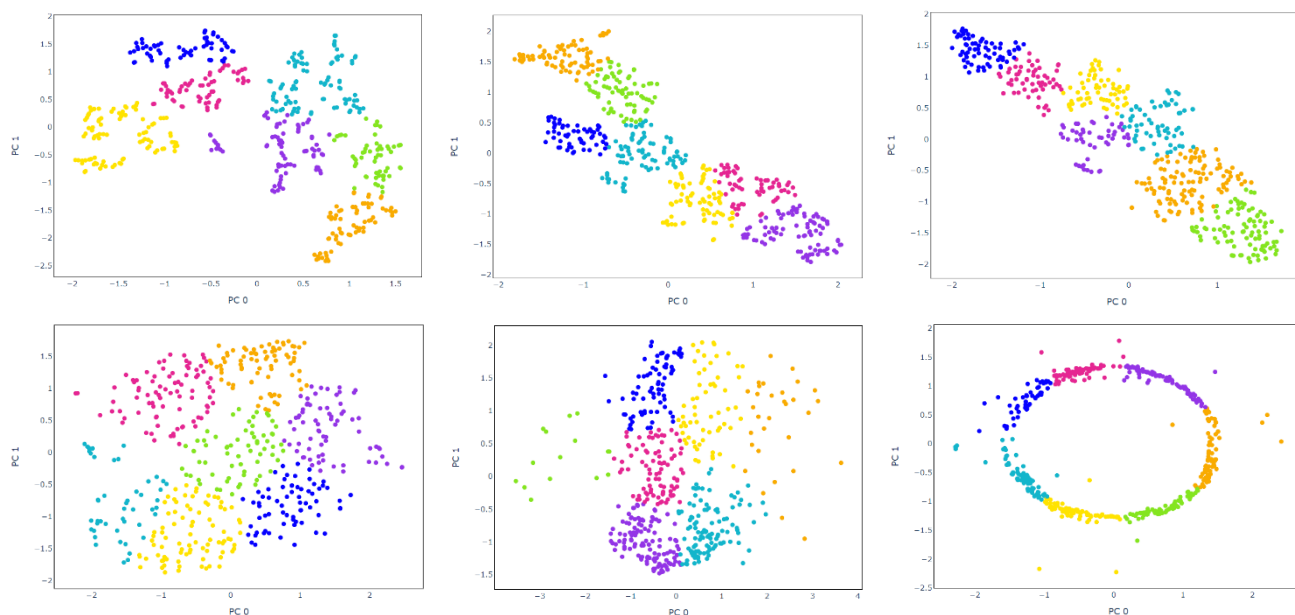


Figure 92: (Clickable) K-means on the standardized, unencoded mouse cortex data with 16 principal components and t-SNE perplexity 5, 10, 20 (top) and 50, 300, 600 (lower)

### Summary of K-Means on Mouse Cortex Data

K-means did not work well on this dataset as the cells are not separated enough. Applying t-SNE helped prevent the cells overlapping as much in the two-dimensional space and the best result overall was when using t-SNE with perplexity 300 achieving 37% accuracy. However, t-SNE is highly variable when run multiple times and accuracy is low meaning it was unable to replicate the BackSPIN labels. See Appendix B for the full table of results.

### 6.4.3 Agglomerative Hierarchical Clustering

Now a second clustering algorithm, agglomerative hierarchical clustering is applied to see how the results compare to k-means. These experiments have not been documented as thoroughly, but full tables of results can be found in Appendix B.

### 6.4.3.1 Benchmarking Data

#### Dendrogram

To find the recommended number of clusters, a dendrogram was plotted and the number determined using the method explained in the Literature Review. For the benchmarking data, this was three as represented by the dashed line on Figure 93.

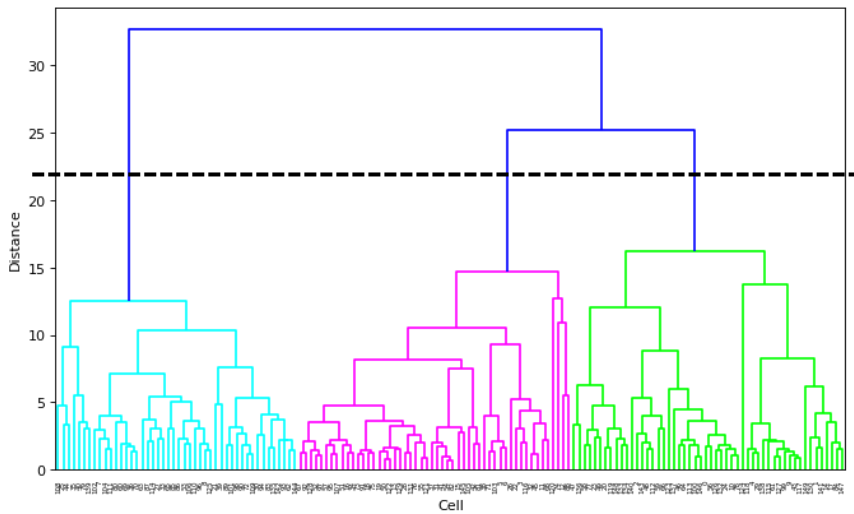


Figure 93: Dendrogram for benchmarking data agglomerative hierarchical clustering

#### Hierarchical Clustering with PCA

PCA was run on both the encoded and uncoded data, with and without applying standardized. The best result reached 100% accuracy through applying standardization to the uncoded data with 3 principal components (Figure 98). This is the same combination that achieved the best performance when using k-means.

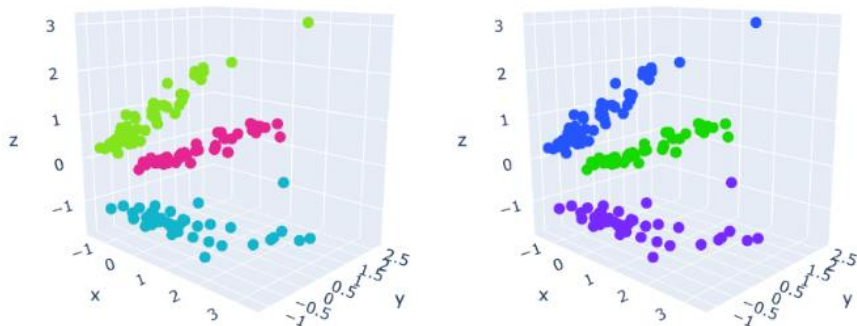


Figure 94: (Clickable) Hierarchical clustering prediction (left) and actual clusters (right) for 3 PC standardized uncoded benchmarking data

The highest accuracy on the encoded data was 99.3% with 2 principal components when applying standardization (Figure 95). Again, like in the k-means experiments, the encoding was unable to identify CELL\_000061. This could reflect a limitation of the autoencoder as perhaps it was not able to capture this cell's gene expression. However, this is the best result so far using only 2 principal components for this dataset.



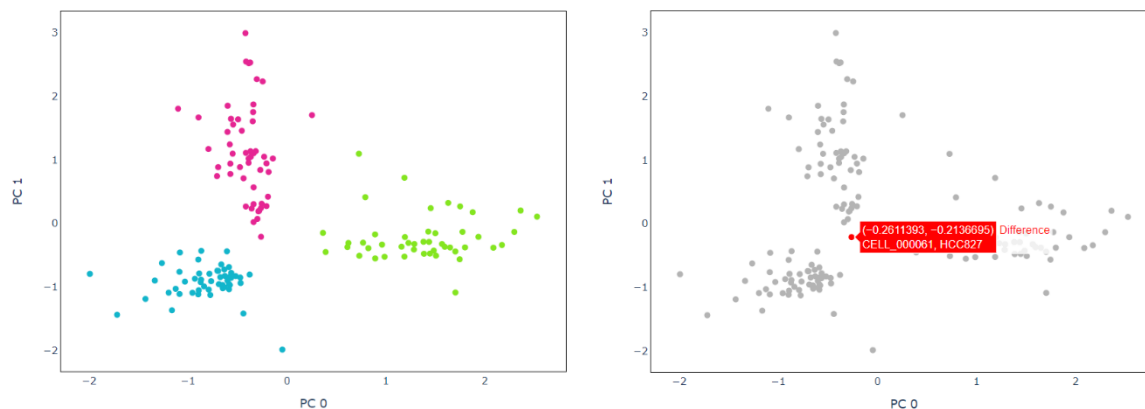


Figure 95: (Clickable) Hierarchical clustering prediction (left) and incorrect cell (right) for 2 PC standardized encoded benchmarking data

### 6.4.3.2 Simulated Data

#### Dendrogram

For the simulated data, the dendrogram dataset correctly predicted four as the number of clusters, as demonstrated by the dotted line on Figure 96.

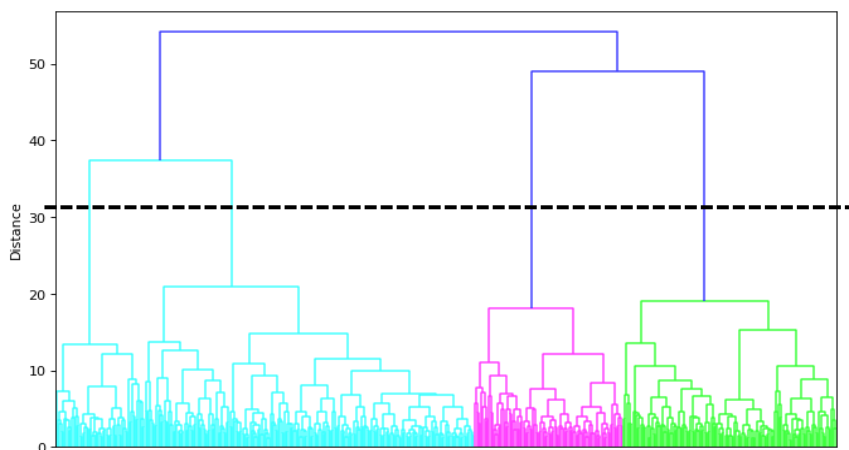


Figure 96: Dendrogram for simulated data agglomerative hierarchical clustering

#### Hierarchical Clustering with PCA

The top result for hierarchical clustering on the simulated data was 99.8% with ARI 0.996. This was achieved both when using 6 principal components on the encoded data (Figure 97) and 9 with standardization. These combinations are the same as the best results in the k-means experiments, though performance is slightly worse.

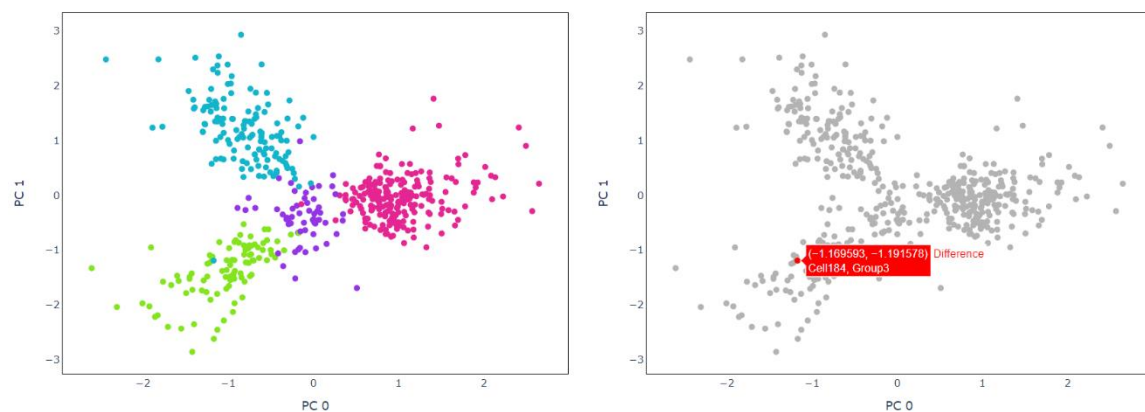


Figure 97: (Clickable) Hierarchical clustering prediction (left) and incorrect cell (right) for 6 PC encoded benchmarking data



### 6.4.3.3 Mouse Cortex Data

#### Dendrogram

The dendrogram implies that two is the best number of clusters. However, I again set the number as seven to compare against the BackSPIN labels.

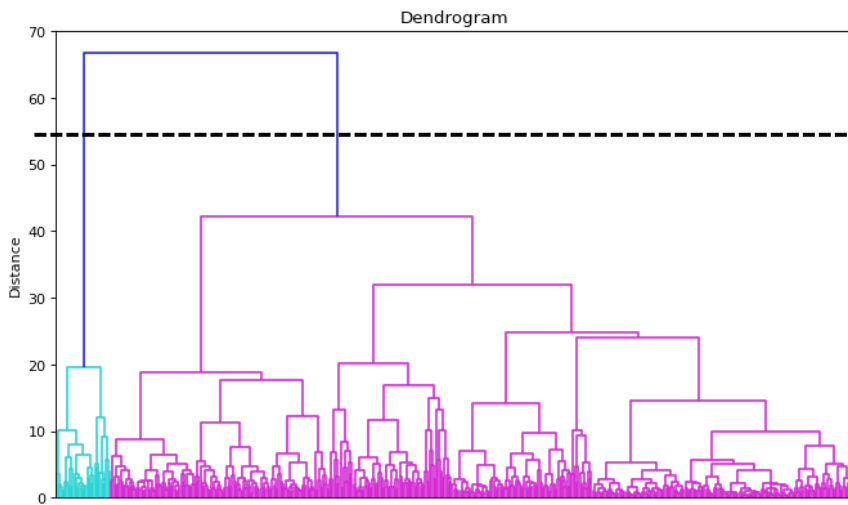


Figure 98: Dendrogram for mouse cortex data agglomerative hierarchical clustering

#### Hierarchical Clustering with PCA

Hierarchical clustering outperformed k-means achieving the highest so far accuracy at 44.4%. This was attained twice on the unencoded data using 9 principal components with standardization (Figure 99) and 15 without. Although it could be concluded that hierarchical clustering is therefore better suited to this type of dataset, it is hard to determine without a reliable ground truth as it is possible this is reflecting that the BackSPIN algorithm is more closely related to hierarchical clustering than a partitional algorithm such as k-means.

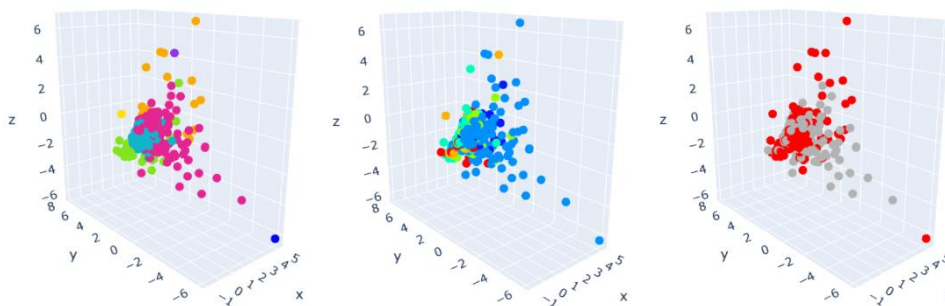


Figure 99: (Clickable) Hierarchical clustering prediction (left) actual clusters (middle) and incorrect predictions (right) for 9 PC standardized unencoded mouse cortex data

### 6.4.4 Visualising PCA Gene Expression

As PCA can be applied directly to the gene counts, I thought it would be interesting to view the contribution of each gene to the principal components for each dataset.

#### 6.4.4.1 Benchmarking Data

To do this I calculated the eigenvectors of the top 20 principal components. These are one-dimensional vectors consisting of loading scores between -1 to 1 for each gene. A value of 0 signifies that a gene does not contribute to the component, while a high absolute value such as  $|0.9|$  or  $|-0.9|$  indicate a large contribution [86]. The first 20 loading scores were then plotted against their corresponding genes as demonstrated in Figure 100.

This revealed that when using PCA on the raw data, most genes contribute very little to each principal component. I hypothesise this is because most genes do not provide significant variation compared to those with high expression. Interestingly after applying standardization, the gene contribution of each component was far

more evenly distributed. This may play a part in why standardization generally improves performance as the variation within the data is enhanced.

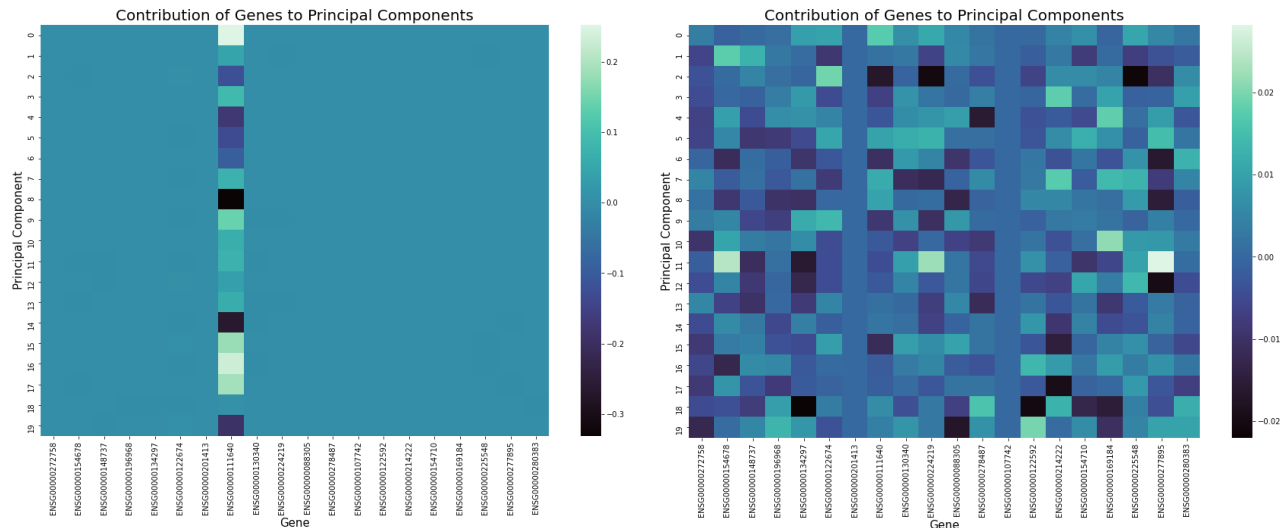


Figure 100: Loading scores for the top 20 principal components against 20 genes for the unstandardized (left) and standardized (right) benchmarking data

The top three principal components account for 67.2% of the variation of the original gene counts, each capturing 41.2%, 15.9% and 10.1% respectively. To find which genes have the greatest influence, I calculated the absolute values of the loading scores for these components. Ordering them from highest to lowest revealed that ENSG00000132432, a gene known as SEC61G that encodes a subunit protein for polypeptide transport across the endoplasmic reticulum [87], had the largest scores for all three. This suggests that expression of this gene is a useful marker for genetic variation between cells in this dataset.

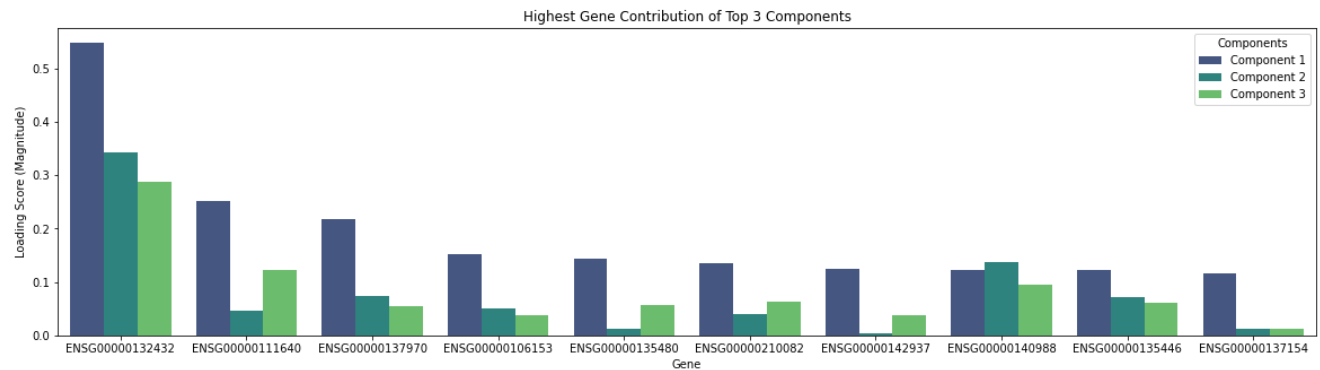


Figure 101: 10 genes with the highest loading scores for the first three principal components of the benchmarking data

#### 6.4.4.2 Simulated Data

Compared to the benchmarking and mouse cortex data (see section below), far more genes from the raw data significantly contribute to the top principal components. Again, when standardization is applied, gene contribution is increased, although this does not seem to have such a substantial effect. This is interesting as it reveals a difference between the biological and artificial data, perhaps because Splatter produces less extreme outliers than observed in real data.

The total variation of the top three principal components was similar to the benchmarking dataset at 66% with the first PC contributing 53.3%, and the second and third 8.1% and 4.5%. Overall, there was far more variation between genes with high loading scores for the top components (see Figure 103). For example, even though Gene13543 has the most significant contribution to the first PC, it does not contribute the most to the second or third. Regardless, this gene still provides the most variation overall as the first component is considerably more important than the others.

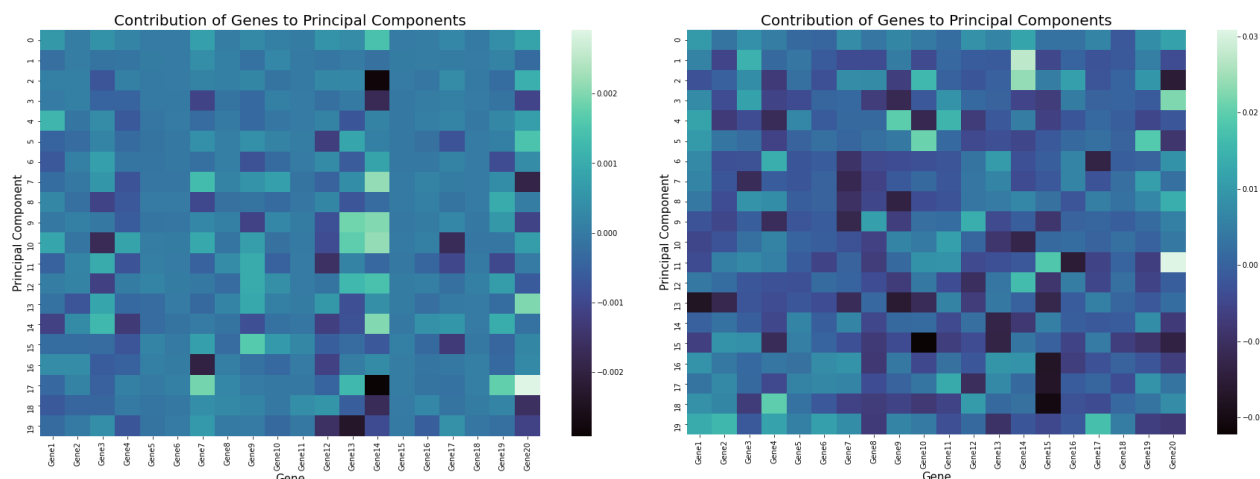


Figure 102: Loading scores for the top 20 principal components against 20 genes for the unstandardized (left) and standardized (right) simulated data

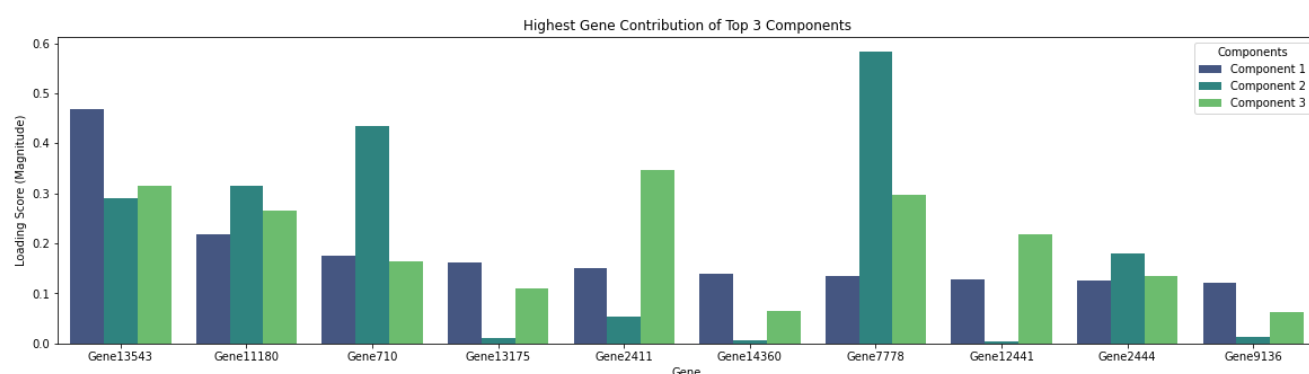


Figure 103: 10 genes with the highest loading scores for the first three principal components of the simulated data

#### 6.4.4.3 Mouse Cortex Data

Like the benchmarking dataset, without standardization only a minority of genes significantly contribute to the top principal components. While after applying standardization, the contribution is far more evenly distributed (see Figure 104). The top three principal components contain a much higher total variation of 92.9% compared to the other datasets, with the first supplying 80% and the following two 9.9% and 3%. The most representative gene was *r\_SSU-rRNA\_Hsa* (Figure 105), denoting the small subunit rRNA gene for has, followed by the large subunit rRNA gene for hsa. This result is not too surprising as ribosomes consist of two subunits and about 80 percent of RNA is ribosomal RNA [88].

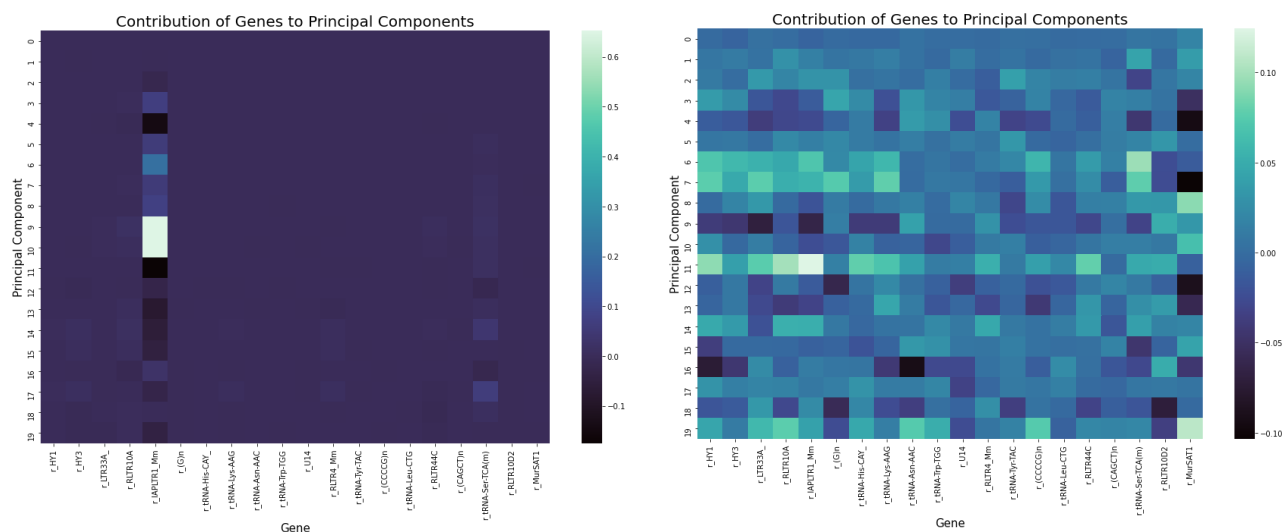


Figure 104: Loading scores for the top 20 principal components against 20 genes for the unstandardized (left) and standardized (right) mouse cortex data

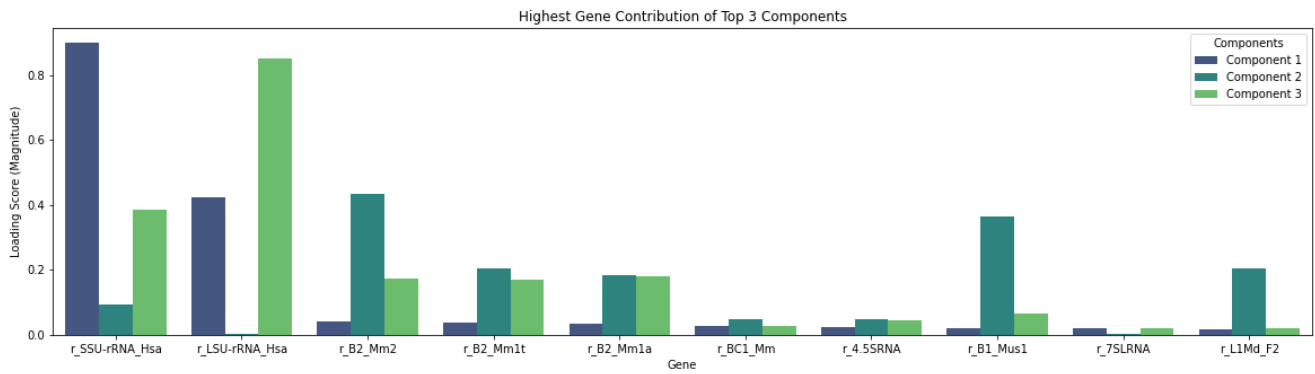


Figure 105: 10 genes with the highest loading scores for the first three principal components of the mouse cortex data

## 6.4.5 Alternative Algorithms

In the experiments above, k-means was able to correctly assign the cell lines / groups to all cells in the test sets for both the benchmarking and simulated dataset. Agglomerative hierarchical was also able to get a perfect score for the benchmarking data and got only one cell wrong for the simulated data. While these results are good, the experiments above were limited to only 150 and 500 samples respectively. Therefore, in this section, all 902 cells in the benchmarking data and 2000 in the simulated data will be tested against six different clustering algorithms and four dimensionality reduction techniques.

The full table of results for each experiment is available in Appendix B. Additionally, note that for some of the 2D graphs in this part, clicking on the figure will link to a 3D version as this better captures the shape of the clusters when applied to high dimensional data.

### 6.4.5.1 Benchmarking Data

#### Principal Component Analysis (PCA)

As in the previous experiments, PCA was run with between 2 to 20 components on both the encoded and original data with and without applying standardization. Each time, six different clustering algorithms were applied and the combination of data and parameters that achieved the highest accuracy were recorded and plotted in Figure 106. The new algorithms include: BIRCH, mini-batch k-means, spectral clustering and gaussian mixture.

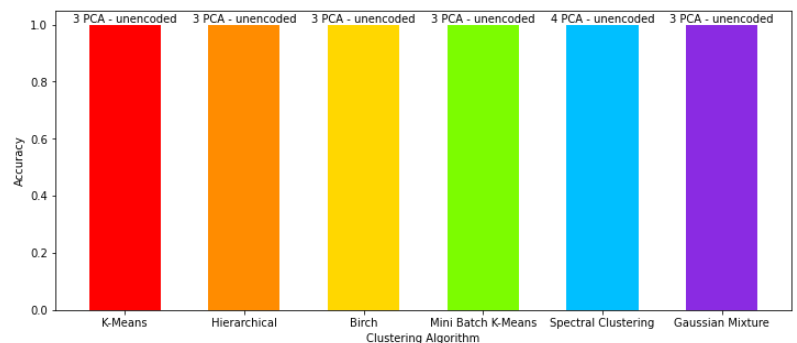


Figure 106: Histogram displaying best accuracy of the six clustering algorithms, along with the optimal number of principal components and whether the benchmarking data was encoded or not

Overall, every clustering algorithm was able to get at least 99.8% accuracy when applying standardization to the original gene count data. Both agglomerative hierarchical (shown in Figure 107) and BIRCH were able to achieve a 99.9% accuracy and ARI 0.997 using three principal components with only CELL\_000002 being clustered incorrectly. Spectral clustering gave the same results, although a fourth principal component was required.

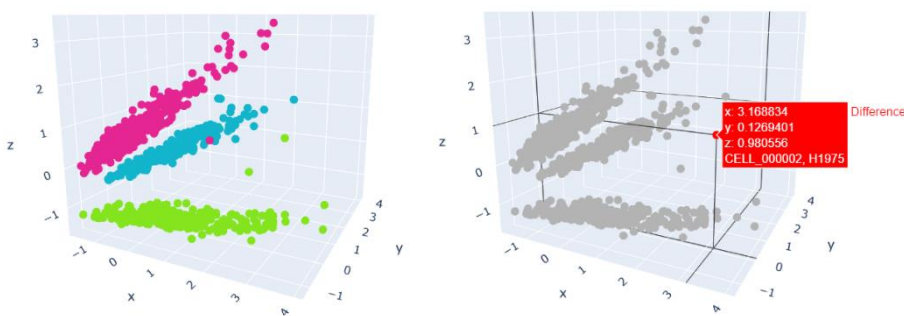


Figure 107: (Clickable) Hierarchical clustering prediction (left) and incorrect cell (right) for 3 PC standardized unencoded benchmarking data

## Independent Component Analysis (ICA)

ICA is an algorithm designed for separating signals from linearly mixed sources [89]. A study by Feng et al. [90] compared PCA, ICA and NMF for some scRNA-seq datasets and has shown that combining this technique with clustering can at times outperform PCA.

The results of ICA on this dataset are same as when using PCA (refer to Figure 110). Once again, CELL\_000002 was not able to be identified with agglomerative hierarchical (Figure 109), BIRCH and spectral clustering.

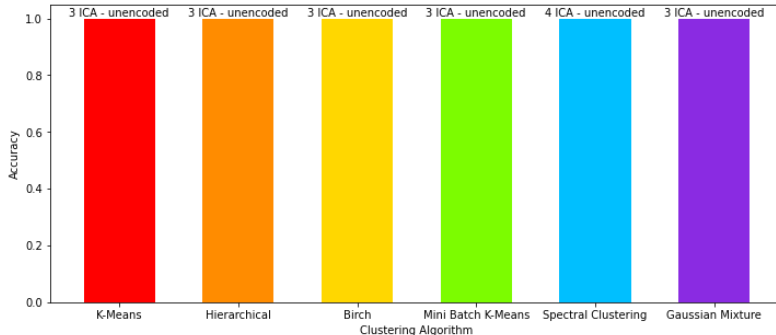


Figure 110: Histogram displaying best accuracy of the six clustering algorithms, along with the optimal number of independent components and whether the benchmarking data was encoded or not

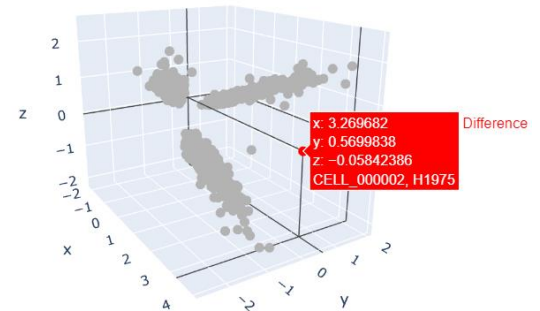


Figure 109: (Clickable) Hierarchical clustering prediction (left) and incorrect cell (right) for 3 IC standardized unencoded benchmarking data

## Non-negative matrix factorization (NMF)

NMF is a class of algorithms for feature extraction of a matrix  $X$  that finds two matrices  $W$  and  $H$  whose product approximates the original [91]. As this technique only works on non-negative data, a constant sometimes had to be added to all features of the data to prevent any values being negative.

Figure 108 suggests this technique seemed to work well with most clustering algorithms except spectral clustering, in which the best result assigning only one cell to two clusters and all others to a third cluster. It is also worth noting that this algorithm was far slower to run PCA or ICA.

The best documented result was k-means with two basis components on the unencoded, standardized data achieving a 99.9% accuracy. However, when the algorithm was run again to view the graph, this resulted in a lower accuracy of 87% and ARI 0.696. On inspection of Figure 111, it is obvious that the cell lines have not been separated enough for k-means to reliably cluster them.

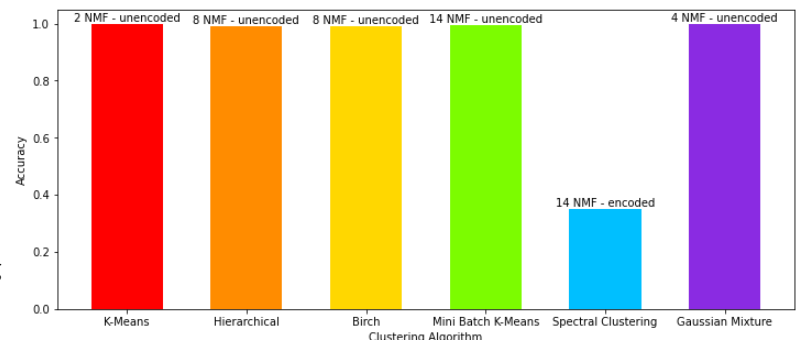


Figure 108: Histogram displaying best accuracy of the six clustering algorithms, along with the optimal number of NMF basis components and whether the benchmarking data was encoded or not

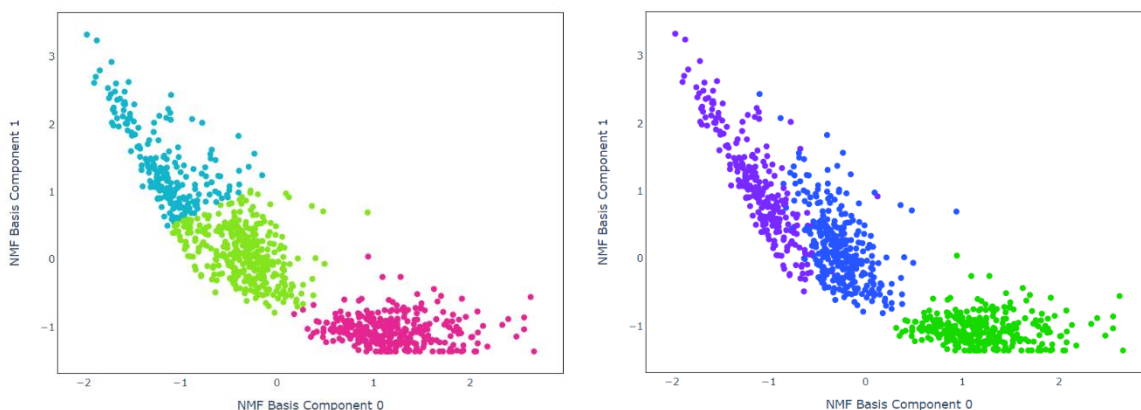


Figure 111: (Clickable) K-means prediction (left) and true cell lines (right) for 2 NMF basis components on standardized benchmarking data

The next best was Gaussian mixture with 4 basis components and standardization getting 99.8% accuracy and 0.993 ARI. Again CELL\_000002 was not identified, along with CELL\_000077, though this hard to see in the 2D version of the graph as the cells overlap.

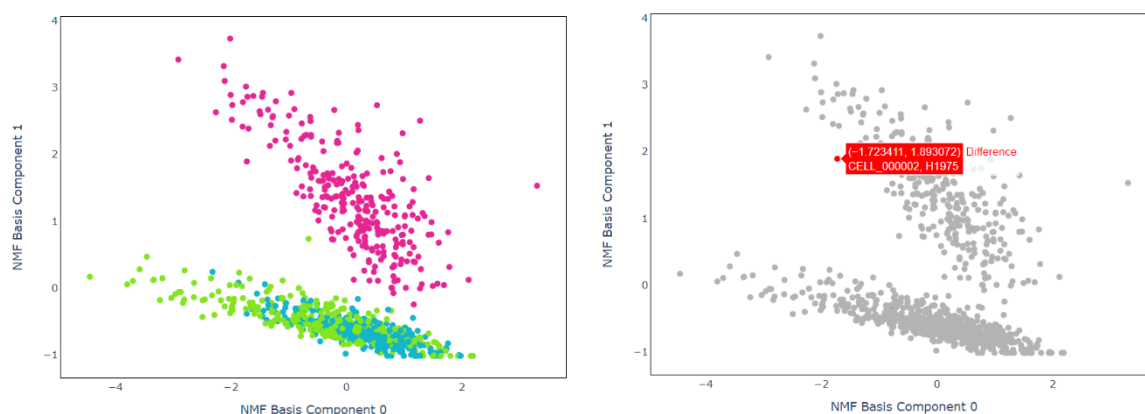


Figure 113: (Clickable) Gaussian mixture prediction (left) and identifying one of two incorrect cells (right) for 4 NMF basis components on standardized benchmarking data

## PCA with t-SNE

As PCA was the best technique, again different numbers of principal components were tested, but this time applied t-SNE before clustering. The best results were identical for all clustering techniques with 99.67% accuracy (Figure 112), with each performing best on the encoded data with standardization and two principal components.

When running mini batch k-means, it performed slightly worse than expected with 99% accuracy and 0.970 ARI. On observation, all clusters seem to be well defined and clearly separated. However, this is misleading as t-SNE has incorrectly grouped nine of the cells. Interestingly though, this is the only experiment to correctly cluster CELL\_000002. It is also the first in which the encoding outperformed the original data.

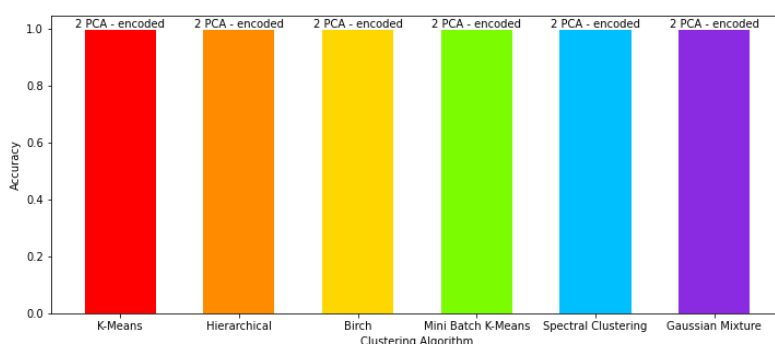


Figure 112: Histogram displaying best accuracy of the six clustering algorithms with t-SNE, along with the optimal number of PCA components and whether the benchmarking data was encoded or not

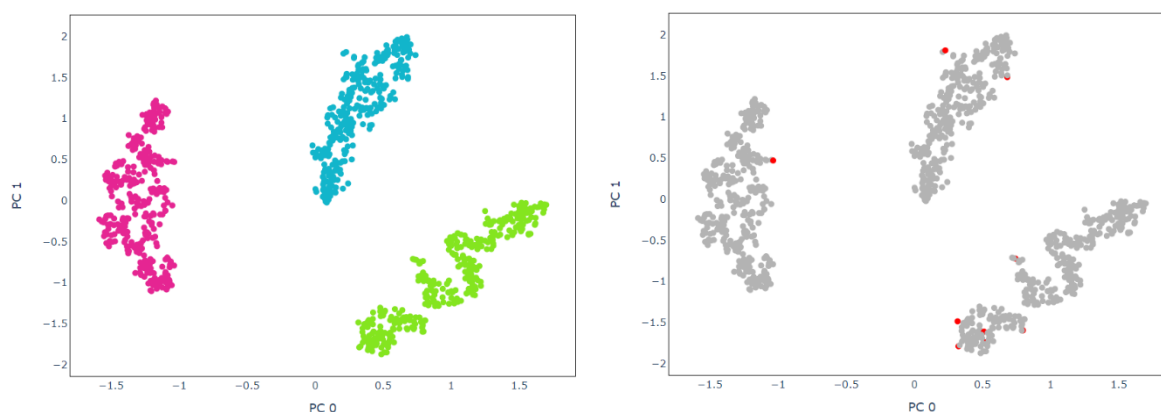


Figure 114: (Clickable) Mini batch k-means prediction (left) and incorrectly identified cells (right) for 2 principal components and t-SNE for the encoded, standardized benchmarking data

## Conclusion

The best dimensionality reduction techniques are PCA and ICA achieving 99.9% accuracy with agglomerative hierarchical, BIRCH and spectral clustering. NMF did not perform so well with poor results when combined with spectral clustering. Applying t-SNE led to a lower accuracy, however different samples were incorrectly clustered compared to when it is not used.



### 6.4.5.2 Simulated Data

#### Principal Component Analysis (PCA)

With PCA, k-means, agglomerative hierarchical, BIRCH and gaussian mixture were all able to identify the group for all 2000 cells. Figure 117 demonstrates this for BIRCH with four independent components. K-means was the only algorithm for which the encoding improved performance, although both it and gaussian mixture required far more principal components than either hierarchical or BIRCH. Spectral clustering was worse than the other algorithms as 39 of the cells were incorrect.

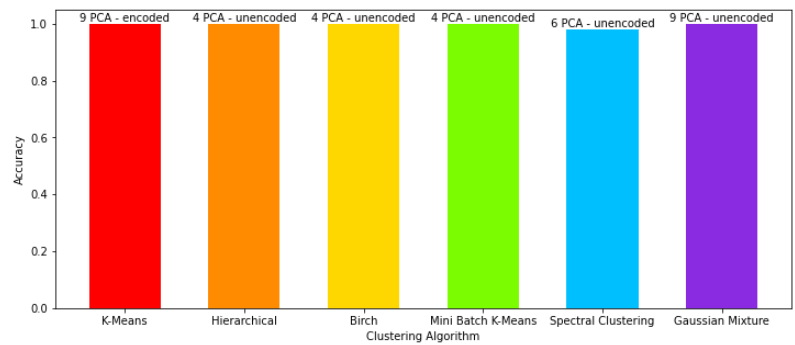


Figure 115: Histogram displaying best accuracy of the six clustering algorithms, along with the optimal number of principal components and whether the simulated data was encoded or not

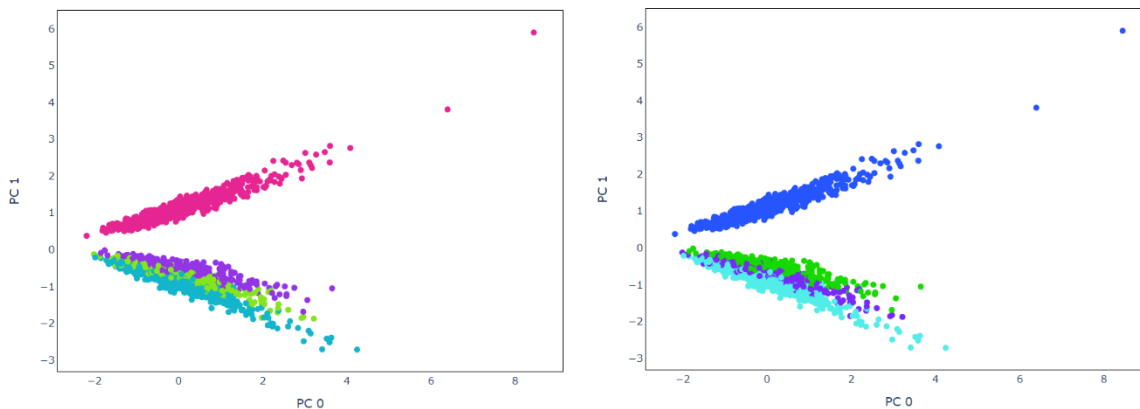


Figure 117: (Clickable) BIRCH prediction (left) and cell lines (right) for 4 principal components on standardized benchmarking data

#### Independent Component Analysis (ICA)

Again, ICA's results are very similar to PCA. The only difference is that mini-batch k-means required another component and was unable to identify three cells rather than two. Though it could be said that ICA is therefore worse than PCA for this dataset, when comparing BIRCH with four independent components (Figure 118) to the previous experiment (Figure 117), the groups seems better separated in both the 2D and 3D graphs. This would be beneficial if more samples were included or if there was more noise in gene expression.

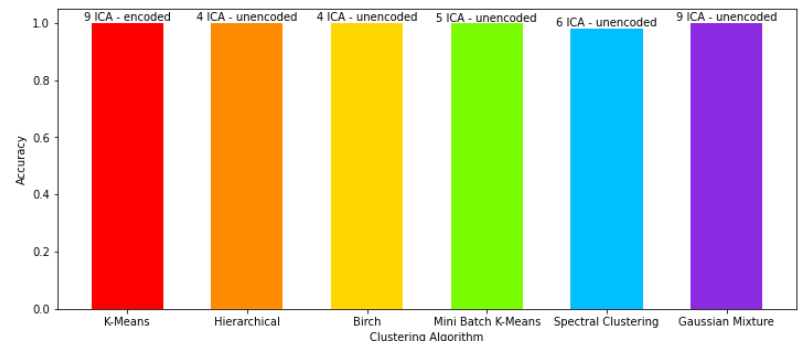


Figure 116: Histogram displaying best accuracy of the six clustering algorithms, along with the optimal number of independent components and whether the simulated data was encoded or not

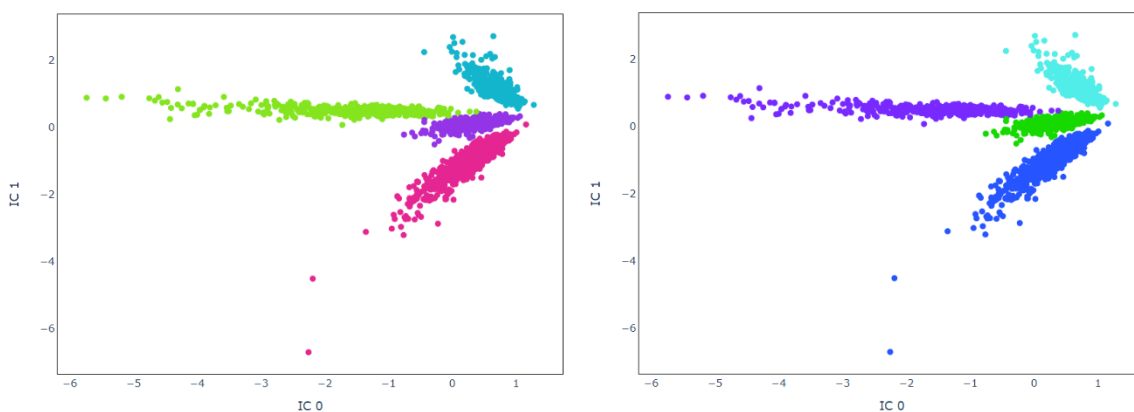


Figure 118: (Clickable) BIRCH prediction (left) and cell lines (right) for 4 independent components on standardized benchmarking data

## Non-negative matrix factorization (NMF)

The optimal results for each clustering algorithm with NMF were far more varied than with PCA and ICA. Both agglomerative hierarchical clustering and BIRCH outperformed the rest as their results were the same. This is probably because BIRCH is another form of hierarchical clustering. However, even the best performing algorithms with were not as good as the worst performance with either PCA or ICA.

Agglomerative hierarchical clustering with 11 basis components is displayed in Figure 120.

This received 85% accuracy and 0.857 ARI which unfortunately is lower than the documented best result of this algorithm at 92.2%.

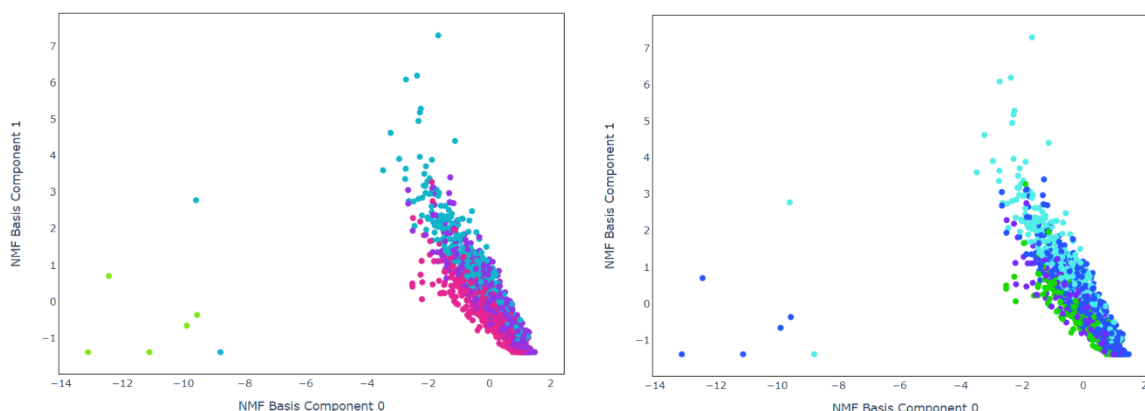


Figure 120: (Clickable) Agglomerative hierarchical prediction (left) and true cell lines (right) for 11 NMF basis components on unstandardized benchmarking data

## PCA with t-SNE

Combining PCA and t-SNE, every clustering algorithm was able to get 100% accuracy with two principal components and encoding. However, when running multiple times with this configuration, the result was not reproducible every time. This is because t-SNE can be highly variable even with the same parameters. For example, Figure 122 shows the outcome of spectral clustering with 2 principal components achieving 100% accuracy, while Figure 123 shows the same algorithms run again with just 50% accuracy.

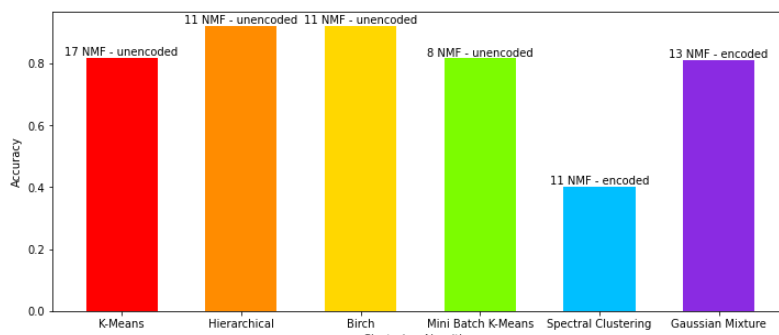


Figure 119: Histogram displaying best accuracy of the six clustering algorithms, along with the optimal number of NMF basis components and whether the simulated data was encoded or not

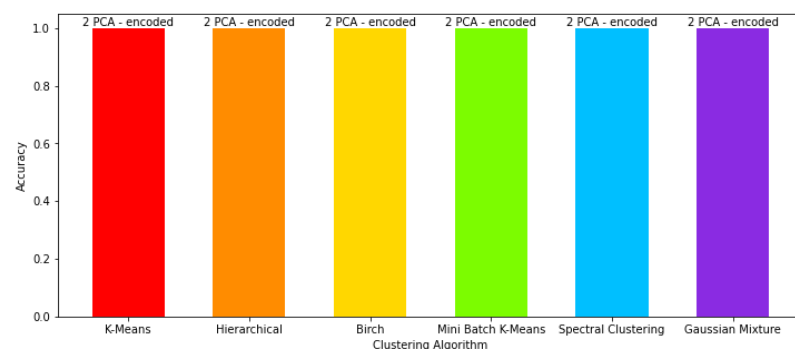


Figure 121: Histogram displaying best accuracy of the six clustering algorithms, along with the optimal number of t-SNE components and whether the simulated data was encoded or not



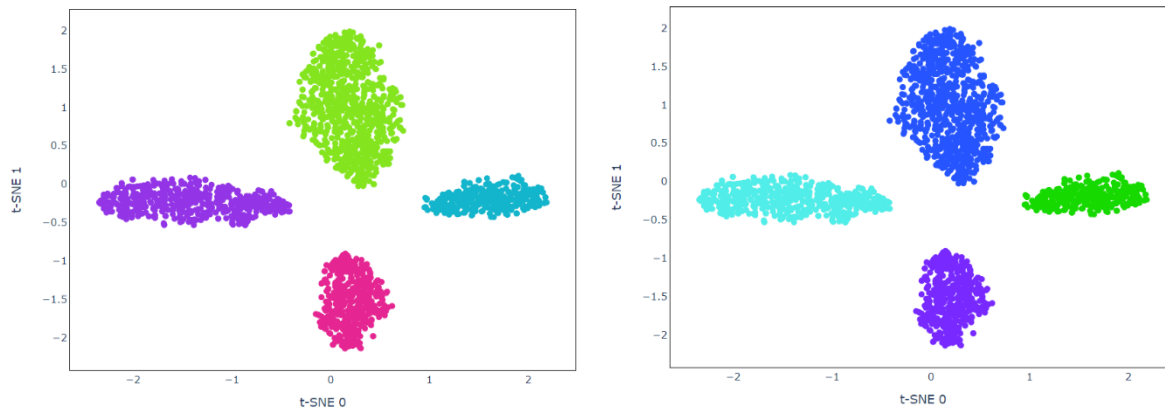


Figure 122: (Clickable) Spectral clustering prediction (left) and excepted groups (right) for 2 principal components with t-SNE for the encoded, unstandardized benchmarking data getting 100% accuracy

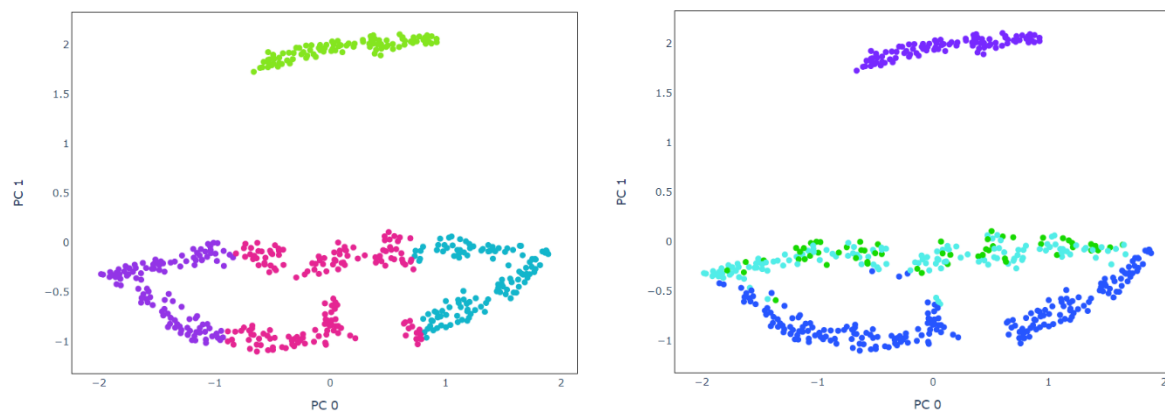


Figure 123: (Clickable) Spectral clustering prediction (left) and excepted groups (right) for 2 principal components with t-SNE for the encoded, unstandardized benchmarking data getting 50% accuracy

## Conclusion

Again, both PCA and ICA performed best with each able to attain 100% with multiple clustering algorithms. Although applying t-SNE will sometimes separate the groups, it is highly variable across runs and not reliable.

### 6.4.6 Biclustering

A limitation of standard clustering algorithms like k-means and hierarchical is that they cluster genes on the assumption that all genes show similar behaviour in any condition. However, some genes are co-expressed meaning they will show a similar expression pattern to other genes under certain experimental conditions but not under others. Biclustering is a type of clustering algorithm that clusters both cells and genes simultaneously therefore making it suitable for detecting co-expression [92].

#### 6.4.6.1 Mouse Cortex Data

In the study by Zeisel et al. [49] in which the mouse cortex dataset was created, they determined that biclustering would be a more suitable method of separating this dataset than hierarchical clustering. This is backed up by my findings in the experiments above as there does not seem to be a clear way to separate the data with traditional clustering. Therefore, I decided to try spectral biclustering with  $6 \times 6$  clusters on the gene counts of the test data (right of Figure 125) and plot the clusters against the original gene expression matrix (left of Figure 125).

The unclustered expression matrix was dominated by counts from a single gene, MER115, uncovering that many cells show a far stronger expression of this gene than any other. Note that although the name of this gene is not visible on the axis, it can be seen as a thin coloured line on the right side of the gene expression matrix. Biclustering caused the order of the cells and genes to be rearranged, though the graph looked similar to the original gene expression matrix as MER115 still dominated. To make the results clearer, I plotted a graph

identifying the region of each bicluster. This revealed that there are several small sub-groups of genes with similar yet unusual patterns of co-expression.

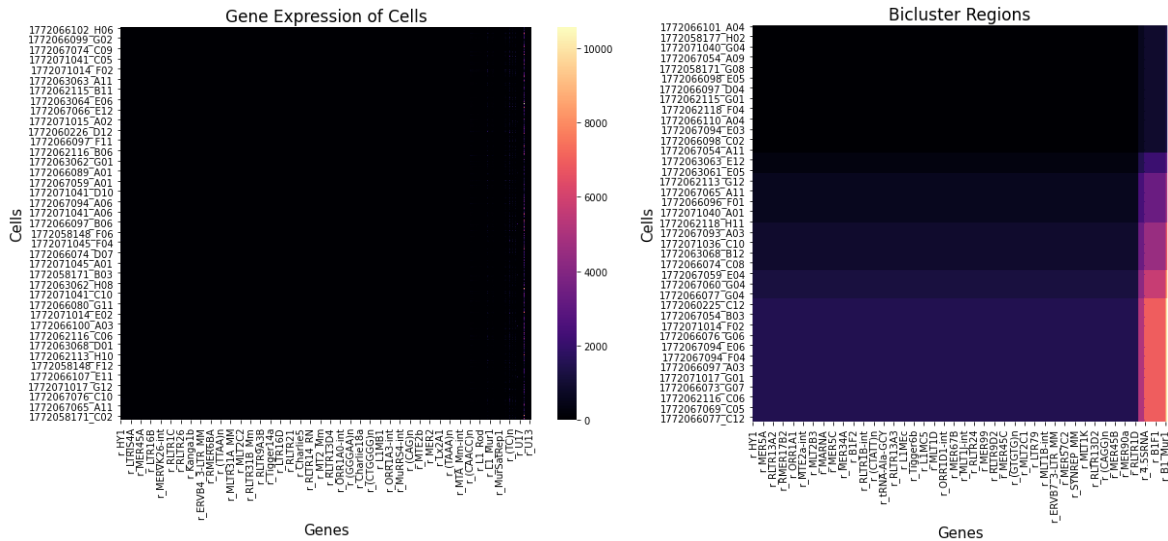


Figure 125: The original gene expression of the mouse cortex data (left) and bicluster regions (right)

Whilst the biclustering above reflects true gene expression, biclusters are not well distributed which may not be useful if using biclustering as a tool to separate cells and genes into regions that can be further clustered. Therefore, I repeated the experiment after applying standardization. This provides a better visualisation of how the gene expression matrix is rearranged (Figure 124) and produces more evenly sized biclusters (Figure 126).

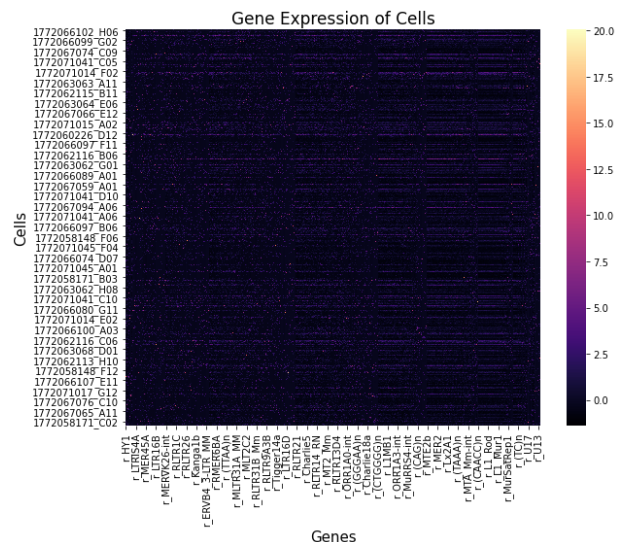


Figure 124: Gene expression of the standardized mouse cortex data

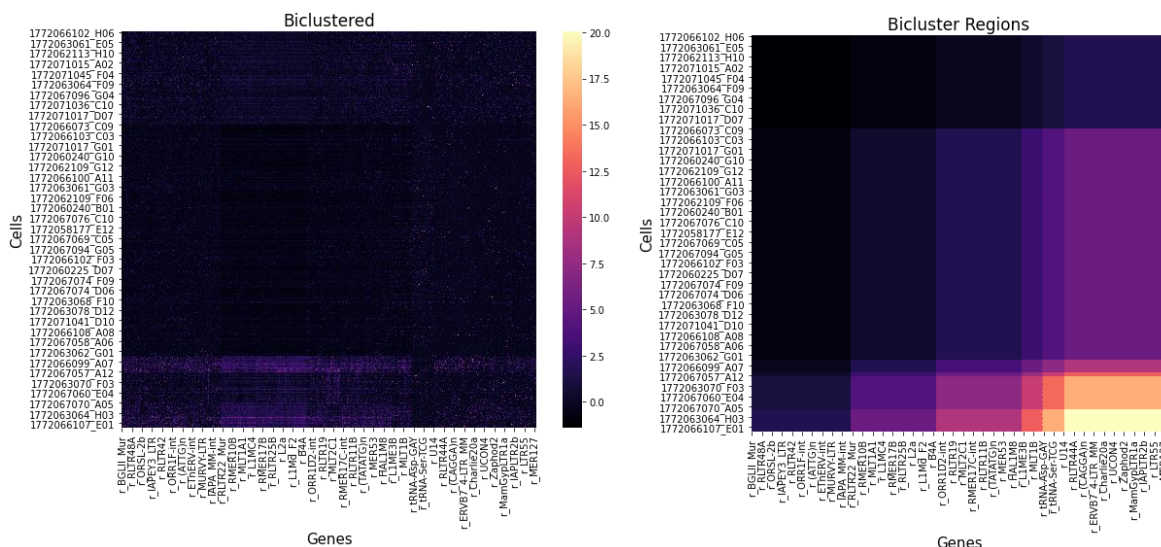


Figure 126: Rearrangement of the matrix after biclustering (left) and identification of the bicluster regions (right) for the standardized mouse cortex data

## 6.5 Topological Data Analysis

To meet objective 6, Kepler Mapper was run on the datasets to visualise their high-dimensional shapes. First each dataset was projected onto a lower-dimensional space using 2 principal components, then a simplicial complex constructed using 10 hypercubes with 0.15 overlap and k-means clustering. The selection of these parameters is explained during Implementation and they have been kept consistent so that each dataset can be directly compared. Again, links to interactive versions of the graphs have been included by clicking the figures.

### 6.5.1 Benchmarking Data

The simplicial complex of the benchmarking dataset is below in Figure 127. Most nodes are connected suggesting that most cells have some correlation in gene expression. Even so, the cell lines can still be identified as the three most connected regions. In addition, there are also several nodes that are not joined to the main complex which represent clusters of cells with more unusual gene expression. However as mentioned during Implementation, if the overlap  $\epsilon$  was increased then some of these nodes would join the main complex, while if it was decreased then more of these distinct nodes would form.

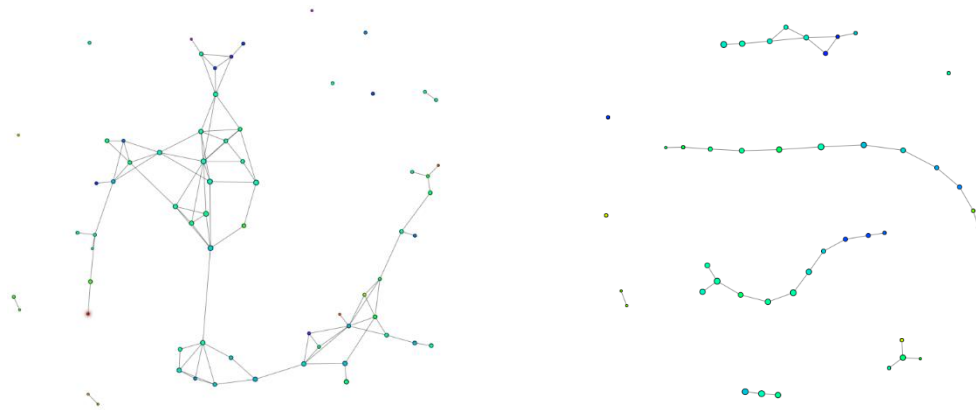


Figure 127: (Clickable) Simplicial complex of benchmarking data (left) and simulated data (right)

### 6.5.2 Simulated Data

Compared to the benchmarking data, the groups of the simulated data are far more clearly defined and separated from one another (see Figure 127). This suggests that gene expression between the groups do not overlap as much as true biological data. Furthermore, the groups themselves form straight lines indicating that there is a lot less variation within each of them.

### 6.5.3 Mouse Cortex Data

The simplicial complex of this dataset (Figure 128) is far more complicated than the others reflecting the greater variation of gene expression within the mouse cortex and hippocampus. As seen during Clustering, the groups are not well defined, though this graph implies most cells can be split into two distinct regions. There are also far more individual nodes compared to the other datasets, though this is likely because this dataset contains the most cells.

### 6.5.4 Comparing TDA to Clustering

The graphs produced by Mapper provide a better overview of the whole datasets than the lower dimensional graphs created during Clustering. With hindsight, TDA would be a beneficial tool to use to inspect a dataset before clustering to give insight into how to approach the problem. For example, perhaps it would have been better if the mouse cortex dataset was first split into two sub-groups before clustering.



Figure 128: (Clickable) Simplicial complex of the mouse cortex data

## 7. Conclusions and Future Work

This is the final section of the report consisting of a general overview of the project, evaluation against aims and objectives, findings and conclusions from the results, proposals for future work and the final statement.

### 7.1 Overview

The aim set at the start is to detect hidden gene expression in single-cell RNA-sequencing data. The focus has been around exploring pre-processing techniques to improve performance of clustering cells with similar gene expression for three different datasets.

An optimised autoencoder was created to encode each dataset to capture key features in a lower dimensional space. Standardization and PCA were applied to both the original and encoded data before running k-means and agglomerative hierarchical clustering. The effectiveness of each technique was thoroughly assessed by testing different combinations and parameters. Other algorithms briefly explored include: ICA and NMF for dimensionality reduction, BIRCH, mini batch k-means, spectral clustering, and Gaussian mixture clustering algorithms and spectral biclustering. Finally, for each dataset a simplicial complex was constructed using Mapper to view the topology of the higher dimensional space of the data.

### 7.2 Evaluation Against Objectives

Overall, the project has been successful with most sub-objectives fulfilled, all key milestones met by their deadlines, as well as experimentation with many techniques not originally proposed. The objectives and sub-objectives, detailed earlier in Requirements and Specification, are listed in Table 19. Achieved sub-objectives are coloured green, while those not fulfilled are red. For each, an explanation has been written justifying either how the objective was met or why it was not satisfied.

ID	Objective	Sub-objective	Justification
1.1	Research and understand single-cell RNA-seq and the resulting datasets	Review multiple sources, including papers and articles on single-cell RNA-seq	At the start of the project, I performed detailed background research on the problem area and selected three different types of dataset as documented in the Literature Review.
1.2		Write a summary of the technical details of RNA-seq and challenges with the type of data in the Literature Review	
1.3		Select datasets for benchmarking and evaluation	
2.1	Review literature for data encoding techniques, neural network optimisation methods, clustering methods and topological data analysis	Research and compare at least two data encoding methods in the Literature Review	Early on I researched PCA and autoencoders and documented them in the Literature Review. Though these were my initial focus, later other dimensionality reduction methods were explored briefly during Clustering. Several techniques for optimising the autoencoder were studied in both the Literature Review and System and Experiment Design, along with k-means, hierarchical clustering and constructing simplicial complexes with Mapper.
2.2		Research neural network optimisation techniques and write a summary in Literature Review	
2.3		Research and compare at least two clustering methods in the Literature Review	
2.4		Research topological data analysis and write a summary in the Literature Review	
3.1	Implement an autoencoder that can extract a lower dimensional representation of biological data	Split dataset into testing and training data	After opening the selected dataset, samples are batched and divided into testing, training, and validation. The autoencoder was implemented as a PyTorch Lightning class, initially with the architecture set out in System and Experiment Design. This model can take batched cells as input and return both encoded and decoded representations.
3.2		Implement an encoder that can produce a compressed version of input data with less features	
3.3		Implement a decoder that can reconstruct the compressed data so that it has the same number of features as the input data	
4.1	Test and experiment with different neural network architectures and	Experiment with different learning rates and update the autoencoder to use the optimum	To optimise the autoencoder, several different learning rates, batch sizes, optimisers, number of hidden layers, activation functions and epochs were tested as documented in Testing and Validation. The
4.2		Experiment with different batch sizes and update the autoencoder to use the optimum	

4.3	apply optimisation techniques to the autoencoder to determine how these change the quality of the encoding	Experiment with different optimisers and select the best for the autoencoder	optimal parameters were then set and autoencoders trained and save to file for each dataset. Unfortunately, not all sub-objectives were met as only the autoencoder of the mouse cortex dataset was able to surpass 50% accuracy when recreating the gene counts from the encoding and variational autoencoders were not attempted.
4.4		Experiment with different numbers of hidden layers and features to optimise the structure of the autoencoder	
4.5		Improve decoded output so it produces 50% or less error	
4.6		Train the optimised autoencoder	
4.7		Convert autoencoder to VAE autoencoder	
5.1	Apply clustering methods to the encoded data	Run the trained autoencoder on test data to retrieve the encoding	Trained autoencoders for each dataset were loaded from file to encode the gene counts. Both k-means and hierarchical clustering were performed on the encoded and original data with PCA, and the results recorded in System and Experiment Design. In addition, alternative clustering algorithms BIRCH, mini batch k-means, spectral clustering, Gaussian mixture were tested in combination with new dimensionality reduction techniques independent component analysis and non-negative matrix factorisation. t-SNE and spectral biclustering were also covered.
5.2		Perform k-means clustering on encoded data	
5.3		Perform agglomerative hierarchical clustering on encoded data	
5.4		Perform clustering on secondary datasets	
6.1	Apply a topological data analysis method to the data	Create a simplicial complex from the first dataset	Kepler Mapper was run to produce a simplicial complex for each dataset. These can be seen in System and Experiment Design, while parameter selection is detailed in Implementation.
6.2		Plot the simplicial complex	
6.3		Perform TDA on secondary datasets	
7.1	Evaluate the performance of topological data analysis in comparison to clustering methods	Document clustering results in report	System and Experiment Design documents experiments from both clustering and TDA. A brief comparison between the two methods is written at the end of this section.
7.2		Document TDA results in report	
7.3		Compare the results from the two methods and explain findings	
8.1	Review the results of the project and recommend future improvements	Evaluate project against aims and objectives	The project has been evaluated against objectives in this section. The overall results, improvements and considerations are detailed below in under Results and Recommendations and Future Work.
8.2		Suggest how the project could have been improved with hindsight and how it could be continued in the future	

Table 19: Evaluation of objectives and sub-objectives

## 7.3 Results and Recommendations

I achieved state-of-the-art results for two out of three datasets. For the first, sc\_10x, I identified all three cell lines with 99.9% accuracy through a combination of standardization, PCA or ICA with three components and BIRCH or agglomerative hierarchical clustering. These results outperformed experiments by the dataset's creators L. Tian et al. [79]. The second was a simulated dataset created during this project using Splat that comprises four sub-populations of differently expressed genes. For this I accomplished 100% accuracy using PCA and ICA with multiple clustering algorithms.

Unfortunately, I was not able to get good results for the third dataset, mouse cortex mRNA. Unlike the others, there is a lot of overlap in genetic expression between cells and so the techniques attempted were not effective at separating them into distinct regions for clustering. Therefore, for this dataset, a different approach should be adopted than that tested in this project.

I have learnt a lot and therefore have several recommendations for future researchers:

- ◆ **Topological Data Analysis** –TDA gives a high-dimensional overview of a dataset and good insight into relationships in gene expression between cells. Though I applied Mapper at the end, it would be beneficial to



run this at the start of a project to visualise the structure of a new dataset as this can help determine the type of approach required.

- ◆ **Autoencoders** – Autoencoders are used for noise reduction and dimensionality reduction by compressing gene expression into a lower-dimensional space. The best clustering results for the first dataset were attained without encoding, while top results were achieved both with and without encoding the simulated data. Effectiveness is therefore dependent on the data, though it is always worth testing multiple approaches.
- ◆ **Standardization** – Applying standardization before dimensionality reduction can often improve results by increasing the separation of the clusters.
- ◆ **Feature optimisation** – Testing different number of components is valuable when using dimensionality reduction techniques such as PCA or ICA. If the value is too low, too little variation will be captured for the cells to be separated, while increasing it too much will introduce noise which can also be detrimental. Whilst I did not test different feature numbers for the autoencoder's encoding, I imagine this would be applicable too.
- ◆ **t-SNE** – This is a good option to try if cells are not separable by other methods, however, it should be used with caution as it can produce misleading results. If used, different perplexity values should be tested as changing this parameter can cause results to vary significantly.

## 7.4 Future Work

Throughout the project, I encountered many different techniques and approaches that could be worth investigating. However, due to limited scope and time constraints, it was not possible for each to be attempted. As a result, there are many ways in which this research could be improved or extended including:

- ◆ **Variational autoencoders** – Compared to deterministic autoencoders, variational autoencoders give greater control over how the latent distribution is modelled. This means that a VAE may be able to capture a better representation of the data resulting in a more precise encoding [93].
- ◆ **Data simulators** – To generate the simulated dataset, I used the Splatter simulator which is part of the R package Splatter. However, as explained during the Literature Review, this package also supports several other simulation tools and the ability of each to imitate a dataset is highly variable. Several of the experiments conducted in this project revealed obvious differences between the simulated and real datasets. Therefore, it would be worthwhile to try different simulators to create more biologically realistic data.
- ◆ **Normalisation** – When collecting scRNA-seq data from an experiment, normalisation is an important step in which data is adjusted due to differences in experimental conditions. I did not attempt to apply normalisation as I used secondary data and did not feel it was applicable. However, with hindsight I think research into normalisation methods would have been beneficial as leaving this step out can lead to false differences in gene expression [94].
- ◆ **Ensemble clustering** – Combining different clustering algorithms and pre-processing methods as an ensemble may produce a better result than any one individual method. For example, the best result when clustering all cells of the benchmark dataset correctly identified the cell line of all but one sample. However, when t-SNE was applied, all incorrectly clustered samples were different. In theory, 100% accuracy could be achieved for this dataset through merging the results of these and other techniques.
- ◆ **Biclustering** – As explained in the paper by Zeisel et al. [49] and demonstrated in this project, the mouse cortex dataset is not suitable for separating by traditional clustering methods due to overlapping gene expression. Therefore, other methods are recommended such as biclustering to split cells and genes into sub-groups onto which clustering can be further applied. Although I did attempt spectral biclustering, I did not have time to thoroughly explore this method.

## 7.5 Final Statement

This project has been a great opportunity to apply knowledge learnt throughout my degree and to contribute research to a rapidly evolving field. Reflecting back to the start, I had little prior knowledge of the problem background, autoencoders or topological data analysis. Through extensive literature review and guidance from my sponsor I have now gained useful skills in these areas.

If I were to start the project again, an interesting approach would be to create a software application rather than Jupyter notebooks since this would be beneficial to biologists with little experience of coding. Another point is that I probably should have created more complicated simulated data. Despite using it for secondary testing, I found it easier to separate than the original biological data.

## References

- [1] British Computing Society, "BCS Code of Conduct," 2020. [Online]. Available: <https://www.bcs.org/membership/become-a-member/bcs-code-of-conduct/>.
- [2] UK Government, "Computer Misuse Act 1990," 2021. [Online]. Available: <https://www.legislation.gov.uk/ukpga/1990/18/contents>.
- [3] U. Government, "The Data Protection Act," 2018. [Online]. Available: <https://www.gov.uk/data-protection>.
- [4] Information Commissioner's Office, "Special category data," 2018. [Online]. Available: <https://ico.org.uk/for-organisations/guide-to-data-protection/guide-to-the-general-data-protection-regulation-gdpr/lawful-basis-for-processing/special-category-data/>.
- [5] A. f. C. Machinery, "ACM Code of Ethics and Professional Conduct," 2018. [Online]. Available: <https://www.acm.org/code-of-ethics>.
- [6] Y. Zheng, "Chapter 6 - Identification of miRNA and siRNA Targets in Plants," in *Computational Non-coding RNA Biology*, 2019, pp. 177-205.
- [7] J. M. Churko, G. L. Mantalas, M. P. Snyder and J. C. Wu, "Overview of High Throughput Sequencing Technologies to Elucidate Molecular Pathways in Cardiovascular Diseases," PubMed Central, 2014.
- [8] GENEWIZ, "Single-Cell RNA Sequencing Frequently Asked Questions," 2020. [Online]. Available: <https://web.genewiz.com/single-cell-faq>.
- [9] A. Haque, J. Engel, S. A. Teichmann and T. Lönnerberg, "A practical guide to single-cell RNA-sequencing for biomedical research and clinical applications," BMC, 2017.
- [10] R. Mackenzie, "RNA-seq: Basics, Applications and Protocol," 2018. [Online]. Available: <https://www.technologynetworks.com/genomics/articles/rna-seq-basics-applications-and-protocol-299461>.
- [11] LC Sciences, "single-cell-analysis," 2020. [Online]. Available: <https://www.lcsciences.com/discovery/signle-cell-analysis/>.
- [12] Harvard Chan Bioinformatics Core (HBC), "Introduction to Single-cell RNA-seq," 2020. [Online]. Available: [https://hbctraining.github.io/scRNA-seq/lessons/O1\\_intro\\_to\\_scRNA-seq.html](https://hbctraining.github.io/scRNA-seq/lessons/O1_intro_to_scRNA-seq.html).
- [13] F. W. Townes, S. C. Hicks, M. J. Aryee and R. A. Irizarry, "Feature selection and dimension reduction for single-cell RNA-Seq based on a multinomial model," 2019.
- [14] J. Cadima and I. Jolliffe, "Principal component analysis: a review and recent developments," 2016. [Online]. Available: <https://royalsocietypublishing.org/doi/10.1098/rsta.2015.0202>.
- [15] D. Oehm, "PCA vs Autoencoders for Dimensionality Reduction," 2018. [Online]. Available: <http://gradientdescending.com/pca-vs-autoencoders-for-dimensionality-reduction/>.
- [16] J. Jordan, "Introduction to autoencoders," 2018. [Online]. Available: <https://www.jeremyjordan.me/autoencoders/>.
- [17] S. Kim, "Improved survival analysis by learning shared genomic information from pan-cancer data," 2020. [Online]. Available: <https://www.ncbi.nlm.nih.gov/pmc/articles/PMC7355236/>.
- [18] A. F. Agarap, "Implementing an Autoencoder in PyTorch," 2020. [Online]. Available: <https://medium.com/pytorch/implementing-an-autoencoder-in-pytorch-19baa22647d1>.
- [19] S. Flores, "Variational Autoencoders are Beautiful," 2019. [Online]. Available: <https://www.compthree.com/blog/autoencoder/>.
- [20] National Cancer Institute, "The Cancer Genome Atlas Program," 2019. [Online]. Available: <https://www.cancer.gov/about-nci/organization/ccg/research/structural-genomics/tcga>.
- [21] StackExchange, "What is the difference between model hyperparameters and model parameters?," 2016. [Online]. Available: <https://datascience.stackexchange.com/questions/14187/what-is-the-difference-between-model-hyperparameters-and-model-parameters>.



- [22] G. Liu, "Optimizing Neural Networks — Where to Start?," 2019. [Online]. Available: <https://towardsdatascience.com/optimizing-neural-networks-where-to-start-5a2ed38c8345>.
- [23] D. Mack, "How to pick the best learning rate for your machine learning project," 2018. [Online]. Available: <https://medium.com/octavian-ai/which-optimizer-and-learning-rate-should-i-use-for-deep-learning-5acb418f9b2>.
- [24] J. Jordan, "Setting the learning rate of your neural network.," 2018. [Online]. Available: <https://www.jeremyjordan.me/nn-learning-rate/>.
- [25] L. N. Smith, "Cyclical Learning Rates for Training Neural Networks".
- [26] P. Lê, "What's up with Deep Learning optimizers since Adam?," 2018. [Online]. Available: <https://medium.com/vitalify-asia/whats-up-with-deep-learning-optimizers-since-adam-5c1d862b9db0>.
- [27] J. Brownlee, "How to Control the Stability of Training Neural Networks With the Batch Size," 2019. [Online]. Available: <https://machinelearningmastery.com/how-to-control-the-speed-and-stability-of-training-neural-networks-with-gradient-descent-batch-size/>.
- [28] M. Peixeiro, "The 3 Best Optimization Methods in Neural Networks," 2019. [Online]. Available: <https://towardsdatascience.com/the-3-best-optimization-methods-in-neural-networks-40879c887873>.
- [29] Y. Bengio, "Practical recommendations for gradient-based training of deep architectures," 2012.
- [30] J. Brownlee, "Difference Between a Batch and an Epoch in a Neural Network," 2018. [Online]. Available: <https://machinelearningmastery.com/difference-between-a-batch-and-an-epoch/>.
- [31] J. Brownlee, "Overfitting and Underfitting With Machine Learning Algorithms," 2016. [Online]. Available: <https://machinelearningmastery.com/overfitting-and-underfitting-with-machine-learning-algorithms/>.
- [32] Programmer Sought, "[deep learning] keras' EarlyStopping usage and tips," 2018. [Online]. Available: <https://www.programmersought.com/article/8892539067/>.
- [33] S. Kaushik, "An Introduction to Clustering and different methods of clustering," 2016. [Online]. Available: <https://www.analyticsvidhya.com/blog/2016/11/an-introduction-to-clustering-and-different-methods-of-clustering/#:~:text=Clustering%20is%20the%20task%20of,and%20assign%20them%20into%20clusters..>
- [34] I. Jamail and A. Moussa, "Current State-of-the-Art of Clustering Methods for Gene Expression Data with RNA-Seq," IntechOpen, 2020.
- [35] T. A. Geddes, T. Kim, L. Nan, J. G. Burchfield, J. Y. H. Yang, D. Tao and P. Yang, "Autoencoder-based cluster ensembles for single-cell RNA-seq data analysis," BMC.
- [36] K. Arvai, "K-Means Clustering in Python: A Practical Guide," 2020. [Online]. Available: <https://realpython.com/k-means-clustering-python/>.
- [37] B. Zhao, A. Erwin and B. Xue, "How many differentially expressed genes: A perspective from the comparison of genotypic and phenotypic distances," ScienceDirect.
- [38] C. M. Koch, S. F. Chiu, M. Akbarpour, B. Ankit, K. M. Ridge, E. T. Bartom and D. R. Winter, "A Beginner's Guide to Analysis of RNA Sequencing Data," ATS Journals, 2018.
- [39] Z. Singer, "Topological Data Analysis — Unpacking the Buzzword," 2019. [Online]. Available: <https://towardsdatascience.com/topological-data-analysis-unpacking-the-buzzword-2fab3bb63120>.
- [40] E. Munch, "A User's Guide to Topological Data Analysis," *Journal of Learning Analytics*, vol. 4, no. 2, p. 47–61, 2017.
- [41] H. J. van Veen and N. Saul, "KeplerMapper 1.4.1 documentation," 2019. [Online]. Available: <https://kepler-mapper.scikit-tda.org/en/latest/theory.html>.
- [42] M. Piekenbrock, "Data Science and Security Cluster (DSSC)," 2018. [Online]. Available: [https://peekxc.github.io/resources/DSSC\\_TDA\\_selected\\_slides.pdf](https://peekxc.github.io/resources/DSSC_TDA_selected_slides.pdf).
- [43] C. S. Pun, K. Xia and S. X. Lee, "Persistent-Homology-based Machine Learning and its Applications - A Survey," arXiv, 2018.
- [44] Wikipedia, "Persistent homology," 2021. [Online]. Available: [https://en.wikipedia.org/wiki/Persistent\\_homology](https://en.wikipedia.org/wiki/Persistent_homology).

- [45] L. Tian, "single cell mixology: single cell RNA-seq benchmarking," 2018. [Online]. Available: [https://github.com/LuyiTian/sc\\_mixology](https://github.com/LuyiTian/sc_mixology).
- [46] L. Tian, S. Su, X. Dong, D. Amann-Zalcenstein, C. Biben, A. Seidi, D. J. Hilton, S. H. Naik and M. E. Ritchie, "scPipe: A flexible R/Bioconductor preprocessing pipeline for single-cell RNA-sequencing data," PMC, 2018.
- [47] L. Zappia, B. Phipson and A. Oshlack, "Splatter: simulation of single-cell RNA sequencing data," BMC, 2017.
- [48] Oshlack, "Splatter," 2016. [Online]. Available: <https://github.com/Oshlack/splatter>.
- [49] A. Zeisel, A. B. Muñoz-Manchado, S. Codeluppi, P. Lönnerberg, G. La Manno, A. Juréus, S. Marques, H. Munguba, L. He, J. Hjerling-Leffler and S. Linnarsson, "Cell types in the mouse cortex and hippocampus revealed by single-cell RNA-seq," Science Magazine, 2015.
- [50] Linnarsson Lab, "Single-cell analysis of mouse cortex," 2015. [Online]. Available: <http://linnarssonlab.org/cortex/>.
- [51] PyTorch, "PyTorch," 2020. [Online]. Available: <https://pytorch.org/>.
- [52] J. Nabi, "PyTorch for Deep Learning: A Quick Guide for Starters," 2019. [Online]. Available: <https://towardsdatascience.com/pytorch-for-deep-learning-a-quick-guide-for-starters-5b60d2dbb564>.
- [53] Sayantini, "Keras vs TensorFlow vs PyTorch : Comparison of the Deep Learning Frameworks," 2020. [Online]. Available: <https://www.edureka.co/blog/keras-vs-tensorflow-vs-pytorch/#:~:text=Keras%20has%20a%20simple%20architecture,less%20when%20compared%20to%20Keras..>
- [54] J. Brownlee, "TensorFlow 2 Tutorial: Get Started in Deep Learning With tf.keras," 2019. [Online]. Available: <https://machinelearningmastery.com/tensorflow-tutorial-deep-learning-with-tf-keras/>.
- [55] AI Business, "TensorFlow or PyTorch? A guide to Python machine learning libraries," 2019. [Online]. Available: [https://aibusiness.com/author.asp?section\\_id=789&doc\\_id=761069](https://aibusiness.com/author.asp?section_id=789&doc_id=761069).
- [56] K. Dubovikov, "PyTorch vs TensorFlow — spotting the difference," 2017. [Online]. Available: <https://towardsdatascience.com/pytorch-vs-tensorflow-spotting-the-difference-25c75777377b>.
- [57] UFLDL Tutorial, "Autoencoders," 2020. [Online]. Available: <http://ufldl.stanford.edu/tutorial/unsupervised/Autoencoders/>.
- [58] J. Brownlee, "Gentle Introduction to the Adam Optimization Algorithm for Deep Learning," 2017. [Online]. Available: <https://machinelearningmastery.com/adam-optimization-algorithm-for-deep-learning/>.
- [59] V. Bushaev, "Adam — latest trends in deep learning optimization.,," 2018. [Online]. Available: <https://towardsdatascience.com/adam-latest-trends-in-deep-learning-optimization-6be9a291375c>.
- [60] S. Ruder, "An overview of gradient descent optimization algorithms," 2016. [Online]. Available: <https://ruder.io/optimizing-gradient-descent/index.html#adadelta>.
- [61] Kaggle, "When to use Mean Absolute Error vs Mean Squared Error?," 2019. [Online]. Available: <https://www.kaggle.com/c/home-data-for-ml-course/discussion/143364>.
- [62] M. Binieli, "Machine learning: an introduction to mean squared error and regression lines," 2018. [Online]. Available: <https://www.freecodecamp.org/news/machine-learning-mean-squared-error-regression-line-c7dde9a26b93/>.
- [63] A. S. V, "Understanding Activation Functions in Neural Networks," 2017. [Online]. Available: <https://medium.com/the-theory-of-everything/understanding-activation-functions-in-neural-networks-9491262884e0>.
- [64] J. Brownlee, "A Gentle Introduction to k-fold Cross-Validation," 2018. [Online]. Available: <https://machinelearningmastery.com/k-fold-cross-validation/>.
- [65] C. Liu, "Data Transformation: Standardization vs Normalization," 2017. [Online]. Available: <https://www.kdnuggets.com/2020/04/data-transformation-standardization-normalization.html>.
- [66] P.-N. M. Tan, M. Steinbach and V. Kumar, K-Means Cluster Analysis, Pearson, 2005.
- [67] V. Mallawaarachchi, "Matching of Bipartite Graphs using NetworkX," 2020. [Online]. Available: <https://towardsdatascience.com/matching-of-bipartite-graphs-using-networkx-6d355b164567>.

- [68] StackExchange, "How to test accuracy of an unsupervised clustering model output?," 2017. [Online]. Available: <https://datascience.stackexchange.com/questions/17461/how-to-test-accuracy-of-an-unsupervised-clustering-model-output>.
- [69] Wikipedia, "Rand index," 2021. [Online]. Available: [https://en.wikipedia.org/wiki/Rand\\_index](https://en.wikipedia.org/wiki/Rand_index).
- [70] scikit-learn, "sklearn.metrics.adjusted\_rand\_score," 2020. [Online]. Available: [https://scikit-learn.org/stable/modules/generated/sklearn.metrics.adjusted\\_rand\\_score.html](https://scikit-learn.org/stable/modules/generated/sklearn.metrics.adjusted_rand_score.html).
- [71] M. Minervino, "Topological data analysis with Mapper," 2020. [Online]. Available: <https://www.quantmetry.com/blog/topological-data-analysis-with-mapper/>.
- [72] L. Zappia, B. Phipson and A. Oshlack, "Introduction to Splatter," 2020. [Online]. Available: <https://www.bioconductor.org/packages/release/bioc/vignettes/splatter/inst/doc/splatter.html>.
- [73] S. Chilamkurthy, "WRITING CUSTOM DATASETS, DATALOADERS AND TRANSFORMS," 2017. [Online]. Available: [https://pytorch.org/tutorials/beginner/data\\_loading\\_tutorial.html](https://pytorch.org/tutorials/beginner/data_loading_tutorial.html).
- [74] PyTorch, "TORCH.UTILS.DATA," 2019. [Online]. Available: <https://pytorch.org/docs/stable/data.html#dataset-types>.
- [75] PyTorch Lightning, "TRAINER," 2018. [Online]. Available: <https://pytorch-lightning.readthedocs.io/en/latest/common/trainer.html>.
- [76] Stack Overflow, "Keras: How to save models or weights?," 2019. [Online]. Available: <https://stackoverflow.com/questions/57152978/keras-how-to-save-models-or-weights>.
- [77] Plotly, "Plotly Python Open Source Graphing Library," 2021. [Online]. Available: <https://plotly.com/python/>.
- [78] Biostars, "What is the difference between transcript id and Ensembl gene id," 2013. [Online]. Available: <https://www.biostars.org/p/199073/>.
- [79] L. Tian, X. Dong, S. Freytag and K.-A. Lê Cao, "scRNA-seq mixology: towards better benchmarking of single cell RNA-seq protocols and analysis methods," ResearchGate, 2018.
- [80] J. Brownlee, "How to Get Reproducible Results with Keras," 2019. [Online]. Available: <https://machinelearningmastery.com/reproducible-results-neural-networks-keras/#:~:text=Neural%20network%20algorithms%20are%20stochastic,data%20can%20produce%20different%20results.&text=The%20random%20initialization%20allows%20the,for%20the%20function%20be>.
- [81] S. J. Reddi, S. Kale and S. Kumar, "On the Convergence of Adam and Beyond".
- [82] A. Sharma, "Tackling Underfitting And Overfitting Problems In Data Science," 2018. [Online]. Available: <https://analyticsindiamag.com/tackling-underfitting-and-overfitting-problems-in-data-science/>.
- [83] C. Versloot, "Why you shouldn't use a linear activation function," 2019. [Online]. Available: <https://www.machinecurve.com/index.php/2019/06/11/why-you-shouldnt-use-a-linear-activation-function/>.
- [84] S. Pusuluri, "Autoencoders," 2020. [Online]. Available: <https://medium.com/@sakeshpusuluri123/autoencoders-52c81a6f1ae1>.
- [85] M. Wattenberg, "How to Use t-SNE Effectively," 2016. [Online]. Available: <https://distill.pub/2016/misread-tsne/>.
- [86] ResearchGate, "What is the meaning of negative values in components from PCA analysis?," 2013. [Online]. Available: <https://www.researchgate.net/post/What-is-the-meaning-of-negative-values-in-components-from-PCA-analysis>.
- [87] GeneCards, "SEC61G Gene," 2021. [Online]. Available: <https://www.genecards.org/cgi-bin/carddisp.pl?gene=SEC61G>.
- [88] Wikipedia, "SSU rRNA," 2020. [Online]. Available: [https://en.wikipedia.org/wiki/SSU\\_rRNA](https://en.wikipedia.org/wiki/SSU_rRNA).
- [89] A. Delorme, "Infomax Independent Component Analysis for dummies," 2021. [Online]. Available: [http://arnaudelorme.com/ica\\_for\\_dummies/](http://arnaudelorme.com/ica_for_dummies/).
- [90] C. Feng, S. Liu, H. Zhang, R. Guan, D. Li, F. Zhou, Y. Liang and X. Feng, "Dimension Reduction and Clustering Models for Single-Cell RNA Sequencing Data: A Comparative Study," PCM, 2020.

- [91] sci-kit learn, "Non-Negative Matrix Factorization (NMF)," 2020. [Online]. Available: <https://scikit-learn.org/stable/modules/generated/sklearn.decomposition.NMF.html>.
- [92] F. M. Al-Akwaa, "Analysis of Gene Expression Data Using Biclustering Algorithms," 2012. [Online]. Available: <https://www.intechopen.com/books/functional-genomics/analysis-of-gene-expression-data-using-biclustering-algorithms>.
- [93] Stack Exchange, "When should I use a variational autoencoder as opposed to an autoencoder?," 2018. [Online]. Available: <https://stats.stackexchange.com/questions/324340/when-should-i-use-a-variational-autoencoder-as-opposed-to-an-autoencoder>.
- [94] C. Evans, J. Hardin and D. M. Stoebe, "Selecting between-sample RNA-Seq normalization methods from the perspective of their assumptions," PMC, 2018.
- [95] Encode Box, "Autoencoder in biology – review and perspectives," 2019. [Online]. Available: <https://medium.com/@encodebox/auto-encoder-in-biology-9264da118b83>.
- [96] G. Eraslan, L. M. Simon, M. Mircea, N. S. Mueller and F. J. Theis, "Single-cell RNA-seq denoising using a deep count autoencoder," Nature Communications, 2019.
- [97] statistiXL, "Principal Component Analysis," 2020. [Online]. Available: <https://www.statistixl.com/features/principal-components/>.
- [98] Y. Liu, "Things about High-throughput Sequencing," 2020. [Online]. Available: <https://www.jieandze1314.com/post/enposts/hts/>.
- [99] QRA, "Functional vs Non-Functional Requirements: The Definitive Guide," 2020. [Online]. Available: <https://qracorp.com/functional-vs-non-functional-requirements/>.
- [100] PyTorch, "MSELOSS," 2019. [Online]. Available: <https://pytorch.org/docs/stable/generated/torch.nn.MSELoss.html#torch.nn.MSELoss>.
- [101] K. Tsuyuzaki, H. Sato, K. Sato and I. Nikaido, "Benchmarking principal component analysis for large-scale single-cell RNA-sequencing," BMC, 2020.
- [102] B. J. Erickson, P. Korfiatis, Z. Akkus and T. L. Kline, "Machine Learning for Medical Imaging," PubMed, 2017.
- [103] Javatpoint, "K-Means Clustering Algorithm," 2018. [Online]. Available: <https://www.javatpoint.com/k-means-clustering-algorithm-in-machine-learning>.
- [104] D. M. J. Garbade, "Understanding K-means Clustering in Machine Learning," 2018. [Online]. Available: <https://towardsdatascience.com/understanding-k-means-clustering-in-machine-learning-6a6e67336aa1>.
- [105] T. Wang, T. Johnson, J. Zhang and K. Huang, "Topological Methods for Visualization and Analysis of High Dimensional Single-Cell RNA Sequencing Data," PubMed Central, 2019.
- [106] L. McInnes, "How UMAP Works," 2018. [Online]. Available: [https://umap-learn.readthedocs.io/en/latest/how\\_umap\\_works.html](https://umap-learn.readthedocs.io/en/latest/how_umap_works.html).
- [107] M. Ulmer, L. Ziegelmeier and C. M. Topaz, "A topological approach to selecting models of biological experiments," PLOS One, 2019.
- [108] V. Sze, Y.-H. Chen, T.-J. Yang and J. S. Emer, "Efficient Processing of Deep Neural Networks: A Tutorial and Survey," ResearchGate, 2017.
- [109] A. Harlan, "You Might Be Leaking Data Even if You Cross Validate," 2018. [Online]. Available: <https://alexforrest.github.io/you-might-be-leaking-data-even-if-you-cross-validate.html>.
- [110] J. Murugan and D. Robertson, "An Introduction to Topological Data," arXiv, 2019.

# Appendix

## Appendix A - Ethics Review Sage-HDR Form

### SAGE-HDR

Response ID	Completion date
640816-640807-69910323	28 Dec 2020, 15:08 (GMT)

1	<b>Applicant Name</b>	Tom Wilson
1.a	<b>University of Surrey email address</b>	tw00550@surrey.ac.uk
1.b	<b>Level of research</b>	Undergraduate
1.b.i	<b>Please enter your University of Surrey supervisor's name. If you have more than one supervisor, enter the details of the individual who will check this submission.</b>	Tom Thorne
1.b.ii	<b>Please enter your supervisor's University of Surrey email address. If you have more than one supervisor, enter the details of the supervisor who will check this submission.</b>	tom.thorne@surrey.ac.uk
1.c	<b>School or Department</b>	Computer Science
1.d	<b>Faculty</b>	FEPS - Faculty of Engineering and Physical Sciences Sciences

2	<b>Project title</b>	Clustering of single-cell biological data
---	----------------------	---

3	<b>Please enter a brief summary of your project and its methodology in 250 words. Please include information such as your research method/s, sample, where your research will be conducted and an overview of the aims and objectives of your research.</b>	The aim of the project is to experiment with clustering techniques that can be used to detect hidden variation in single cell data. Neural network architectures will be implemented to extract a lower representation of the data before clustering methods are applied. All real world biological data for testing and training of the implemented neural networks will be from open source datasets freely available online based on previous experiments of RNA sequencing of human and animal cells. Some new data will be artificially generated. No new real world experiments or lab work will be conducted to gather genetic data.
---	---	---

4	<b>Are you making an amendment to a project with a current University of Surrey favourable ethical opinion in place?</b>	NO
---	--	----

5	<b>Does your research involve any animals, animal data or animal derived tissue, including cell lines?</b>	YES
---	--	-----

6	<b>Please read and confirm the following statements.</b>	<ul style="list-style-type: none"> <li>I understand that I have to complete this SAGE-HDR form to assess whether I need an ethics review for human research</li> <li>I understand that upon completion of the SAGE-AR form, I need to complete the SAGE-AR form to determine whether I require an ethics review for animal research</li> </ul>
---	--	--

7	<b>Does your project involve any of the following: human participants (including human data and/or any human tissue*); or is your project linked to engineering and/or the physical sciences?</b>	YES
---	---	-----

8	<b>Does your project involve any type of human tissue research? This includes Human Tissue Authority (HTA) relevant, or irrelevant tissue (e.g. non-cellular such as plasma or serum), any genetic material, samples that have been previously collected, samples being collected directly from the donor or obtained from another researcher, organisation or commercial source.</b>	NO
---	---	----

9	<b>Does your research involve exposure of participants to any hazardous materials e.g. chemicals, pathogens, biological agents or does it involve any activities or locations that may pose a risk of harm to the researcher or participant?</b>	NO
---	--	----

10	<b>Will any activities in your research take place in the Surrey Clinical Research Building (CRB)?</b>	NO
----	--	----

11	<b>Will you be accessing any organisations, facilities or areas that may require prior permission? This includes organisations such as schools (Headteacher authorisation), care homes (manager permission), military facilities etc. If you are unsure, please contact RIGO.</b>	NO
----	---	----

12	<b>Will you be working with any collaborators or third parties to deliver any aspect of the research project?</b>	NO
----	---	----

13	<b>Does your funder, collaborator or other stakeholder require a mandatory ethics review to take place at the University of Surrey?</b>	YES (please contact ethics@surrey.ac.uk)
----	---	--

14	<b>Are you undertaking security-sensitive research, as defined in the text above?</b>	NO
----	---	----

15	Does your project process personal data <sup>1</sup> ? Processing covers any activity performed with personal data, whether digitally or using other formats, and includes contacting, collecting, recording, organising, viewing, structuring, storing, adapting, transferring, altering, retrieving, consulting, marketing, using, disclosing, transmitting, communicating, disseminating, making available, aligning, analysing, combining, restricting, erasing, archiving, destroying.	NO
16	Does your project require the processing of special category <sup>2</sup> data?	YES
16.a	Please ensure that you adhere to the data protection guidance	I am an UG or PGT student and I understand that I have to abide by the 'Data protection and security for undergraduate and postgraduate taught student projects' policy found at <a href="https://research.surrey.ac.uk/ethics">https://research.surrey.ac.uk/ethics</a>
17	Are you using a platform, system or server <sup>3</sup> that is external to the University of Surrey to collect, process and/or store any personal and/or special category data?	NO
18	Does your research involve any of the above statements? If yes, your study may require external ethical review or regulatory approval	NO
19	Does your research involve any of the above? If yes, your study may require external ethical review or regulatory approval	NO
20	Does your project require ethics review from another institution? (For example: collaborative research with the NHS REC, the Ministry of Defence, the Ministry of Justice and/or other universities in the UK or abroad)	NO

24	<b>Declarations</b>	<ul style="list-style-type: none"> <li>I confirm that I have read the University's Code on Good Research Practice and ethics policy and all relevant professional and regulatory guidelines applicable to my research and that I will conduct my research in accordance with these.</li> <li>I confirm that I have provided accurate and complete information regarding my research project</li> <li>I understand that a false declaration or providing misleading information will be considered potential research misconduct resulting in a formal investigation and subsequent disciplinary proceedings liable for reporting to external bodies</li> <li>I understand that if my answers to this form have indicated that I must submit an ethics and governance application, that I will NOT commence my research until a Favourable Ethical Opinion is issued and governance checks are cleared. If I do so, this will be considered research misconduct and result in a formal investigation and subsequent disciplinary proceedings liable for reporting to external bodies.</li> <li>I understand that if I have selected any options on the higher, medium or lower risk criteria then I MUST submit an ethics and governance application (EGA) for review before conducting any research. If I have NOT selected any of the higher, medium or lower risk criteria, I understand I can proceed with my research without review and acknowledge that my SAGE answers and research project will be subject to audit and inspection by the RIGO team at a later date to check compliance</li> </ul>
25	<b>If I am conducting research as a student:</b>	<ul style="list-style-type: none"> <li>I confirm that I have discussed my responses to the questions on this form with my supervisor to ensure they are correct.</li> <li>I confirm that if I am handling any information that can identify people, such as names, email addresses or audio/video recordings and images, I will adhere to the security requirements set out in the relevant Data protection Policy</li> </ul>

## SAGE-AR (Animal Research)

Response ID	Completion date
638929-638920-69566078	16 Dec 2020, 17:15 (GMT)

1	<b>Applicant Name</b>	Tom Wilson
1.a	<b>University of Surrey email address</b>	tw00550@surrey.ac.uk
1.b	<b>Level of research</b>	Undergraduate
1.b.i	<b>Please enter your University of Surrey supervisor's name. (If you have more than one supervisor, enter the details of the supervisor who will check this submission).</b>	Tom Thorne
1.b.ii	<b>Please enter your supervisor's University of Surrey email address. (If you have more than one supervisor, enter the details of the supervisor who will check this submission)</b>	tom.thorne@surrey.ac.uk
1.c	<b>School or Department</b>	Computer Science

2	<b>Project title</b>	Clustering of single-cell biological data
---	----------------------	---

3	<b>Are you joining a supervisors pre-existing research project that has already received Favourable Ethical Opinion?</b>	NO
---	--	----

8	<b>Are you making an amendment to a project with a current University of Surrey favourable ethical opinion in place?</b>	NO
---	--	----

9	<b>Does your research involve ANY animals, animal data or animal derived tissue, including cell lines?</b>	YES
---	--	-----

10	<b>Does your research involve a protected animal; data from a protected animal or tissue derived from a protected animal, including cell lines?</b>	YES
----	---	-----

11	<b>Is the work restricted to the use of commercially available established cell-lines*?</b>	NO
----	---	----

12	<b>Does any of the animal work involve a regulated procedure with respect to the The Animal (Scientific Procedures) Act 1986 if it is to be conducted in the UK?</b>	NO
----	--	----

13	<b>Does the work involve tissues or cells from Genetically Modified Animals?</b>	NO
----	--	----

14	<b>Does your research involve one of, or a combination of, the options above?</b>	NO
----	---	----

15	<b>Does your study include one of, or a combination of, the statements above?</b>	YES
15.a	<b>Please specify the numbers of the statements applicable to your project (e.g. 1, 3, 4)</b>	5

16	<b>Will you be accessing any organisations, facilities or areas that may require prior permission? This includes organisations such as schools (Headteacher authorisation), care homes (manager permission), veterinary services etc. If you are unsure, please contact RIGO.</b>	NO
----	---	----

17	<b>Does your research involve exposure of participants to any hazardous materials e.g. chemicals, pathogens, biological agents or does it involve any activities or locations that may pose a risk of harm to the researcher or participant?</b>	NO
----	--	----

18	<b>Will you be working with any collaborators or third parties to deliver any aspect of the research project?</b>	NO
----	---	----

19	<b>Does your research involve lone working?</b>	NO
----	---	----

20	<b>Is any research activity being conducted outside of the UK?</b>	NO
----	--	----

21	<b>Does your project process personal data<sup>1</sup>? Processing covers any activity performed with personal data, whether digitally or using other formats, and includes contacting, collecting, recording, organising, viewing, structuring, storing, adapting, transferring, altering, retrieving, consulting, marketing, using, disclosing, transmitting, communicating, disseminating, making available, aligning, analysing, combining, restricting, erasing, archiving, destroying.</b>	NO
----	--	----

22	<b>Does your project require the processing of special category<sup>2</sup> data?</b>	NO
----	---	----



23	Are you using a platform, system or server* that is external to the University of Surrey to collect, process and/or store any personal data?	NO
----	--	----

24	Does the research involving animals also involve humans and their personal data or some other form of human participation or human involvement.	NO
----	---	----

30	Declarations	<ul style="list-style-type: none"> <li>I confirm that I will uphold the core elements of the concordat to support research integrity: honesty, rigour, transparency and open communication, and care and respect.</li> <li>I confirm that I have provided accurate and complete information regarding my research project</li> <li>I understand that a false declaration or providing misleading information will be considered potential research misconduct resulting in a formal investigation and subsequent disciplinary proceedings liable for reporting to external bodies</li> <li>I confirm that I have consulted the University guidelines and available ethical review assessments and confirm that the answers provided above are true to the best of my knowledge. I also confirm that, where appropriate to the nature of my research, I will abide by Home Office regulations, University of Surrey Research Integrity and Governance requirements, local research policies and procedures, and all other appropriate University of Surrey policies and guidance, in the management, conduct and administration of this research project.</li> <li>I understand if I make any changes to my research protocol then I should consult the University guidelines and available ethical review assessments again to confirm that the changes do not merit an Ethics committee review.</li> <li>I undertake to abide by legal and ethical requirements and any other guidelines including the University of Surrey Research Data Management policy. I also confirm that there are appropriate procedures for the collection, handling and storage of samples and/or data and the appropriate risk assessments had been undertaken.</li> <li>I understand that the project may be monitored and audited by the University of Surrey to ensure that it is carried out in accordance with University of Surrey code on good research practice, legal and ethical requirements and any other guidelines and policies. I agree to maintain the highest standards of rigour and integrity throughout the course of this project.</li> <li>I confirm that an appropriate process of scientific critique has demonstrated that this research proposal is worthwhile and of high scientific quality and there is adequate funding for the work planned and necessary contracts and/or agreements are in place.</li> </ul>
----	--------------	--

31	If this SAGE-AR form informed you that you had to complete the SAGE-HDR form as your study may require a human ethics review, please confirm you will complete the SAGE-HDR form and will not commence any research until both animal and human reviews are complete.	Not applicable (my research does not contain human participants, their data and/or any human tissue)
----	---	--

32	Undergraduate and Masters' students: If you project includes lone-working or working with hazardous materials, please confirm that your supervisor has seen your risk assessment.	Not applicable - I am a staff member and/or I am not lone-working or working with hazardous materials
----	---	---

33	For studies with overseas research components, including work or travel:	Not applicable - no component of my research is overseas.
----	--	---

34	If I am conducting research as a student:	I confirm that my supervisor has checked the responses on this form are correct.
----	---	--

## Appendix B - Experiment Results

Here additional print screens and tables of results are documented from Testing and Validation.

### Testing the Initial Model

```
Input:
tensor([[ 0.,  0.,  0., ...,  2.,  0.,  2.],
        [ 0., 11.,  3., ...,  0.,  0.,  6.],
        [ 0.,  0.,  0., ...,  1.,  0.,  2.],
        ...,
        [ 0.,  0.,  1., ...,  0.,  1.,  1.],
        [ 2.,  2.,  2., ...,  0.,  0.,  6.],
        [ 0.,  0.,  7., ...,  0.,  0.,  5.]])

Decoded:
tensor([[1., 3., 3., ..., 0., 0., 3.],
        [1., 0., 2., ..., 0., 0., 3.],
        [1., 2., 3., ..., 0., 0., 3.],
        ...,
        [1., 0., 1., ..., 0., 0., 0.],
        [1., 1., 3., ..., 0., 0., 3.],
        [2., 2., 3., ..., 0., 0., 3.]])
grad_fn=<RoundBackward>
Batch average accuracy: 23.26%

Testing average accuracy: 32.03269219091571%
```

Figure 129: Print screen from testing the initial model showing a batch's input tensor, output tensor, accuracy, and average accuracy across all batches

### Activation Function

#### ReLU-ReLU

```
Input:
tensor([[2., 0., 0., ..., 0., 0., 0.],
        [0., 0., 2., ..., 0., 0., 0.],
        [0., 0., 1., ..., 0., 0., 0.],
        [0., 0., 1., ..., 0., 0., 0.]])

Decoded:
tensor([[0., 0., 0., ..., 1., 1., 1.],
        [1., 0., 1., ..., 0., 0., 0.],
        [0., 0., 1., ..., 0., 0., 0.],
        [0., 0., 0., ..., 0., 0., 1.]])
grad_fn=<RoundBackward>
Batch average accuracy: 38.41%

Input:
tensor([[1., 0., 0., ..., 0., 0., 0.],
        [0., 0., 3., ..., 1., 0., 0.]])

Decoded:
tensor([[0., 0., 0., ..., 1., 0., 0.],
        [0., 0., 2., ..., 0., 1., 0.]])
grad_fn=<RoundBackward>
Batch average accuracy: 32.49%

Testing average accuracy: 35.66933638443936%
```

#### ReLU-Linear

```
Input:
tensor([[2., 0., 0., ..., 0., 0., 0.],
        [0., 0., 2., ..., 0., 0., 0.],
        [0., 0., 1., ..., 0., 0., 0.],
        [0., 0., 1., ..., 0., 0., 0.]])

Decoded:
tensor([[0., 1., 2., ..., 0., 0., 0.],
        [0., 2., 1., ..., 0., 0., 0.],
        [0., 1., 1., ..., 0., 0., 0.],
        [0., 0., 1., ..., 0., 0., 0.]])
grad_fn=<RoundBackward>
Batch average accuracy: 42.41%

Input:
tensor([[1., 0., 0., ..., 0., 0., 0.],
        [0., 0., 3., ..., 1., 0., 0.]])

Decoded:
tensor([[1., 2., 2., ..., 0., 0., 0.],
        [0., 1., 3., ..., 0., 0., 0.]])
grad_fn=<RoundBackward>
Batch average accuracy: 35.93%

Testing average accuracy: 38.76553251601191%
```

Figure 130: Two batch's input tensors, outputs, accuracy, and average accuracy across all batches for ReLU-ReLU (left) and ReLU-Linear (right)

#### ReLU + PReLU

```
Input:
tensor([[2., 0., 0., ..., 0., 0., 0.],
        [0., 0., 2., ..., 0., 0., 0.],
        [0., 0., 1., ..., 0., 0., 0.],
        [0., 0., 1., ..., 0., 0., 0.]])

Decoded:
tensor([[0., 2., 0., ..., 2., 0., 0.],
        [0., 2., 2., ..., 0., 0., 1.],
        [0., 1., 0., ..., 0., 0., 0.],
        [0., 0., 2., ..., 0., 1., 0.]])
grad_fn=<RoundBackward>
Batch average accuracy: 35.53%

Input:
tensor([[1., 0., 0., ..., 0., 0., 0.],
        [0., 0., 3., ..., 1., 0., 0.]])

Decoded:
tensor([[0., 1., 2., ..., 0., 0., 0.],
        [0., 1., 0., ..., 1., 0., 0.]])
grad_fn=<RoundBackward>
Batch average accuracy: 28.48%

Testing average accuracy: 32.23660240594197%
```

#### Linear + Linear

```
Input:
tensor([[2., 0., 0., ..., 0., 0., 0.],
        [0., 0., 2., ..., 0., 0., 0.],
        [0., 0., 1., ..., 0., 0., 0.],
        [0., 0., 1., ..., 0., 0., 0.]])

Decoded:
tensor([[0., 0., 1., ..., 0., 0., 1.],
        [1., 2., 1., ..., 0., 0., 1.],
        [1., 1., 0., ..., 0., 0., 0.],
        [0., 0., 1., ..., 0., 0., 0.]])
grad_fn=<RoundBackward>
Batch average accuracy: 41.4%

Input:
tensor([[1., 0., 0., ..., 0., 0., 0.],
        [0., 0., 3., ..., 1., 0., 0.]])

Decoded:
tensor([[1., 1., 1., ..., 0., 1., 2.],
        [0., 0., 2., ..., 1., 0., 1.]])
grad_fn=<RoundBackward>
Batch average accuracy: 34.68%

Testing average accuracy: 38.2560915587487%
```

Figure 131: Two batch's input tensors, outputs, accuracy, and average accuracy across all batches for ReLU-PReLU (left) and Linear-Linear (right)

# Training the Optimised Network

## K-Fold Cross Validation

### Benchmarking Data

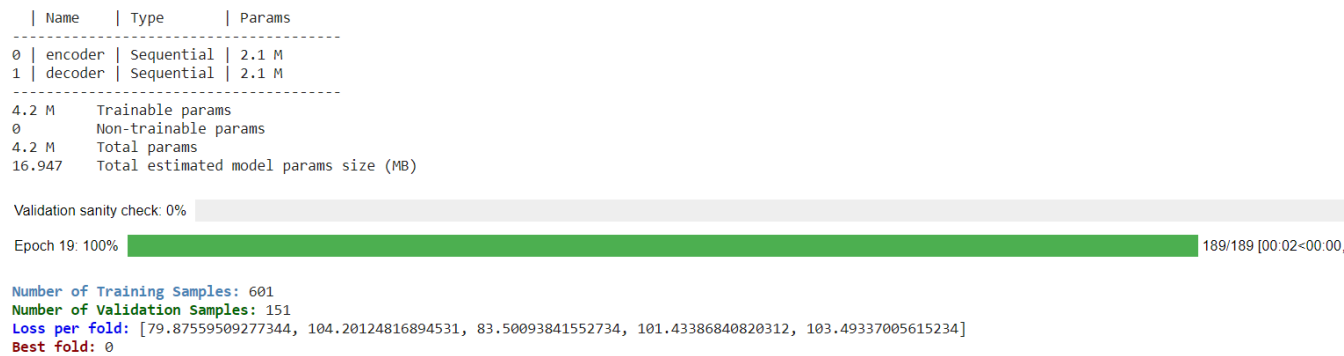


Figure 132: Results of 5-fold cross validation for the benchmarking dataset

### Simulated Data

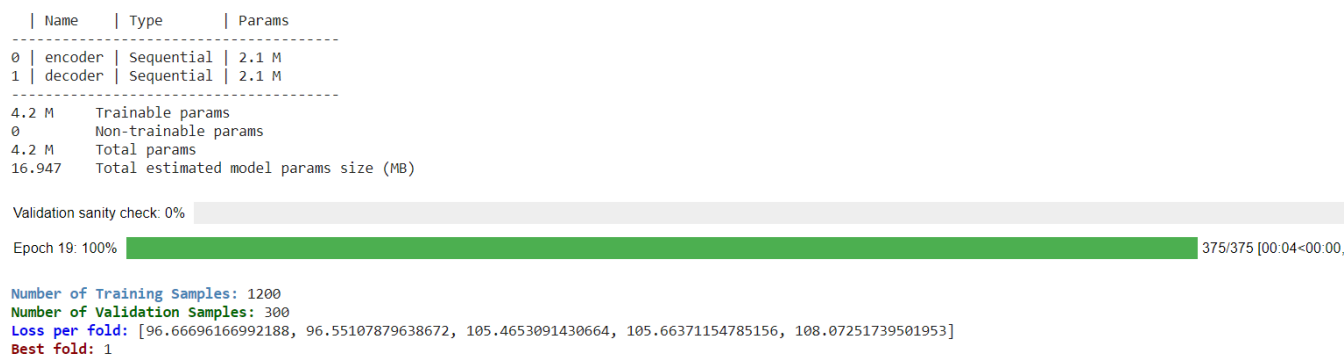


Figure 133: Results of 5-fold cross validation for the simulated dataset

### Mouse Cortex Data

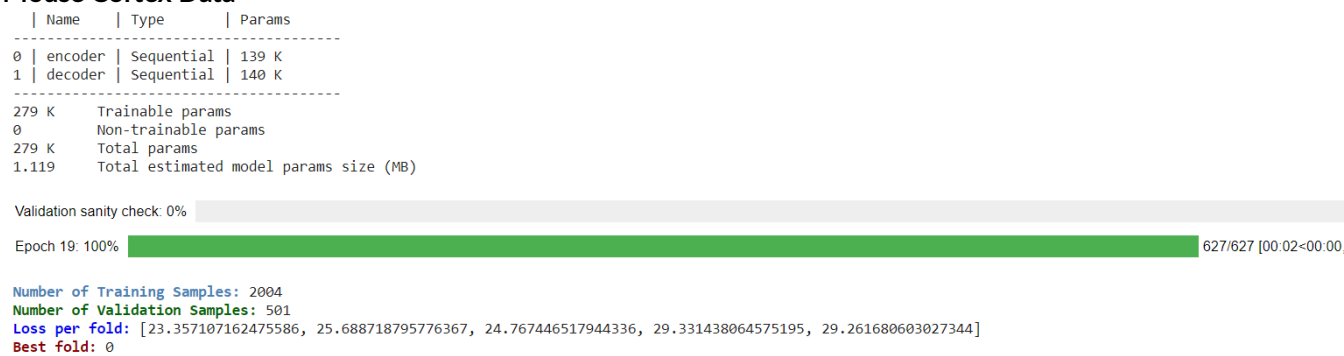


Figure 134: Results of 5-fold cross validation for the mouse cortex dataset

## Testing

### Benchmarking Data

```
Input:
tensor([[0., 0., 1., ..., 0., 0., 0.],
        [0., 0., 4., ..., 0., 0., 3.]])

Encoded:
tensor([[ -70.1457,  -0.5272,   2.2225, -61.6991,  -70.3221,  -1.5425,
          -26.1585, -188.2877,  34.5224,  38.2465,   80.6506,   63.2564,
          -22.8477,  233.7038, -105.3474, -47.9143],
        [ -38.9478,  134.2433, -22.5568, -42.3778, -271.6851,  -85.6588,
          51.5675, -378.4795, -20.4906,  -1.7884,   17.0424,  145.6059,
          -89.4490,  449.1552,  24.4117, -81.9819]])

Decoded:
tensor([[0., 0., 1., ..., 0., 0., 0.],
        [1., 0., 4., ..., 1., 0., 1.]])

Batch average accuracy: 37.25%

Testing average accuracy: 39.945132825383844%
```

Figure 136: Batches and accuracy after training the benchmarking data autoencoder

### Mouse Cortex Data

```
Input:
tensor([[ 1., 10., 2., ..., 1., 0., 0.],
        [ 0., 0., 0., ..., 0., 0., 1.],
        [ 0., 0., 0., ..., 0., 0., 0.],
        [ 0., 7., 1., ..., 0., 1., 0.]])

Encoded:
tensor([[ 125.2051,  423.6984, -106.4994, -180.3438,   65.5065, -269.8898,
          -238.1715,   74.2716, -103.9874,   60.5206,   60.0718,   91.3239,
          -428.1793, -13.3151,   57.9009,  111.5563],
        [ 44.3129,   39.0782, -60.5256,   9.1209, -134.1934,  -45.5595,
          68.2319, -23.9267, -141.2930,  14.3040,   61.2943,  -80.2577,
          -97.2920, 150.2702, -135.3318,   26.2234],
        [ 216.3407,  918.9120,   23.1305, -458.6349,  129.3726, -376.1922,
          -541.6882, -25.0886, -451.6561, -14.6268,  380.8749,   46.6436,
          -777.2988, -231.2023, -142.9618,  124.5012],
        [ 83.5522,  362.4302, -131.1443, -161.0904,   65.1543, -281.6749,
          -250.9871, 129.9018, -105.5186,   56.0630,   45.4922,   65.8867,
          -406.3011, -10.9974,  116.6847,  113.4843]])

Decoded:
tensor([[0., 1., 1., ..., 0., 1., 0.],
        [0., 1., 2., ..., 1., 1., 0.],
        [0., 1., 0., ..., 1., 1., 0.],
        [0., 0., 1., ..., 0., 1., 0.]])

Batch average accuracy: 50.89%

Testing average accuracy: 50.869776119403%
```

Figure 137: Batches and accuracy after training the simulated data autoencoder

### Simulated Data

```
Input:
tensor([[ 2., 0., 2., ..., 10., 0., 1.],
        [ 0., 1., 7., ..., 0., 3., 0.],
        [ 2., 0., 3., ..., 3., 1., 0.],
        [ 2., 0., 1., ..., 15., 0., 1.]])

Encoded:
tensor([[ -2.9273e+01, -1.2050e+02,  3.1610e+01,  3.6534e+01,  2.9003e+01,
          2.2664e+02,  2.7746e+01, -4.9976e+01, -4.4125e+01, -1.8481e+01,
          -1.1692e+01,  1.1314e+01,  4.8926e+01, -7.4451e+01, -5.8594e+02,
          2.8550e+01],
        [ 4.3418e+00,  3.9310e+01, -1.9103e+01, -4.8057e+01,  5.6760e+01,
          7.6769e+00,  3.2309e+01,  1.6482e+01,  7.3766e+00, -4.6713e+01,
          -4.4065e-01, -7.0918e+00, -5.5051e+01, -3.1049e+01, -3.7399e+02,
          1.1783e+01],
        [ -1.7089e+01, -5.6623e+01,  6.6960e+01,  2.4678e+01,  1.2921e+01,
          1.7026e+02,  4.2372e+00, -4.9679e+00,  1.7283e+01, -5.9646e+01,
          9.8561e+00,  1.7942e+01, -5.6028e+00, -2.6169e+01, -3.9861e+02,
          -3.7912e+01],
        [ 5.5243e+01,  3.6151e+00,  1.4449e+01, -4.8631e+01,  6.0673e+01,
          -5.3218e+01, -1.4450e+01,  2.1108e+01, -5.8111e+00,  3.7035e+00,
          5.4735e+01, -4.1914e+01, -2.3165e+01,  2.9204e+01, -5.0824e+02,
          -1.7985e+01]])

Decoded:
tensor([[6., 0., 4., ..., 5., 1., 0.],
        [4., 0., 2., ..., 3., 1., 0.],
        [4., 0., 2., ..., 4., 0., 0.],
        [6., 0., 4., ..., 6., 2., 0.]])

Batch average accuracy: 33.74%

Testing average accuracy: 36.02534612581976%
```

Figure 135: Batches and accuracy after training the simulated data autoencoder

## Clustering

### K-Means

The tables below document the effects of encoding, standardization, and principal components on the performance of k-means clustering. The best number of principal components for each variation has been underlined, and the best overall results coloured red.

### Benchmarking Data

	Encoded						Unencoded			
	Raw			Standardized			Raw		Standardized	
	2 PCs	3 PCs	<u>4 PC</u>	2 PCs	3 PCs	<u>7 PCs</u>	2 PCs	<u>3 PCs</u>	2 PCs	<u>3 PCs</u>
Accuracy	68.7%	98.7%	99.3%	97.3%	98%	99.3%	64%	99.3%	72.7%	100%
ARI	0.396	0.959	0.980	0.920	0.940	0.980	0.387	0.980	0.546	1.0
Incorrectly identified cells	47	3	1	4	2	1	54	1	41	0

Table 20: Results of PCA and k-means on the benchmarking data

## Simulated Data

	Encoded				Unencoded			
	Raw		Standardized		Raw		Standardized	
	2 PCs	<u>6 PCs</u>	2 PCs	<u>9 PCs</u>	2 PCs	<u>19 PCs</u>	2 PCs	<u>5 PCs</u>
Accuracy	49.8%	100%	92.4%	100%	48.8%	100%	53.4%	88.4%
ARI	0.167	1.0	0.861	1.0	0.171	1.0	0.409	0.897
Incorrectly identified cells	251	0	38	0	256	0	228	58

Table 21: Results of PCA and k-means on the simulated data

## Mouse Cortex Data

	Encoded				Unencoded			
	Raw		Standardized		Raw		Standardized	
	2 PCs	<u>16 PCs</u>	2 PCs	<u>15 PCs</u>	2 PCs	<u>18 PCs</u>	2 PCs	<u>17 PCs</u>
Accuracy	28.2%	31.8%	25.4%	36.4%	29.8%	32.6%	36.6%	35.4%
ARI	0.016	0.019	0.007	0.017	0.021	0.037	0.051	0.004
Silhouette Coefficient	0.39	0.115	0.369	0.15	0.393	0.12	0.462	0.265
Incorrectly identified cells	359	341	373	318	351	337	317	323

Table 22: Results of PCA and k-means on the mouse cortex data

	Encoded			Unencoded			
	Raw	Standardized		Raw		Standardized	
	<u>2 PCs</u>	2 PCs	<u>4 PCs</u>	2 PCs	<u>3 PCs</u>	2 PCs	<u>16 PCs</u>
Accuracy	27.6%	26.8%	27%	25.8%	25.4%	27.6%	33.2%
ARI	0.044	0.039	0.032	0.033	0.034	0.076	0.121
Silhouette Coefficient	0.422	0.442	0.449	0.427	0.444	0.482	0.402
Incorrectly identified cells	362	366	365	371	373	362	334

Table 23: Results of k-means combined with PCA and t-SNE with perplexity 30 on the mouse cortex data

	t-SNE Perplexity					
	5	10	20	100	300	600
Accuracy	30%	28.4%	29.8%	34%	37%	21.6%
ARI	0.104	0.1	0.112	0.125	0.137	0.005
Silhouette Coefficient	0.451	0.435	0.436	0.374	0.399	0.528

Incorrectly identified cells	350	358	351	330	315	392
------------------------------	-----	-----	-----	-----	-----	-----

Table 24: Results of k-means and t-SNE for different perplexity values on the mouse cortex data

## Agglomerative Hierarchical Clustering

The tables below document the effects of encoding, standardization, and principal components on the performance of hierarchical clustering.

### Benchmarking Data

	Encoded				Unencoded			
	Raw		Standardized		Raw		Standardized	
	2 PCs	3 PCs	2 PCs	3 PCs	2 PCs	3 PCs	2 PCs	3 PCs
Accuracy	70%	95.3%	99.3%	92%	66%	99.3%	72.7%	100%
ARI	0.439	0.863	0.980	0.771	0.447	0.980	0.576	1.0
Incorrectly identified cells	45	7	1	12	51	1	41	0

Table 25: Results of PCA and agglomerative hierarchical clustering on the benchmarking data

### Simulated Data

	Encoded				Unencoded			
	Raw		Standardized		Raw		Standardized	
	2 PCs	6 PCs	2 PCs	9 PCs	2 PCs	7 PCs	2 PCs	4 PCs
Accuracy	55%	99.8%	90.2%	99.8%	47%	97.6%	48.2%	77.4%
ARI	0.256	0.996	0.875	0.996	0.202	0.934	0.184	0.539
Incorrectly identified cells	2251	1	49	1	265	12	259	113

Table 26: Results of PCA and agglomerative hierarchical clustering on the simulated data

### Mouse Cortex Data

	Encoded				Unencoded			
	Raw		Standardized		Raw		Standardized	
	2 PCs	3 PCs	2 PCs	3 PCs	2 PCs	15 PCs	2 PCs	9 PCs
Accuracy	30.2%	39.6%	27%	40.2%	31.4%	44.4%	33.4%	44.4%
ARI	0.028	0.036	0.028	0.07	0.019	0.079	0.02	0.095
Silhouette Coefficient	0.276	0.087	0.326	0.115	0.378	0.095	0.45	0.262
Incorrectly identified cells	349	302	365	299	343	278	333	278

Table 27: Results of PCA and agglomerative hierarchical clustering on the mouse cortex data

## Alternative Algorithms

The tables below show the experiment results that achieved the highest accuracy for six different clustering algorithms. The tables document the best combinations between encoding, standardization, and the best number of components for dimensionality reduction.

### Benchmarking Data

#### Principal Component Analysis (PCA)

	Accuracy	Incorrect Cells	Principal Components	Encoded	Standardized
K-Means	99.8%	2	3	False	True
Hierarchical	99.9%	1	3	False	True
BIRCH	99.9%	1	3	False	True
Mini Batch K-Means	99.8%	2	3	False	True
Spectral	99.9%	1	4	False	True
Gaussian Mixture	99.8%	2	3	False	True

Table 28: Benchmarking data alternative clustering algorithms results for PCA

#### Independent Component Analysis (ICA)

	Accuracy	Incorrect Cells	Independent Components	Encoded	Standardized
K-Means	99.8%	2	3	False	True
Hierarchical	99.9%	1	3	False	True
BIRCH	99.9%	1	3	False	True
Mini Batch K-Means	99.8%	2	3	False	True
Spectral	99.9%	1	4	False	True
Gaussian Mixture	99.8%	2	3	False	True

Table 29: Benchmarking data alternative clustering algorithms results for ICA

#### Non-negative matrix factorization (NMF)

	Accuracy	Incorrect Cells	Basis Components	Encoded	Standardized
K-Means	99.9%	1	2	False	True
Hierarchical	99.1%	8	8	False	True
BIRCH	99.1%	8	8	False	True
Mini Batch K-Means	99.6%	3	14	False	True
Spectral	35.1%	585	14	True	False



Gaussian Mixture	99.8%	2	4	False	True
------------------	-------	---	---	-------	------

Table 30: Benchmarking data alternative clustering algorithms results for NMF

#### PCA with t-SNE

	Accuracy	Incorrect Cells	Principal Components	Encoded	Standardized
K-Means	99.67%	3	2	True	True
Hierarchical	99.67%	3	2	True	True
BIRCH	99.67%	3	2	True	True
Mini Batch K-Means	99.67%	3	2	True	True
Spectral	99.67%	3	2	True	True
Gaussian Mixture	99.67%	3	2	True	True

Table 31: Benchmarking data alternative clustering algorithms results for PCA and t-SNE

#### Simulated Data

##### Principal Component Analysis (PCA)

	Accuracy	Incorrect Cells	Principal Components	Encoded	Standardized
K-Means	100%	0	9	True	False
Hierarchical	100%	0	4	False	True
BIRCH	100%	0	4	False	True
Mini Batch K-Means	99.9%	2	4	False	True
Spectral	98.1%	39	6	False	True
Gaussian Mixture	100%	0	9	False	False

Table 32: Simulated data alternative clustering algorithms results for PCA

##### Independent Component Analysis (ICA)

	Accuracy	Incorrect Cells	Independent Components	Encoded	Standardized
K-Means	100%	0	9	True	False
Hierarchical	100%	0	4	False	True
BIRCH	100%	0	4	False	True
Mini Batch K-Means	99.8%	3	5	False	True
Spectral	98.1%	39	6	False	True

Gaussian Mixture	100%	0	9	False	False
------------------	------	---	---	-------	-------

Table 33: Simulated data alternative clustering algorithms results for ICA

#### Non-negative matrix factorization (NMF)

	Accuracy	Incorrect Cells	Basis Components	Encoded	Standardized
K-Means	81.8%	363	17	False	True
Hierarchical	92.2%	156	11	False	False
BIRCH	92.2%	156	11	False	False
Mini Batch K-Means	81.7%	366	8	False	True
Spectral	40.2%	1196	11	True	False
Gaussian Mixture	81.1%	379	13	True	True

Table 34: Simulated data alternative clustering algorithms results for NMF

#### PCA with t-SNE

	Accuracy	Incorrect Cells	Principal Components	Encoded	Standardized
K-Means	100%	0	2	True	False
Hierarchical	100%	0	2	True	False
BIRCH	100%	0	2	True	False
Mini Batch K-Means	100%	0	2	True	False
Spectral	100%	0	2	True	False
Gaussian Mixture	100%	0	2	True	False

Table 35: Simulated data alternative clustering algorithms results for PCA and t-SNE