Master Thesis

Unsupervised Pattern Discovery for Mobile Network Tuning Optimization

Author:
Nadezhda Ilieva

Supervisors:
Prof. Haitham Al Hassanieh
Dr. David Froelicher
Dr Daniel Dobos

Laboratory of Sensing and Networking Systems (SENS)
School of Computer and Communication Sciences, EPFL
Swisscom Digital Lab, Swisscom AG

Lausanne, 22 August 2025

L'important dans la vie ce n'est point le triomphe, mais le combat,
l'essentiel ce n'est pas d'avoir vaincu mais de s'être bien battu.
— Pierre de Coubertin

Acknowledgements

I would like to thank my supervisors Prof. Haitham Al Hassanieh, Dr. David Froelicher and Dr. Daniel Dobos for their support and guidance throughout the duration of this thesis. I am also very grateful for the collaboration and invaluable discussions with Ivan Vallejo Vall from the Wireless Analytics team, as well as Josko Kresic, Guarisco Noris and Francesco Pellegrini, radio network engineers at Swisscom. Finally, many thanks to everyone at the Swisscom Digital Lab, including the research engineers and fellow interns, for their kindness and support, which made my time during the thesis much more enjoyable.

I would also like to acknowledge and thank everyone else who has been part of my journey at EPFL. Huge thank you to my parents, my sister Jasna and Justin for their love and support, to my second family in Switzerland, and to my friends. I am also immensely grateful to Prof. Serge Vaudenay for making it possible to study at EPFL by graciously offering me a Research Scholar position and for his kindness.

Lausanne, August 2025

Abstract

To ensure reliable service, mobile network operators continuously monitor and manage their networks. Radio network engineers analyze the performance of mobile cells and adjust configuration parameters to optimize operation, through a process commonly known as *tuning*. However, tuning is time-consuming and repetitive, which makes it inefficient at scale. This thesis addresses the problem of detecting recurring performance patterns across mobile cells to support semi-automated network tuning. Instead of relying on predefined rules or known issues, the proposed approach uses unsupervised machine learning techniques to group mobile cells based on their performance similarities. This allows engineers to analyze entire clusters, identify shared issues, and apply common solutions, ultimately reducing workload and improving the response times. The thesis explores various data representations, including image-based and graph-based representations, and compares several embedding learning techniques, such as convolutional neural networks, autoencoders, and graph-based models. It evaluates multiple clustering algorithms, including k-means, DBSCAN, and Spectral clustering. Given the unsupervised nature of the problem, it proposes an evaluation pipeline using multiple visualizations (e.g., PCA, t-SNE, UMAP, custom heatmaps) and clustering metrics (e.g., clustering coefficients, Rand index, mutual information).

Keywords: *unsupervised representation learning, clustering, pattern discovery*

List of Tables

6.1 Clustering parameters by embedding-learning method 36
6.2 Adjusted Rand Index (ARI) between clustering methods (Pretrained CNN) . . . 40
6.3 Adjusted Mutual Information (AMI) between clustering methods (Pretrained CNN) 40
6.4 Adjusted Rand Index (ARI) between clustering methods (Single-layer KPI CAE) 45
6.5 Adjusted Mutual Information (AMI) between clustering methods (Single-layer KPI CAE) 45
6.6 Adjusted Rand Index (ARI) between clustering methods (VGAE) 55
6.7 Adjusted Mutual Information (AMI) between clustering methods (VGAE) 55
6.8 Clustering coefficients for k-means and Spectral clustering (DGLC) 56
6.9 ARI and AMI indices between k-means and Spectral clustering cluster assignments for different numbers of clusters (DGLC) 56

A.1 Training and validation results for different Single-layer KPI CAE configurations 72
A.2 Training and validation results for different Multi-layer KPI CAE configurations . 72
A.3 Training and validation results for different GAE configurations 73
A.4 Training and validation results for different VGAE configurations 75
A.5 Training and validation results for different DGLC configurations 75

List of Figures

3.1 Architecture of a convolutional neural network. 6
3.2 Architecture of a convolutional autoencoder (CAE) (Guo et al. 2017). 7

4.1 Reference table for throughput, SINR, RSRP and RSRQ 21
4.2 Cell-related (left) and tile-related (right) attributes 23
4.3 Pearson and Spearman correlation matrices for the average SINR, RSRP and RSRQ 24

5.1 Methodology overview 25
5.2 Evaluation overview 26
5.3 An example of an interactive footprint map generated by an internal Swisscom tool 27
5.4 Example of an image-based representation of a single mobile cell. The cell site is centered and the image is rotated so the beam direction points north. Tiles are shown as circles, and KPI values are shown using color. The three images represent different KPIs: throughput doominance, SINR, and RSRP. 28
5.5 Example of an graph-based representation of a single mobile cell. Tiles are represented as nodes, and KPI measurements as node features. Edges are drawn between neighboring tiles. 29
5.6 An illustration of the pipeline with a pretrained CNN 30
5.7 An illustration of the pipeline with a convolutional autoencoder (CAE) 30
5.8 Example of a radial sector map 35

6.1 Silhouette, Calinski-Harabasz and Davies-Bouldin scores for different values of k of the k-means algorithm (Pretrained CNN) 37
6.2 Elbow method for k-means (left) and k-dist graph for DBSCAN (right) (Pretrained CNN) 38
6.3 Number of clusters and fraction of outliers as functions of ε and `min_samples` (Pretrained CNN) 39
6.4 Silhouette, Calinski-Harabasz and Davies-Bouldin scores for different values of k of the Spectral clustering algorithm (Pretrained CNN) 39
6.5 Eigengap heuristic for Spectral clustering (Pretrained CNN) 40
6.6 Clustering results on pretrained CNN embeddings visualized with PCA (left), t-SNE (center), and UMAP (right), using k-means (top row), DBSCAN (middle row), and Spectral clustering (bottom row). 41
6.7 Spatial performance patterns discovered using k-means, shown as radial sector maps of bad throughput probability per sector (Pretrained CNN) 42

6.8 Spatial performance patterns discovered using DBSCAN, shown as radial sector maps of bad throughput probability per sector (Pretrained CNN) 42
6.9 Spatial performance patterns discovered using Spectral clustering, shown as radial sector maps of bad throughput probability per sector (Pretrained CNN) 42
6.10 Example mobile cells from Cluster 9 discovered by DBSCAN (Pretrained CNN) . 43
6.11 Example mobile cells from Cluster 14 discovered by DBSCAN (Pretrained CNN) 43
6.12 Example mobile cells from Cluster 17 discovered by DBSCAN (Pretrained CNN) 43
6.13 Clustering results on Single-layer KPI CAE embeddings visualized with PCA (left), t-SNE (center), and UMAP (right), using k-means (top row), DBSCAN (middle row), and Spectral clustering (bottom row). 45
6.14 Spatial performance patterns discovered using k-means, shown as radial sector maps of bad throughput probability per sector (Single-layer KPI CAE) 46
6.15 Spatial performance patterns discovered using DBSCAN, shown as radial sector maps of bad throughput probability per sector (Single-layer KPI CAE) 46
6.16 Spatial performance patterns discovered using Spectral clustering, shown as radial sector maps of bad throughput probability per sector (Single-layer KPI CAE) . . 46
6.17 Example mobile cells from Cluster 4 discovered by k-means (Single-layer KPI CAE) 47
6.18 Example mobile cells from Cluster 8 discovered by k-means (Single-layer KPI CAE) 47
6.19 Example mobile cells from Cluster 13 discovered by k-means (Single-layer KPI CAE) 48
6.20 Multi-layer KPI CAE embeddings visualized with PCA (left), t-SNE (center), and UMAP (right) 49
6.21 Spatial performance patterns discovered using k-means, shown as radial sector maps of tile probability per sector (Multi-layer KPI CAE) in the top row. The bottom row shows example mobile cells from each cluster, using the image corresponding to throughput dominance. 50
6.22 Clustering results on FEATHER embeddings visualized with PCA (left), t-SNE (center), and UMAP (right), using k-means (top row), DBSCAN (middle row), and Spectral clustering (bottom row). 51
6.23 Examples of clusters corresponding to isomorphic graph structures discovered by DBSCAN (FEATHER). Each row corresponds to a different cluster and shows the graph structure along with a few examples. 53
6.24 Clustering results on VGAE embeddings visualized with PCA (left), t-SNE (center), and UMAP (right), using k-means (top row), DBSCAN (middle row), and Spectral clustering (bottom row). 54
6.25 Average number of tiles per mobile cell per cluster for 10 (top row), 20 (middle row), and 30 (bottom row) clusters produced by k-means (left) and Spectral clustering (right) (DGLC). 57
6.26 Radial sector maps of bad throughput probability for Cluster 6 (k-means; top row) and Cluster 4 (Spectral clustering; bottom row), along with two example mobile cells (DGLC). 58

A.1 Spatial performance patterns discovered using DBSCAN on embeddings from the Pretrained CNN, shown as radial sector maps of bad throughput probability per sector for each cluster (1–6). 67
A.2 Spatial performance patterns discovered using DBSCAN on embeddings from the Pretrained CNN, shown as radial sector maps of bad throughput probability per sector for each cluster (7–18). 68
A.3 Spatial performance patterns discovered using Spectral clustering on embeddings from the Pretrained CNN, shown as radial sector maps of bad throughput probability per sector for each cluster (1–12). 69
A.4 Spatial performance patterns discovered using Spectral clustering on embeddings from the Pretrained CNN, shown as radial sector maps of bad throughput probability per sector for each cluster (13–24). 70
A.5 Spatial performance patterns discovered using Spectral clustering on embeddings from the Pretrained CNN, shown as radial sector maps of bad throughput probability per sector for each cluster (25–30). 71
A.6 Original images from the test set and their reconstructions produced by the single-KPI CAE model with a hidden dimension of 64 and batch size of 16. . . . 73
A.7 Original images from the test set and their reconstructions produced by the multi-KPI CAE model with a hidden dimension of 64 and batch size of 16. 74
A.8 Illustration of Steps 1 and 2 of the preliminary experimental setup. 76

Table of Contents

1 Introduction **1**

2 Related work **3**

3 Theoretical Background **5**
3.1 Convolutional Neural Network . . . 5
3.2 Convolutional Autoencoder . . . 6
3.3 Graphs . . . 7
3.4 Graph Neural Network . . . 8
3.5 Graph Convolutional Network . . . 9
3.6 Graph Autoencoder . . . 9
3.7 Variational Graph Autoencoder . . . 10
3.8 Deep Graph-Level Clustering (DGLC) . . . 10
3.9 FEATHER . . . 13
3.10 Clustering algorithms . . . 14
3.10.1 k-means algorithm (Lloyd's algorithm) . . . 14
3.10.2 DBSCAN (Density-Based Spatial Clustering of Applications with Noise) . . . 14
3.10.3 Spectral clustering . . . 15
3.11 Clustering performance evaluation . . . 16
3.11.1 Silhouette Coefficient . . . 16
3.11.2 Calinski-Harabasz Index . . . 17
3.11.3 Davies-Bouldin Index . . . 18
3.11.4 Adjusted Rand Index . . . 18
3.11.5 Adjusted Mutual Information Score . . . 19

4 Data **20**
4.1 Performance metrics . . . 20
4.2 Data preparation . . . 22
4.3 Exploratory data analysis . . . 23

5 Methods **25**
5.1 Data representation . . . 26
5.1.1 Image-based representation . . . 27
5.1.2 Graph-based representation . . . 28
5.2 Embedding techniques . . . 29
5.2.1 Methods for image-based representations . . . 29

5.2.2 Methods for graph-based representations . . . 31
5.3 Clustering algorithms . . . 32
5.4 Evaluation . . . 34

6 Results **36**
6.1 Image-based representations . . . 37
6.1.1 Pretrained CNN . . . 37
6.1.2 Convolutional Autoencoder . . . 44
6.2 Graph-based representations . . . 49
6.2.1 FEATHER . . . 49
6.2.2 (Variational) Graph Autoencoder . . . 52
6.2.3 DGLC . . . 55

7 Discussion **59**
7.1 Summary of results . . . 59
7.2 Future directions . . . 60

8 Conclusion **62**

Bibliography **63**

A Appendix **67**
A.1 Pretrained CNN . . . 67
A.1.1 Radial sector maps . . . 67
A.2 Convolutional Autoencoder . . . 72
A.2.1 Hyperparameter tuning . . . 72
A.3 Graph Autoencoder . . . 72
A.3.1 Hyperparameter tuning . . . 72
A.4 Variational Graph Autoencoder . . . 73
A.4.1 Hyperparameter tuning . . . 73
A.5 Deep Graph Level Clustering (DGLC) . . . 75
A.5.1 Hyperparameter tuning . . . 75
A.6 Satellite images . . . 75

1 Introduction

Mobile networks are complex infrastructures that provide wireless connectivity to mobile users at different locations. They are composed of thousands of *mobile cells* spread across different geographical areas, each served by a *base station*. Formally, each mobile cell is determined by its location (i.e., the physical site), sector (i.e., the beam direction), and the band of radio frequencies it uses to communicate with the user equipment. Typically, neighboring cells are configured to use different bands of radio frequencies, to avoid interference. As users move, their devices are handed over between neighboring cells to maintain continuous service without interruption.

To ensure good performance and reliable coverage, mobile network operators (MNOs) need to continuously monitor and manage their networks. MNOs collect measurements from the user devices and the network itself on quality indicators such as signal strength, signal-to-interference-plus-noise ratio (SINR), and throughput. The collected data can be used to adjust the configuration parameters of the network in order to improve the performance. For example, at the level of a mobile cell, network operators can adjust configuration parameters from handover thresholds (i.e., when the cell transfers the connection to a neighboring cell as the user moves) to transmission power and antenna tilt.

The process of adjusting network configuration parameters to optimize performance is known as *network tuning*. Network tuning relies on the expertise of radio network engineers, who analyze performance data to detect and resolve radio issues. Engineers inspect a mobile cell, or a group of neighboring mobile cells, by manually reviewing performance indicators across the coverage area. To support this process, Swisscom has developed a data processing and visualization tool that converts raw data into interactive visualizations of performance metrics. However, tuning still remains time-consuming and repetitive, making it inefficient at scale.

One challenge to address is the issue of *repetitiveness*. Swisscom manages an infrastructure of tens of thousands of mobile cells, so engineers often encounter the same or similar scenarios repeatedly. One example is the *back-lobe phenomenon*, where the antenna partially radiates in the opposite direction of the main lobe (the beam direction). This is undesirable as it wastes energy and can cause interference. Another instance is the appearance of coverage zones with poor performance at various distances and directions from the mobile cell site, such as in front and back, or at a specific angle. In fact, many spatial performance patterns can exist, and each

can be potentially linked to a specific radio issue. Hence, it would be useful to automatically detect and group mobile cells based on their performance patterns. By clustering together mobile cells with similar characteristics, network engineers can analyze entire clusters simultaneously, determine whether there is a shared problem, and potentially apply a common solution. This would lead to workload reduction and increased efficiency, as the manual case-by-case task would be turned into a semi-automated batch process. In turn, this process would also improve the customer experience for the mobile users, as it would lead to faster problem resolution or even detection of issues that might have otherwise been missed.

A naive approach to solving the described issue would be to precisely define the classes of patterns to be detected and implement automated checks to identify them. However, this approach assumes that all possible patterns are known in advance, which is not the case. Hence, the problem is a natural candidate for a solution using machine learning. Specifically, unsupervised machine learning techniques can be used to partition the set of mobile cells into different clusters based on their similarity. These techniques can also help identify patterns that were previously unknown or might have gone unnoticed by network engineers.

This thesis explores various representation methods for mobile cell data, including image-based and graph-based representations. Additionally, it compares different approaches for learning compact embeddings using machine learning, from pretrained convolutional neural networks and convolutional autoencoders to traditional graph representations learning methods and graph autoencoders. Moreover, it uses several clustering algorithms to discover meaningful clusters, such as k-means, DBSCAN and Spectral clustering. Given the unsupervised nature of the problem, this thesis also proposes an evaluation pipeline consisting of multiple visualizations (e.g., PCA, t-SNE, UMAP projections, and custom heatmaps) and metrics (e.g., clustering coefficients, Rand index, and mutual information) to provide insights into the quality of the results. Finally, it incorporates expert feedback from network engineers at Swisscom on the identified issues.

The rest of this thesis is organized as follows. Chapter 2 reviews existing research literature on methods for grouping mobile cells based on their performance characteristics, to uncover common patterns or to support configuration. Chapter 3 discusses the relevant theoretical background from machine learning architectures to clustering algorithms and clustering performance evaluation metrics. Chapter 4 describes the dataset and the data preparation process. Chapter 5 introduces the methods used to obtain clusters of mobile cells with similar performance characteristics, including the choice of data representation, embedding techniques and clustering algorithms. Finally, Chapter 6 and Chapter 7 showcase the results and discussion respectively.

2 Related work

This section reviews research literature on methods for clustering mobile cells based on their performance characteristics, to uncover common patterns and to support cell configuration and fault detection.

(Raivio et al. 2003) propose two clustering approaches based on *self-organizing maps* to group 3G mobile cells using base station parameters and call quality information. (Gómez-Andrades et al. 2016) present an automatic diagnosis system based on *self-organizing maps* and *Ward's hierarchical clustering* to group 4G/LTE cells exhibiting same fault causes, using a small set of their key performance indicator (KPI) metrics. Similarly, (Liu et al. 2019) introduce an anomaly detection framework based on key quality indicator (KQI) data reflecting cell performance in terms of quality of experience (QoE), using *self-organizing maps* and *k-medoids* clustering. Furthermore, (Wang and Ferrús 2021) propose a 2-stage clustering algorithm, combining *self-organizing maps* and *k-means*, to cluster similar 4G/LTE mobile cells based on their operation, administration and maintenance (OAM) properties. Finally, (Zhang et al. 2019) introduce a cellular radio access system enhanced with deep learning to monitor cell KPIs, predict anomalies, analyze root causes, and self-heal. For the root cause analysis, *agglomerative hierarchical clustering* is applied to features extracted from a synthetic dataset of cell KPIs using an autoencoder, in order to group cells with similar issues prior to the self-healing phase.

Another area of research focuses on clustering mobile cells based on the temporal variations of their KPI metrics, which are time series reflecting the environmental and performance characteristics. (Li, Francini, and Magli 2023) develop a novel clustering approach designed to capture temporal dynamics, using first-order difference sequences and distribution summarization as feature extractors, together with *k-means*. (Lu et al. 2022) present a hybrid time series clustering method that uses statistical and temporal data with *k-medoids*. Similarly, (Mazguła, Król, and Jabłoński 2024) introduce a new algorithm that combines *dynamic time warping* and *spectral clustering* to cluster 5G mobile network performance data.

In a different manner, (Shibli and Zanouda 2024) incorporate satellite imagery of the cell coverage area to develop a forecasting model for predicting network KPIs. Specifically, cells are clustered using *k-means* based on the geographical similarity of their coverage areas, and a separate KPI prediction model is trained for each cluster.

To the best of our knowledge, all of the aforementioned research works rely on aggregated performance metrics. That is, the datasets used mainly contain cell-level aggregations, such as average throughput or average SINR, computed over all observations collected from user equipment connected to the mobile cell of interest, regardless of their specific location. The novelty of the work presented in this thesis lies in the granularity of the data. Specifically, as described in detail in Chapter 4, the data is aggregated over fixed 200 meter by 200 meter regions, enabling a finer-grained analysis. As a result, it is possible to pinpoint exactly which part of a mobile cell's coverage area is problematic, and to have cells grouped based on these spatial performance patterns. Finally, this thesis explores deep learning methods to learn embeddings from the mobile cell data, which is a different approach to using traditional machine learning techniques such as self-organizing maps.

3 Theoretical Background

This section outlines the relevant theoretical background needed to understand the methods and results presented in this thesis. These concepts include deep learning architectures suited for images (e.g. convolutional neural networks and convolutional autoencoders) and graphs (e.g. graph neural networks, graph convolutional networks and graph autoencoders). Furthermore, a few clustering algorithms are presented (i.e., k-means, DBSCAN and Spectral clustering), together with clustering performance evaluation metrics (e.g., Silhouette coefficient, Calinski-Harabasz index and more).

3.1 Convolutional Neural Network

Convolutional neural networks (CNNs) (Fukushima 1980; LeCun et al. 1989) are feed-forward neural networks used for object detection, image classification and more. The CNN architecture is composed of three main types of layers: *convolutional*, *pooling* and *fully-connected*.

Convolutional layer The convolutional layer is the main component of a convolutional neural network. The parameters of the convolutional layer are a set of learnable *convolution kernels*, also known as *convolution filters*. Given an input of size $H \times W \times C$ (height, width, number of channels), a convolution filter of size $F \times F \times C$ is applied across the input in a sliding manner. Specifically, the filter shifts by a stride S, performing the convolution operation until it covers the entire input. This produces an output called a *feature* (or *activation*) *map*, which is passed through a non-linear activation function (e.g., ReLU, LeakyReLU, ELU, or Sigmoid (Dubey, Singh, and Chaudhuri 2022)). The filter size F usually takes odd values smaller than 10, while S is often set to 1. Using a larger value of S has a similar effect on the shape of the activation map as the pooling operation described below.

Pooling layer The pooling layer performs dimensionality reduction and is usually applied after a convolution layer. It slides a filter over the input, but unlike with convolutional layers, the filter has no learnable parameters and performs a fixed aggregation. Common types include *max pooling*, which selects the highest value in the filter area, and *average pooling*, which calculates

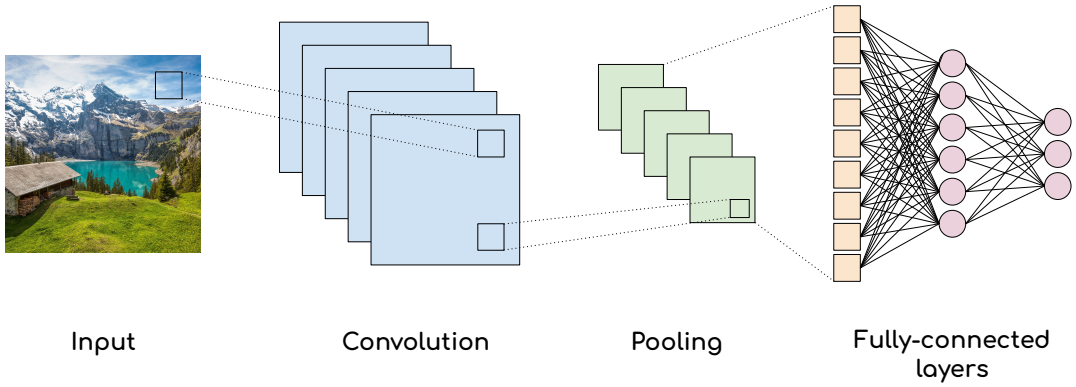

Figure 3.1: Architecture of a convolutional neural network.

the average of the covered pixels.

Fully-connected layer Fully-connected layers are located at the end of the CNN architecture and perform the classification task using the features extracted by earlier layers. The output from the convolutional and pooling layers is flattened and passed to the fully-connected layers. As the name suggests, each input element is connected to every neuron in the layer, and each neuron in one layer connects to all neurons in the following layer.

An overview of the entire CNN architecture is shown in Figure 3.1. In general, multiple convolutional and pooling layers can be stacked sequentially. With each added layer, the complexity of the model increases. Popular convolutional neural network architectures include the LeNet (Lecun et al. 1998), AlexNet (Krizhevsky, Sutskever, and G. E. Hinton 2017), VGGNet (Simonyan and Zisserman 2015), ResNet (He et al. 2015) and more.

3.2 Convolutional Autoencoder

Convolutional autoencoders (CAEs) are neural network autoencoders that use convolutional layers for both encoding and decoding, which makes them suitable for handling image data. The autoencoder architecture consists of two main parts: an *encoder* and a *decoder*. Given a dataset $\mathcal{X} = \{\mathbf{x}_1, \mathbf{x}_2 \dots \mathbf{x}_n\}$ of n images, the encoder f_θ compresses each $\mathbf{x}_i \in \mathbb{R}^{H \times W \times C}$ into a *lower-dimensional representation* (i.e., *embedding*) $\mathbf{h}_i \in \mathbb{R}^d$ as:

$$\mathbf{h}_i = f_\theta(\mathbf{x}_i). \tag{3.1}$$

The decoder g_ϕ attempts to reconstruct the original input from the compressed representation $\mathbf{h}_i$ as:

$$\mathbf{y}_i = g_\phi(\mathbf{h}_i) = g_\phi(f_\theta(\mathbf{x}_i)), \tag{3.2}$$

where $\mathbf{y}_i \in \mathbb{R}^{H \times W \times C}$ is the *reconstruction*.

The parameters of the autoencoder are optimized by minimizing the reconstruction loss, usually defined as the sum of the mean squared errors (MSE) between the input $\mathbf{x}_i$ and the reconstructed output $\mathbf{y}_i$:

$$\mathcal{L} = \frac{1}{n}\sum_{i=1}^{n} \|\mathbf{x}_i - \mathbf{y}_i\|_2^2 = \frac{1}{n}\sum_{i=1}^{n} \|\mathbf{x}_i - g_\phi(f_\theta(\mathbf{x}_i))\|_2^2. \tag{3.3}$$

In CAEs, both the encoder and decoder are composed of convolutional layers, with the decoder using transposed convolutions, as illustrated in Figure 3.2.

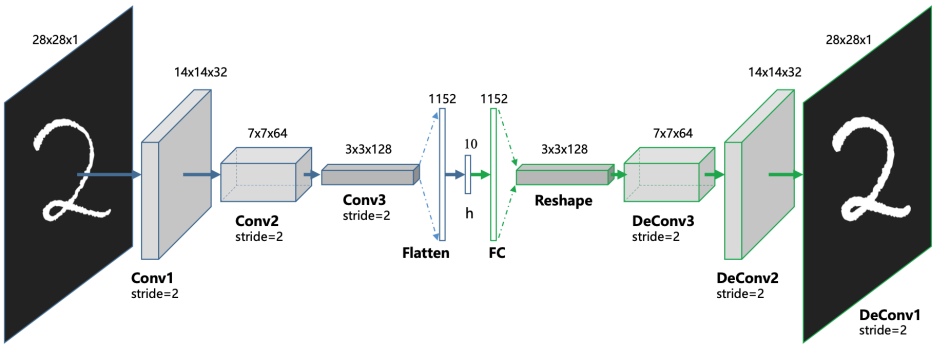

Figure 3.2: Architecture of a convolutional autoencoder (CAE) (Guo et al. 2017).

3.3 Graphs

Definition 1 (Graph) *A graph is a tuple $G = (V, E)$, where V is a finite set of vertices (or nodes), and $E \subseteq V \times V$ is a set of edges. An edge $(u, v) \in E$ represents a connection between vertices u and v.*

The number of nodes and edges in a graph G are denoted as $|V| = n_G$ and $|E| = n_E$, respectively. A graph is called *directed* if the edges are ordered pairs, i.e., $(u, v) \neq (v, u)$. If edge direction is not relevant, the graph is *undirected*.

The *neighbourhood* of a vertex $v \in V$ is the set of adjacent vertices:

$$\mathcal{N}(v) = \{u \in V \mid (v, u) \in E\}.$$

Definition 2 (Adjacency Matrix) *The adjacency matrix of a graph $G = (V, E)$ is a matrix $A \in \mathbb{R}^{n_G \times n_G}$ defined as:*

$$A_{ij} = \begin{cases} 1 & \text{if } (v_i, v_j) \in E, \\ 0 & \text{otherwise.} \end{cases} \tag{3.4}$$

Definition 3 (Labeled Graph) *A graph $G = (V, E)$ is labeled if there exists a labeling function $l : V \to \Sigma$ that assigns to each vertex $v \in V$ a label from a finite alphabet Σ.*

Definition 4 (Attributed Graph) *A graph $G = (V, E)$ is attributed if each vertex $v \in V$ is associated with a feature vector $\mathbf{x}_v \in \mathbb{R}^d$, where d is the dimensionality of the attribute space.*

3.4 Graph Neural Network

Graph Neural Networks (GNNs) are neural network architectures designed to process graph-structured data. The main idea behind GNNs is to iteratively update node representations by aggregating information from local node neighborhoods via the *message passing* mechanism.

Let $G = (V, E)$ be a graph with node features $\mathbf{X} \in \mathbb{R}^{N \times D}$, where N is the number of nodes and D is the input feature dimension. A typical GNN layer updates the representation $\mathbf{h}_v^{(l)}$ of node v at layer l as:

$$\mathbf{h}_v^{(l)} = \text{COMBINE}^{(l)}\left(\mathbf{h}_v^{(l-1)},\ \text{AGGREGATE}^{(l)}\left(\left\{\mathbf{h}_u^{(l-1)} : u \in \mathcal{N}(v)\right\}\right)\right), \tag{3.5}$$

where $\mathcal{N}(v)$ denotes the set of neighbors of node v, $\text{AGGREGATE}^{(l)}$ is an aggregation function (e.g., mean, sum, max pooling, or attention-based aggregation), and $\text{COMBINE}^{(l)}$ is a combination function that can be either linear or non-linear (e.g., sigmoid, ReLU). $\mathbf{h}_v^0$ is initialized as $\mathbf{x}_v$, the feature vector of node v.

GNNs can be applied to various tasks including node classification, link prediction, and graph classification. Variants of GNNs differ in how they design the aggregation and update mechanisms. Examples include Graph Convolutional Networks (Kipf and Welling 2017) and Graph Attention Networks (Veličković et al. 2018).

3.5 Graph Convolutional Network

Graph convolutional networks (GCNs) (Kipf and Welling 2017) are a specific type of GNNs that extend the convolution operation to graphs. The GCN layer updates node representations using the following rule:

$$\mathbf{H}^{(l+1)} = \sigma\left(\hat{\mathbf{D}}^{-\frac{1}{2}}\hat{\mathbf{A}}\hat{\mathbf{D}}^{-\frac{1}{2}}\mathbf{H}^{(l)}\mathbf{W}^{(l)}\right), \tag{3.6}$$

where:

- $\mathbf{H}^{(l)}$ is the matrix of node features at layer l, with $\mathbf{H}^{(0)} = \mathbf{X}$,
- $\hat{\mathbf{A}} = \mathbf{A} + \mathbf{I}$ is the adjacency matrix with added self-loops,
- $\hat{\mathbf{D}}$ is the diagonal degree matrix of $\hat{\mathbf{A}}$,
- $\mathbf{W}^{(l)}$ is the learnable weight matrix for layer l,
- σ is an activation function, typically ReLU.

3.6 Graph Autoencoder

Graph autoencoders (GAEs) (Kipf and Welling 2016) are unsupervised neural network architectures designed to learn low-dimensional node embeddings from graph-structured data. A GAE consists of an encoder that maps nodes to latent representations and a decoder that reconstructs the graph structure (i.e. the adjacency matrix) from these embeddings.

Let $G = (V, E)$ be an *undirected*, *unweighted* graph with adjacency matrix $\mathbf{A}$ and node features $\mathbf{X}$. The encoder of the GAE architecture is often structured as a graph convolutional network (GCN), which encodes the input as:

$$\mathbf{Z} = \mathsf{GCN}(\mathbf{X}, \mathbf{A}), \tag{3.7}$$

where $\mathbf{Z}$ is the matrix of the latent node representations $\mathbf{z}_i$, with dimensions $N \times F$. N is the number of nodes in the graph (i.e., $|V|$), and F is the dimension of the latent space.

On the other hand, the decoder typically reconstructs the adjacency matrix $\hat{\mathbf{A}}$ using an *inner product* between node embeddings:

$$\hat{\mathbf{A}} = \sigma(\mathbf{Z}\mathbf{Z}^T), \tag{3.8}$$

where σ is the sigmoid function. The model is trained by minimizing the binary cross-entropy loss between the original and reconstructed adjacency matrices.

3.7 Variational Graph Autoencoder

Variational graph autoencoders (VGAEs) (Kipf and Welling 2016) extend GAEs by introducing a probabilistic framework based on the variational autoencoder (VAE) architecture (Diederik P Kingma and Welling 2022; Rezende, Mohamed, and Wierstra 2014). Instead of deterministic node embeddings, VGAEs learn a distribution over latent variables for each node.

The encoder outputs the parameters $\boldsymbol{\mu} = \mathsf{GCN}_{\boldsymbol{\mu}}(\mathbf{X}, \mathbf{A})$ and $\log \boldsymbol{\sigma} = \mathsf{GCN}_{\boldsymbol{\sigma}}(\mathbf{X}, \mathbf{A})$ of the posterior distribution:

$$q(\mathbf{Z} \mid \mathbf{X}, \mathbf{A}) = \prod_{i=1}^{N} q(\mathbf{z}_i \mid \mathbf{X}, \mathbf{A}) = \prod_{i=1}^{N} \mathcal{N}(\mathbf{z}_i \mid \boldsymbol{\mu}_i, \mathrm{diag}(\boldsymbol{\sigma}_i^2)). \tag{3.9}$$

The decoder is the same as in GAEs, using the inner product to estimate:

$$p(\mathbf{A} \mid \mathbf{Z}) = \prod_{i=1}^{N} \prod_{j=1}^{N} p(A_{ij} \mid \mathbf{z}_i, \mathbf{z}_j), \tag{3.10}$$

where $p(A_{ij} = 1 \mid \mathbf{z}_i, \mathbf{z}_j) = \sigma(\mathbf{z}_i^T \mathbf{z}_j)$.

The model is trained by minimizing the variational lower bound, w.r.t. the parameters $\mathbf{W}$ of the GCN encoder:

$$\mathcal{L} = \mathbb{E}_{q(\mathbf{Z}|\mathbf{X},\mathbf{A})}[\log p(\mathbf{A} \mid \mathbf{Z})] - \mathrm{KL}[q(\mathbf{Z} \mid \mathbf{X}, \mathbf{A}) \parallel p(\mathbf{Z})], \tag{3.11}$$

where $p(Z)$ is a Gaussian prior:

$$p(\mathbf{Z}) = \prod_{i=1}^{N} p(\mathbf{z}_i) = \prod_{i=1}^{N} \mathcal{N}(\mathbf{z}_i \mid 0, \mathbf{I}). \tag{3.12}$$

3.8 Deep Graph-Level Clustering (DGLC)

Let $\mathcal{G} = \{G_1, G_2, \ldots, G_N\}$ be a set of n graphs, where each graph $G_i = (V_i, E_i)$ is associated with node features $\mathbf{X}_i$, and let $\mathcal{X} = \{\mathbf{X}_1, \mathbf{X}_2, \ldots, \mathbf{X}_n\}$ denote the set of all node features. The goal of the graph clustering problem is to partition $\mathcal{G}$ into disjoint subsets such that graphs within the

same subset are similar, while graphs in different subsets are dissimilar (Cai et al. 2023). The similarity is based on discovering and matching common substructures between the graphs in $\mathcal{G}$.

DGLC (Cai et al. 2023) proposes an end-to-end framework to learn the graph-level embeddings and perform clustering simultaneously, by optimizing the following objective:

$$\mathcal{L}(\phi, \theta) := \mathcal{L}_r(g_\phi(\mathcal{X}, \mathcal{G}), \mathcal{X}, \mathcal{G}) + \mathcal{L}_{c|\theta}(g_\phi(\mathcal{X}, \mathcal{G})). \tag{3.13}$$

where, $\mathcal{L}_r$ is the representation learning objective and $\mathcal{L}_{c|\theta}$ is the clustering objective. The approach uses a graph isomorphism network $g_\phi(\mathcal{X}, \mathcal{G})$ to learn graph-level representations by maximizing the mutual information (MI) between representations of entire graphs and representations of patches (i.e. substructures, such as nodes, edges or triangles) as introduced by (Sun et al. 2020). Specifically, let ϕ denote the parameters of $g_\phi(\mathcal{X}, \mathcal{G})$. After the first l layers of the graph neural network, the input graph G_j is encoded into a set of node representations $\{\mathbf{h}_i^{(l)}\}_{i=1}^{|G_j|}$, where $|G_j|$ denotes the number of nodes in G_j. The approach summarizes features from all depths into patch representations by concatenating them for each node, i.e.:

$$\mathbf{h}_\phi^i = \text{CONCAT}\Big(\{\mathbf{h}_i^{(l)}\}_{l=1}^{L}\Big), \tag{3.14}$$

where L is the number of layers g_ϕ. To obtain a global representation, a readout function is applied to the resulting patch representations:

$$\mathbf{H}_\phi(G_j) = \text{READOUT}\Big(\{\mathbf{h}_\phi^i\}_{i=1}^{|G_j|}\Big). \tag{3.15}$$

The mutual estimator maximizing the MI over $\mathcal{G}$ can be defined as:

$$\hat{\phi}, \hat{\psi} = \arg\max_{\phi,\psi} \sum_{G_j \in \mathcal{G}} \frac{1}{|G_j|} \sum_{i=1}^{|G_j|} I_{\phi,\psi}\Big(\mathbf{h}_\phi^i,\ \mathbf{H}_\phi(G_j)\Big). \tag{3.16}$$

As formulated by (Nowozin, Cseke, and Tomioka 2016), $I_{\phi,\psi}$ can be expressed as:

$$I_{\phi,\psi}\Big(\mathbf{h}_\phi^i(G_j),\ \mathbf{H}_\phi(G_j)\Big) := \mathbb{E}_{\mathbb{P}}\Big[-\text{sp}(-T_\psi(\mathbf{h}_\phi^i(x); \mathbf{H}_\phi(x)))\Big] - \mathbb{E}_{\mathbb{P}\times\tilde{\mathbb{P}}}\Big[\text{sp}(T_\psi(\mathbf{h}_\phi^i(x'); \mathbf{H}_\phi(x')))\Big], \tag{3.17}$$

where T_ψ is a discriminator parameterized by a neural network with parameters ψ, $\mathbb{P}$ denotes the empirical probability distribution over the input space of $\mathcal{G}$, x is a positive input sample, and x' is a negative input sample drawn from $\tilde{\mathbb{P}} = \mathbb{P}$. The function $\text{sp}(z) = \log(1 + e^z)$ denotes the

softplus function.

Building on the work of (Sun et al. 2020), DGLC incorporates a clustering multilayer perceptron (MLP) (Rumelhart, G. E. Hinton, and Williams 1986) network f_θ, parameterized by θ, which is connected to the graph-level representations produced by the previously described architecture. Specifically,

$$\mathbf{z}_j = f_\theta(\mathbf{H}_\phi(G_j)) \in \mathbb{R}^{d_z} \tag{3.18}$$

is the learned cluster embedding for G_j. As proposed by (Xie, Girshick, and Farhadi 2016), the idea is to simultaneously learn a set of C cluster centers $\{\boldsymbol{\mu}_c\}_{c=1}^{C}$ and the parameters θ of the mapping function f_θ, which projects the data points (in this case, graph-level representations) into cluster embeddings $\mathbf{z}_j$. This is achieved by iterating between soft assignment Q and minimizing the Kullback-Leibler (KL) divergence to an auxiliary distribution P. For each cluster embedding $\mathbf{z}_j$ and cluster centroid $\boldsymbol{\mu}_c$, the soft assignment probability q_{jc} is computed as

$$q_{jc} = \frac{(1 + \|\mathbf{z}_j - \boldsymbol{\mu}_c\|^2)^{-1}}{\sum_{c'=1}^{C}(1 + \|\mathbf{z}_j - \boldsymbol{\mu}_{c'}\|^2)^{-1}}. \tag{3.19}$$

(Xie, Girshick, and Farhadi 2016) and (Cai et al. 2023) define the auxiliary distribution P as

$$p_{jc} = \frac{q_{jc}^2 / \sum_j q_{jc}}{\sum_{c'} q_{jc'}^2 / \sum_j q_{jc'}}. \tag{3.20}$$

Hence, the KL divergence is

$$\mathsf{KL}(P \parallel Q) = \sum_{j=1}^{N}\sum_{c=1}^{C} p_{jc} \log \frac{p_{jc}}{q_{jc}}. \tag{3.21}$$

Finally, the objective optimized by DGLC, shown in Equation 3.13, for a batch of graphs $\overline{G} \subseteq \mathcal{G}$ of size N_b is given by:

$$\mathcal{L}_{\text{batch}}(\phi, \psi, \theta) = \underbrace{-\frac{1}{|\bar{G}|}\sum_{i \in \bar{G}} I_{\phi,\psi}(\mathbf{h}_\phi^i, \mathbf{H}_\phi(\bar{G}))}_{L_{r|\phi,\psi}} + \underbrace{\sum_{j=1}^{N_b}\sum_{c=1}^{C} p_{jc} \log \frac{p_{jc}}{q_{jc|\phi,\theta}}}_{L_{c|\phi,\theta}}. \tag{3.22}$$

3.9 FEATHER

Let $G = (V, E)$ be an undirected attributed graph with n nodes, each associated with a feature vector $\mathbf{x}_v \in \mathbb{R}^k$, and let the matrix of all node features be denoted by $\mathbf{X} \in \mathbb{R}^{n \times k}$. FEATHER (Rozemberczki and Sarkar 2020) defines characteristic functions to describe the distribution of features in the neighborhood of a node, using random-walk based affinity weights. For a fixed scale $r \in \mathbb{N}$, the r-step normalized random walk matrix is given by $\hat{\mathbf{A}}^r$, where $\hat{\mathbf{A}} = \mathbf{D}^{-1}\mathbf{A}$ is the row-normalized adjacency matrix.

Let $\boldsymbol{\Theta}^{(1)}, \ldots, \boldsymbol{\Theta}^{(k)} \in \mathbb{R}^d$ be the evaluation point vectors, one for each feature dimension. For a node $u \in V$, a feature index $j \in \{1, \ldots, k\}$, and an evaluation point $\theta \in \mathbb{R}$, the method defines the r-scale random walk weighted characteristic function as:

$$\phi_u^{(j,r)}(\theta) = \sum_{w \in V} \hat{\mathbf{A}}^r_{u,w} \cdot e^{i\theta \mathbf{X}_{w,j}}. \tag{3.23}$$

The real and imaginary components are given by:

$$\mathrm{Re}[\phi_u^{(j,r)}(\theta)] = \sum_{w \in V} \hat{\mathbf{A}}^r_{u,w} \cos(\theta \mathbf{X}_{w,j}), \quad \mathrm{Im}[\phi_u^{(j,r)}(\theta)] = \sum_{w \in V} \hat{\mathbf{A}}^r_{u,w} \sin(\theta \mathbf{X}_{w,j}) \tag{3.24}$$

For d evaluation points $\boldsymbol{\Theta}^{(j)} = (\theta_1^{(j)}, \ldots, \theta_d^{(j)})$, the method evaluates:

$$\mathrm{Re}[\phi_u^{(j,r)}(\boldsymbol{\Theta}^{(j)})] \in \mathbb{R}^d, \quad \mathrm{Im}[\phi_u^{(j,r)}(\boldsymbol{\Theta}^{(j)})] \in \mathbb{R}^d, \tag{3.25}$$

with entries

$$\Big[\mathrm{Re}[\phi_u^{(j,r)}(\boldsymbol{\Theta}^{(j)})]\Big]_l = \sum_{w \in V} \hat{\mathbf{A}}^r_{u,w} \cos\Big(\theta_l^{(j)} \mathbf{X}_{w,j}\Big), \quad \Big[\mathrm{Im}[\phi_u^{(j,r)}(\boldsymbol{\Theta}^{(j)})]\Big]_l = \sum_{w \in V} \hat{\mathbf{A}}^r_{u,w} \sin\Big(\theta_l^{(j)} \mathbf{X}_{w,j}\Big). \tag{3.26}$$

Each pair of real and imaginary outputs forms a $2d$-dimensional descriptor per feature and scale. By concatenating across all k features and all scales $r = 1, \ldots, r_{\max}$, the method obtains a final embedding $\mathbf{z}_u \in \mathbb{R}^{2kdr_{\max}}$ for each node $u \in V$.

To derive a graph-level representation, the node embeddings are aggregated using a permutation-invariant function such as the mean:

$$\mathbf{z}_G = \frac{1}{n} \sum_{u \in V} \mathbf{z}_u \tag{3.27}$$

This representation is invariant to node permutations, robust to small perturbations in individual node features and it can be directly used in downstream tasks such as graph classification or clustering.

3.10 Clustering algorithms

Clustering is a type of unsupervised machine learning technique used to group similar data points into distinct subsets or clusters, based on their features. This section presents k-means, DBSCAN and Spectral clustering.

3.10.1 k-means algorithm (Lloyd's algorithm)

Given a dataset $\mathcal{D}$ of n data points and a set of k initial centroids $\boldsymbol{\mu}_1^{(1)}, \boldsymbol{\mu}_2^{(2)}, \ldots, \boldsymbol{\mu}_k^{(1)}$, the *k-means algorithm* (Lloyd 1982) is an iterative process consisting of the following alternating steps:

1. **Assignment step**: Assign each data point $\boldsymbol{x}_i \in \mathcal{D}$ to the nearest centroid:

$$c_i^{(t)} = \arg \min_{j \in \{1,\ldots,k\}} \|\boldsymbol{x}_i - \boldsymbol{\mu}_j^{(t)}\|^2, \tag{3.28}$$

   where $c_i^{(t)}$ is the cluster assignment for point $\boldsymbol{x}_i$ at iteration t, and $\|\cdot\|$ denotes the Euclidean distance.

2. **Update step**: Recompute the centroids as the mean of the data points assigned to each cluster:

$$\boldsymbol{\mu}_j^{(t+1)} = \frac{1}{n_j^{(t)}} \sum_{\boldsymbol{x}_i \in \mathcal{C}_j^{(t)}} \boldsymbol{x}_i, \tag{3.29}$$

   where $\mathcal{C}_j^{(t)}$ denotes the set of data points assigned to cluster j at iteration t, and $n_j^{(t)}$ is the number of points in the cluster.

These steps are repeated until convergence, typically when the centroids no longer change significantly or a predefined number of iterations is reached.

3.10.2 DBSCAN (Density-Based Spatial Clustering of Applications with Noise)

DBSCAN (Ester et al. 1996) is a density-based clustering algorithm that groups together points in close proximity, while marking as outliers points found alone or with few neighbors. It requires two parameters:

- ε: The maximum distance between two points for one of the points to be considered in the neighborhood of the other.
- `min_samples`: The minimum number of points in the ε-neighborhood of a point required to start a cluster. In other words, this is the minimum number of points needed to form a *dense region*.

Based on the parameters, DBSCAN categorizes data points into three different types:

- *Core points*: A point P is considered a *core point* there are at least `min_samples` points (including P) in its ε-neighborhood.
- *Border points*: A point Q is labeled as a *border point* if it is in the neighborhood of a core point, but it does not itself have enough neighbors to be considered a core point.
- *Noise points (outliers)*: A point R is considered an *noise point* if it is neither a core nor a border point.

Clusters are formed by connecting core points that are density-reachable, meaning that there exists a chain of core points where each point is within the ε-neighborhood of the next. Border points are then assigned to the nearest core point's cluster.

The advantage of DBSCAN is that the number of clusters does not need to be specified in advance, unlike in *k-means* or *Spectral clustering*. Additionally, the algorithm can find clusters of arbitrary shapes and identify and handle outliers. One disadvantage is that DBSCAN is not fully deterministic with respect to border points that can be reached from multiple clusters, as their assignment depends on the order in which data points are processed.

3.10.3 Spectral clustering

Spectral clustering is an unsupervised algorithm that partitions data based on the structure of a similarity graph derived from pairwise relationships between samples. Given a set of data points $\mathbf{x}_1, \mathbf{x}_2, \ldots \mathbf{x}_n$ and a notion of similarity $s(\mathbf{x}_i, \mathbf{x}_j) \geq 0$ between points x_i and x_j, the algorithm performs the following steps:

1. Construct a similarity graph G from the data, where each node corresponds to a single data point. The graph G is *weighted*, i.e., each edge is assigned a weight w_{ij} reflecting the similarity between nodes v_i and v_j. There are three common ways to construct the similarity graph: ε-neighborhood graphs, k-nearest neighbor graphs and fully connected graphs as presented in (Luxburg 2007). The ε-neighborhood similarity graph connects all points whose pairwise distances are less than ε, resulting in an *unweighted* graph. The k-nearest neighbor graph connects each note to its k-nearest neighbors, weighting the edges

by the similarity of the vertices. Finally, a fully connected graph, connects all vertices with weights given by a similarity function as the Gaussian kernel:

$$s(\mathbf{x}_i, \mathbf{x}_j) = \exp\Big(\frac{-\|\mathbf{x}_i - \mathbf{x}_j\|^2}{2\sigma^2}\Big). \tag{3.30}$$

2. Compute the degree matrix $\mathbf{D}$ and weighted adjacency (i.e., similarity) matrix $\mathbf{W}$, and define the unnormalized graph Laplacian as:

$$\mathbf{L} = \mathbf{D} - \mathbf{W}. \tag{3.31}$$

Alternatively, the normalized Laplacian can be used:

$$\mathbf{L}_{\text{sym}} = \mathbf{I} - \mathbf{D}^{-1/2}\mathbf{W}\mathbf{D}^{-1/2}, \tag{3.32}$$

resulting in two variations of the algorithm proposed by (Shi and Malik 2000) and (Ng, Jordan, and Weiss 2001).

3. Compute the first k eigenvectors $\mathbf{u}_1, \mathbf{u}_2, \dots \mathbf{u}_k$ of $\mathbf{L}$, where k is the number of clusters to construct. These eigenvectors are stacked column-wise to form the matrix $\mathbf{U} \in \mathbb{R}^{n \times k}$.

4. Apply the k-means algorithm to cluster the rows of $\mathbf{U}$.

Spectral clustering is effective for identifying non-convex clusters and structures that are not easily separable in the original feature space.

3.11 Clustering performance evaluation

This section presents a few clustering evaluation metrics including the Silhouette coefficient, the Calinski-Harabasz index, the Davies-Bouldin index, the adjusted Rand index and the adjusted mutual information score. Intuitively, better scores indicate non-overlapping, well-separated and dense clusters.

3.11.1 Silhouette Coefficient

The *silhouette* (Rousseeuw 1987) is a measure of how similar an object is to its own cluster compared to other clusters.

Given a dataset $\mathcal{D}$ of n points assigned to k clusters $(\mathcal{C}_1, \mathcal{C}_2, \dots, \mathcal{C}_k)$, let n_i be the number of points in $\mathcal{C}_i$. The *silhouette value* of a data point $\mathbf{x} \in \mathcal{C}_i$, is defined as

$$s(\mathbf{x}) = \frac{b(\mathbf{x}) - a(\mathbf{x})}{\max\{a(\mathbf{x}), b(\mathbf{x})\}}, \tag{3.33}$$

where

$$a(\mathbf{x}) = \frac{1}{n_i - 1} \sum_{\mathbf{y} \in \mathcal{C}_i, \mathbf{y} \neq \mathbf{x}} d(\mathbf{x}, \mathbf{y}) \tag{3.34}$$

is the average distance between the sample $\mathbf{x}$ and all other points in the cluster $\mathcal{C}_i$, and

$$b(\mathbf{x}) = \min_{j \neq i} \frac{1}{n_j} \sum_{\mathbf{y} \in \mathcal{C}_j} d(\mathbf{x}, \mathbf{y}) \tag{3.35}$$

is the average distance between the sample $\mathbf{x}$ and all other points in the nearest cluster $\mathcal{C}_j \neq \mathcal{C}_i$.

The *silhouette coefficient* for a set of samples is defined as the average of the *silhouette values* of the individual samples from the set. The coefficient can take values between -1 and 1. A value close to 1 indicates that the data-points are well-clustered, with a good separation from other clusters, while a value close to -1 indicates incorrect clustering.

3.11.2 Calinski-Harabasz Index

The *Calinski-Harabasz index* (Caliński and JA 1974), also known as the *Variance Ratio Criterion*, measures clustering quality by comparing the between-cluster dispersion to the within-cluster dispersion, with higher values indicating better-defined clusters.

Given a dataset $\mathcal{D}$ of n points assigned to k clusters $(\mathcal{C}_1, \mathcal{C}_2, \ldots, \mathcal{C}_k)$, let $\boldsymbol{\mu}_i$ denote the centroid of the i-th cluster, n_i the number of points in $\mathcal{C}_i$, and $\boldsymbol{\mu}$ the centroid of all points in $\mathcal{D}$. The *Calinski-Harabasz index* is defined as

$$\mathsf{CH} = \frac{\mathsf{BCSS}}{\mathsf{WCSS}} \cdot \frac{n - k}{k - 1}, \tag{3.36}$$

where

$$\mathsf{BCSS} = \sum_{i=1}^{k} n_i \|\boldsymbol{\mu}_i - \boldsymbol{\mu}\|^2 \tag{3.37}$$

is the weighted sum of the squared distances between each cluster centroid and the centroid of all points in the dataset, and

$$\mathsf{WCSS} = \sum_{i=1}^{k} \sum_{\mathbf{x} \in \mathcal{C}_i} \|\mathbf{x} - \boldsymbol{\mu}_i\|^2 \tag{3.38}$$

is the sum of the squared distances between the data points and the respective cluster centroids.

3.11.3 Davies-Bouldin Index

The *Davies-Bouldin index* (Davies and Bouldin 1979) is a metric used to evaluate the quality of clustering by measuring the average similarity ratio of each cluster with the cluster that is most similar to it, where a lower index indicates better clustering.

Given a dataset $\mathcal{D}$ of n points assigned to k clusters $(\mathcal{C}_1, \mathcal{C}_2, \ldots, \mathcal{C}_k)$, let $\boldsymbol{\mu}_i$ denote the centroid of the i-th cluster and n_i the number of points in $\mathcal{C}_i$. The *Davies-Bouldin index* is defined as

$$\mathsf{DB} = \frac{1}{k} \sum_{i=1}^{k} \max_{j \neq i} \frac{s_i + s_j}{d_{i,j}}, \tag{3.39}$$

where

$$s_i = \frac{1}{n_i} \sum_{\mathbf{x} \in \mathcal{C}_i} \|\mathbf{x} - \boldsymbol{\mu}_i\| \tag{3.40}$$

is the average distance between each point in $\mathcal{C}_i$ and $\boldsymbol{\mu}_i$, and

$$d_{i,j} = \|\boldsymbol{\mu}_i - \boldsymbol{\mu}_j\|. \tag{3.41}$$

is the distance between cluster centroids $\boldsymbol{\mu}_i$ and $\boldsymbol{\mu}_j$.

3.11.4 Adjusted Rand Index

The *Rand index* (Rand 1971) is a measure of similarity between the cluster assignments given by two clustering approaches. Given a dataset $X = \{x_1, x_2, \ldots x_n\}$ of n data points, and two partitions: $A = \{A_1, A_2, ...A_k\}$ of k subsets (clusters), and $B = \{B_1, B_2, \ldots B_l\}$ of l subsets (clusters), let:

- s be the number of pairs of data points in X that are assigned to the same subset (cluster) of A and the same subset (cluster) of B. In other words, s is the number of pairs of points that are clustered together by both approaches.
- d be the number of pairs of data points in X that are assigned to different subsets of A and different subsets subsets of B.

The Rand index (RI) is defined as:

$$\mathsf{RI} = \frac{s + d}{\binom{n}{2}}, \tag{3.42}$$

where $\binom{n}{2}$ is the total number of pairs of points in X. It can take values between 0 and 1 (perfect

agreement). However, the Rand index does not account for chance; that is, random assignments can sometimes result in high RI values, which can be misleading. The *adjusted Rand index* (ARI) mitigates this issue and it is defined as:

$$\mathsf{ARI} = \frac{\mathsf{RI} - \mathbb{E}(\mathsf{RI})}{\max(\mathsf{RI}) - \mathbb{E}(\mathsf{RI})}, \tag{3.43}$$

where $\mathbb{E}(\mathsf{RI})$ is the expected value of the Rand index for random cluster assignments. Hence, the ARI takes values between -0.5 and 1. A negative ARI score means that the assignment is worse than random, while a value of 1 indicates perfect agreement.

3.11.5 Adjusted Mutual Information Score

The adjusted mutual information score (Vinh, Epps, and Bailey 2009) is an adjusted version of the mutual information, which can be used to compare the cluster assignments given by two clustering approaches. Given a dataset $X = \{x_1, x_2, \dots x_n\}$ of n data points, and two partitions: $A = \{A_1, A_2, ...A_k\}$ of k subsets (clusters), and $B = \{B_1, B_2, \dots B_l\}$ of l subsets (clusters), let:

- $\Pr_A(i) = \frac{|A_i|}{n}$ be the probability that a data point is assigned to cluster A_i.
- $\Pr_B(j) = \frac{|B_j|}{n}$ be the probability that a data point is assigned to cluster B_j.
- $\Pr_{AB}(i,j) = \frac{|A_i \cap B_j|}{n}$ be the probability that a data point is assigned to A_i in A and B_j in B.

The mutual information (MI) between the two clustering assignments (i.e., partitions) is defined as:

$$\mathsf{MI} = \sum_{i=1}^{k} \sum_{j=1}^{l} \mathsf{Pr}_{AB}(i,j) \log \frac{\Pr_{AB}(i,j)}{\Pr_A(i) \cdot \Pr_B(j)}. \tag{3.44}$$

Similarly, the adjusted mutual information (AMI) score can defined as:

$$\mathsf{AMI} = \frac{\mathsf{MI} - \mathbb{E}(\mathsf{MI})}{\max(\mathsf{MI}) - \mathbb{E}(\mathsf{MI})}, \tag{3.45}$$

where $\mathbb{E}(\mathsf{MI})$ is the expected mutual information between two random clustering assignments.

4 Data

For the purposes of this thesis, a dataset of measurements from 10,308 outdoor 5G mobile cells was collected from Swisscom's live network. The measurements include performance and quality indicators such as throughput, signal strength, and interference recorded directly as experienced by the users. Rather than aggregating the measurements at cell level, the geographical area covered by each cell is divided into fixed 200 meter by 200 meter regions, over which the measurements are averaged and reported. These fixed-size regions are referred to as *tiles*, and in total, the dataset contains 150,049 of them. It is important to mention that one tile can have aggregated measurements for more than one different cell. This is because multiple cells can serve the same geographical region, with different users connected to different cells. In this format, the data provides finer-grained insights into each cell's performance and can be used to detect localized issues within a single cell.

4.1 Performance metrics

The dataset includes a few key performance (KPIs) and quality indicators (KQIs), which describe signal quality and user experience.

Throughput The throughput can be defined as the rate at which data is successfully transmitted over a communication channel within a given period of time. The throughput experienced by the end users is aggregated per tile for each cell. However, the measure used by Swisscom for performance evaluation is not the raw number, but rather the ranking. For each tile, the cells serving it are ranked from best to worst based on the average throughput of their users. If the cell is among the top five cells serving a tile in terms of user throughput, its performance is considered good. Hence, the *throughput dominance* measure reflects the cell's competitiveness in an area, and it can take binary values (i.e. *good* and *bad*, as shown in Figure (4.1)).

Signal-to-Interference-plus-Noise Ratio The signal-to-interference-plus-noise ratio (SINR) is defined as the ratio of the power of the signal of interest to the sum of the power of interference and noise. It is typically measured in decibels (dBs) and quantifies the quality of the received

	Throughput dominance
Good	Cell is among top 5 cells serving a tile in terms of user throughput.
Bad	Cell is not among top 5 cells serving a tile in terms of user throughput.

	SINR (dB)	RSRP (dBm)	RSRQ (dB)
Excellent	more than 20	more than -85	more than -10
Good	13 to 20	-94 to -85	-15 to -10
Medium	0 to 13	-107 to -94	-20 to -15
Weak	less than 0	less than -107	less than -20

Figure 4.1: Reference table for throughput, SINR, RSRP and RSRQ

signal. Figure (4.1) shows the SINR ranges considered weak, medium, good, and excellent, respectively.

Reference Signal Received Power The reference signal received power (RSRP) is defined as the average power of the received reference signals in the downlink wireless channel of a mobile network. The strength of the received reference signals is used for channel estimation and handover decisions, ensuring the user is connected to the cell with the strongest signal power at a given location. The RSRP is measured in decibel-milliwatts (dBms), and its ranges are shown in Figure (4.1).

Reference Signal Received Quality The reference signal received quality (RSRQ) is defined as the ratio between the received signal strength indicator (RSSI) and the reference signal received power (RSRP), scaled by the number of physical resource blocks. It is an indication of the quality of the received reference signals and, like RSRP, it can be used to assess the connection quality and the need for a handover. The RSRQ is measured in decibels (dB), and its reference values are given in Figure (4.1).

4.2 Data preparation

For the purposes of this thesis, Swisscom provided multiple data sources containing information about the recorded performance metrics and the configuration of their mobile cells. The data was collected within the span of one week, and the observations were aggregated for each tile and cell pair. After standard data preparation procedures, such as merging, dropping missing or invalid values and keeping features relevant for this project, the resulting dataframe contains 10,308 unique cells and 150,049 unique tiles. Additionally, following common practices by Swisscom radio network engineers, tiles with less than 5 observations were dropped, as well as tiles contributing to less than 95% of the total traffic.

As mentioned before, each tile is a fixed-size 200 meter by 200 meter region, uniquely defined by its geographical location, i.e. its latitude and longitude coordinates. One tile can have multiple separate KPI aggregations for each of the cells serving it. Consequently, each record of the processed dataframe describes a measurement of a tile with respect to a cell, and it contains the following information: first, cell-related attributes such as

- *Cell identifier*: Unique string identifier for each mobile cell.
- *Cell location*: Location of the cell specified by its latitude and longitude coordinates. Precisely, this is the location of the physical site where the antenna is positioned.
- *Cell range*: Coverage radius of the cell given in meters (m).
- *Beam direction*: Defines the direction of the beam of the antenna relative to the north direction in degrees (e.g., north = 0°, east = 90°, etc.).
- *Sector angle*: Gives the width of the beam (or circular sector) in angles. It is measured in degrees.

and second, tile-related attributes such as

- *Tile location*: Location of the tile given by its latitude and longitude coordinates. Precisely, this is the location of the center of the square.
- *Distance from cell site*: The distance from the tile center to the site where the antenna is located. The distance is given in meters (m).
- *Relative angle from cell site*: The azimuth of the tile relative to the cell's beam direction, measured in degrees.
- *Number of observations*: Number of distinct measurements recorded from users in the tile connected to the cell of interest.

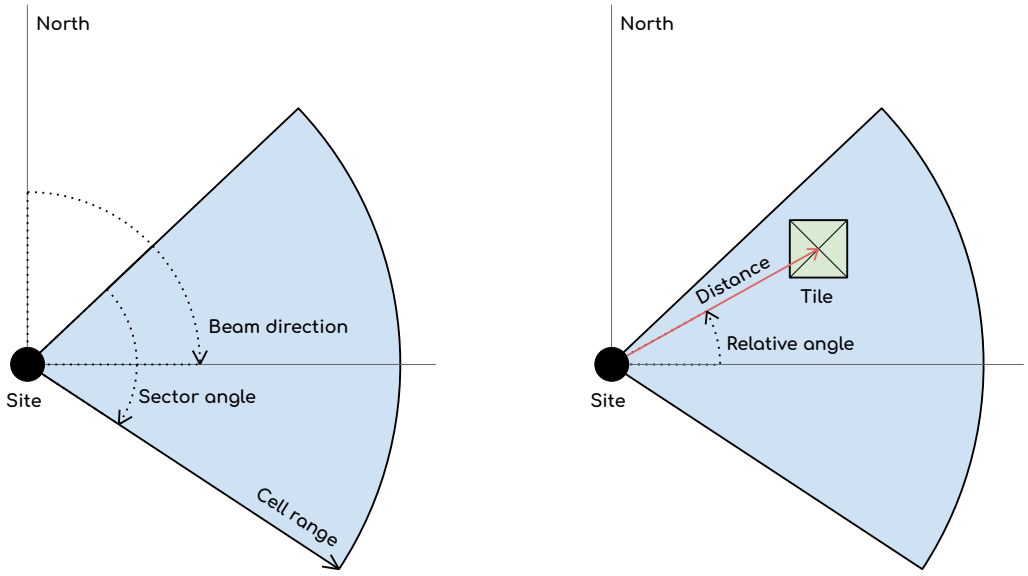

Figure 4.2: Cell-related (left) and tile-related (right) attributes

- *Average SINR, RPSP and RSRQ*: The averages of the recorded KPI measurements from the users in the tile connected to the cell of interest.
- *Throughput dominance*: A categorical value describing the performance of the tile in terms of throughput as described in Section (4.1)

A graphical overview of some of the cell-related and tile-related attributes is shown in Figure (4.1).

Although SINR, RSRP, and RSRQ are continuous measures, radio network engineers are more concerned with whether they lie within a certain range than with their exact values. Hence, as the final step of the data preparation process they are mapped into categorical values corresponding to the standard levels considered weak, moderate, good and excellent given in Figure (4.2).

4.3 Exploratory data analysis

To understand the distribution of the features in the dataset, standard data description and visualization techniques were used. One interesting result is given in Figure 4.3. The plot shows the Pearson and Spearman correlation matrices between the average SINR, average RSRP and average RSRQ of each tile and mobile cell pair in the dataset. The Pearson correlation coefficient (Pearson and Galton 1895) computes the linear correlation between two datasets, while the

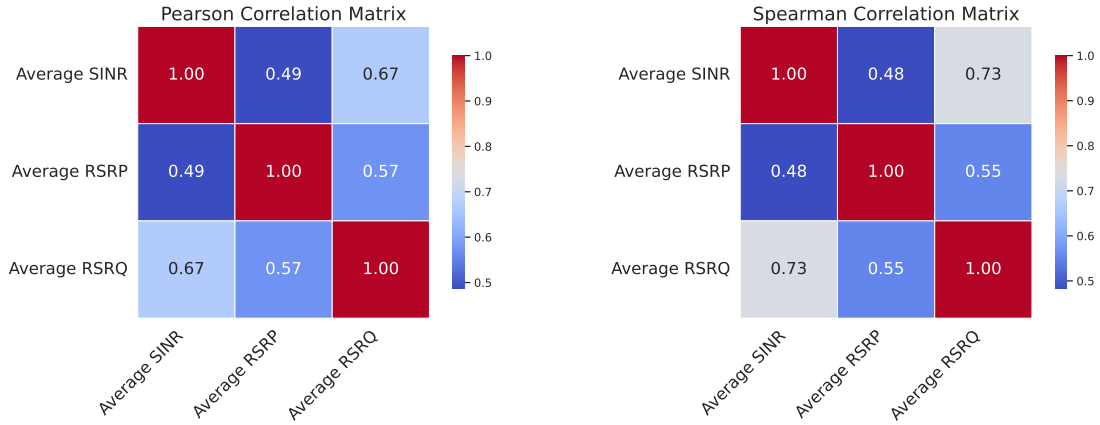

Figure 4.3: Pearson and Spearman correlation matrices for the average SINR, RSRP and RSRQ

Spearman correlation coefficient (Spearman 1904) measures the correlation between two sets of ranks.

The average SINR and the average RSRQ have a *high* correlation according to the Pearson correlation coefficient (i.e., the coefficient lies between $\pm; 0.5$ and $\pm; 0.7$, which (Kuckartz et al. 2013) characterizes as *high*) and a *very high* correlation according to the Spearman correlation coefficient (i.e., the value falls between $\pm; 0.7$ and $\pm; 1$, which (Kuckartz et al. 2013) characterizes as *very high*). Similarly, both the Pearson and Spearman correlation show *high* correlation between the average RSRP and average RSRQ. On the other hand, the correlation between the average SINR and the average RSRP is *moderate* according to both scores (i.e., in the range between $\pm$ 0.3 and $\pm$ 0.5. Since the average RSRQ is highly correlated with both the average SINR and average RSRP, it will not be used in the further analysis for simplicity.

5 Methods

This section describes the methods used to obtain clusters of cells with similar performance characteristics, including the choice of data representation, embedding techniques and clustering algorithms. In addition, given the unsupervised nature of the problem an evaluation pipeline is proposed.

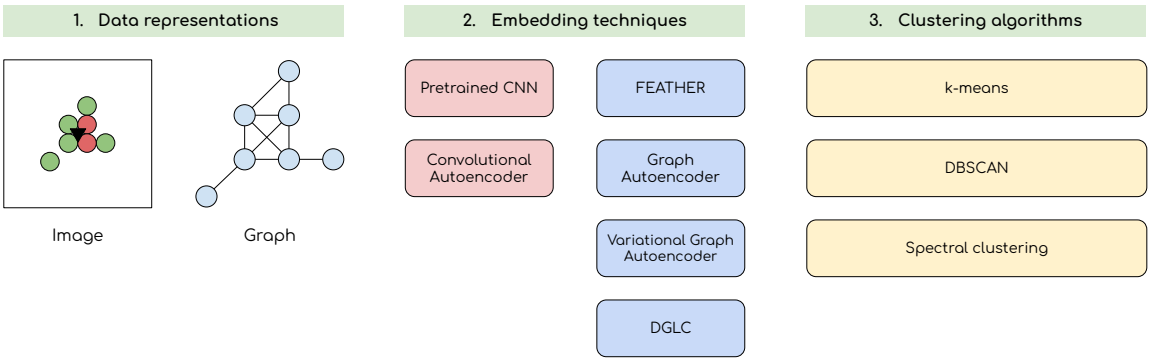

Figure 5.1: Methodology overview

Figure 5.1 provides an overview of the methodology, consisting of three main steps. Given the dataset of mobile cell performance data described in Section 4, the first step is to find suitable structured representations. The two natural choices considered are *image-based* and *graph-based* representations, i.e., representing the data corresponding to each mobile cell as an image or a graph, respectively. For each of the two data representations, the second step involves applying an embedding learning technique to obtain compact vector representations of the mobile cells. Specifically, for images, a pretrained CNN and a convolutional autoencoder were applied, whereas for graphs, FEATHER, a graph autoencoder, a variational graph autoencoder, and the DGLC framework were used.

The third and final step is the use of clustering algorithms on the resulting embeddings in order to group similar mobile cells. Three different clustering algorithms were considered: k-means, DBSCAN, and Spectral clustering. k-means was selected due to its simplicity, efficiency, and

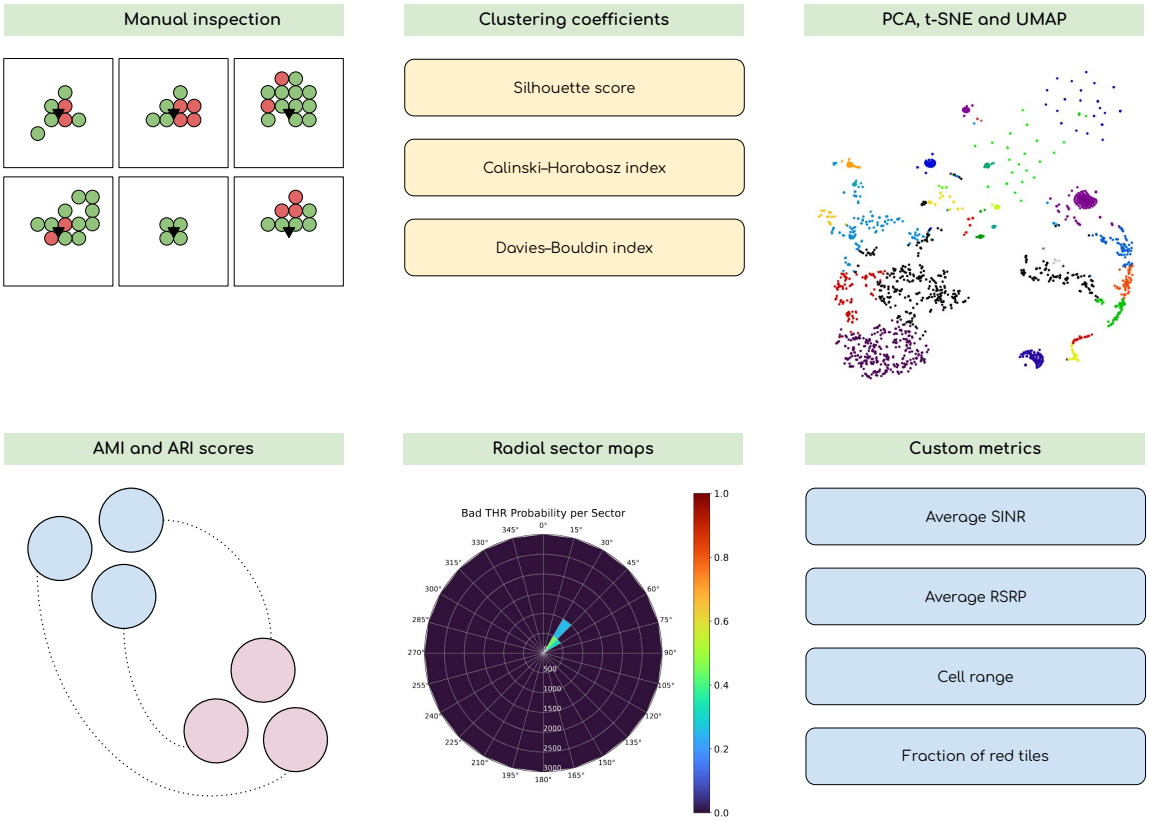

Figure 5.2: Evaluation overview

widespread use in clustering tasks, particularly when the number of clusters is known or can be reasonably estimated. DBSCAN was included for its ability to discover clusters of arbitrary shape and to identify noise or outliers, which is useful when the cluster structure is irregular or not well-separated. Finally, Spectral clustering was chosen for its strength in handling non-convex clusters.

Figure 5.2 gives an overview of the evaluation pipeline used to interpret the results. The pipeline consists of manual inspections, clustering coefficients and different visualizations. A detailed explanation of each step is provided in Section 5.4.

5.1 Data representation

Considering the properties of the dataset described in Section 4, the first step is to select an appropriate representation of the mobile cell data. We consider two types of representations: *image-based* and *graph-based* representations.

5.1.1 Image-based representation

As the name suggests, the data from each mobile cell, i.e. the KPI measurements from the respective tiles, is represented as an image. This representation approach was inspired by an existing Swisscom tool used to analyze the performance of mobile cells. The tool generates interactive footprint maps showing the KPI measurements of the tiles associated with the cell of interest, as shown in Figure 5.3. In these maps, the cell is represented by a black triangle, and each tile is drawn as a circle. The throughput dominance of each tile is indicated by color, as discussed in Section 4.1. Other metrics and configuration parameters can be viewed by clicking on different elements (e.g. the site, or tiles) of the maps. Among other tools, radio network engineers use the footprints to assess the performance of a given mobile cell, identify the root cause of an issue, and propose a solution.

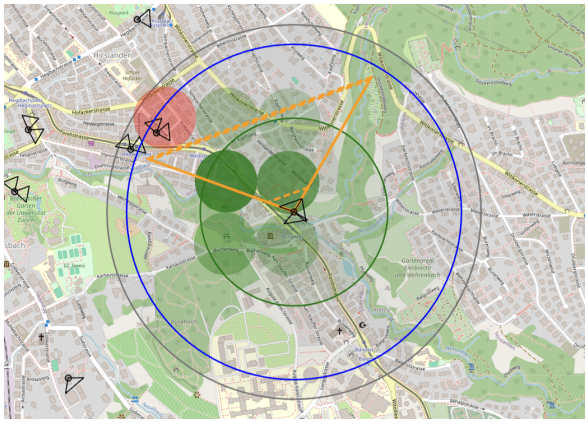

Figure 5.3: An example of an interactive footprint map generated by an internal Swisscom tool

The image-based representations used in this thesis were constructed in a similar fashion. First, the site where the cell is located was placed at the center of the image. Then, the image was rotated so that the beam direction points toward geographical north, for consistency across mobile cell with different beam directions. Furthermore, the geographical context, i.e., the map, was removed for simplicity. It is worth noting that the geographical location can influence the performance of mobile cells, as discussed in (Shibli and Zanouda 2024), but this aspect was not fully explored in this thesis. Some preliminary experimental setups and ideas for how to use this information are provided in Appendix A.6. The tiles are shown as circles on the images, with KPI measurements encoded by color. For example, with throughput dominance, *red* indicates poor performance (i.e., the cell is not among the top 5 servers for the tile), while *green* indicates good performance. Similar layers are created for SINR and RSRP where *red*, *orange*, *yellow* and *green* colors indicate weak, medium, good and excellent performance respectively. Figure 5.4 illustrates an example.

As described in detail later, the image-based representations are used to create embedding vectors, i.e. latent representations of the mobile cells. In some cases, embeddings were created using a single KPI layer, such as the throughput dominance. In other cases, all three KPI metrics were combined by stacking the images into a multi-channel tensor before passing it into a convolutional neural network.

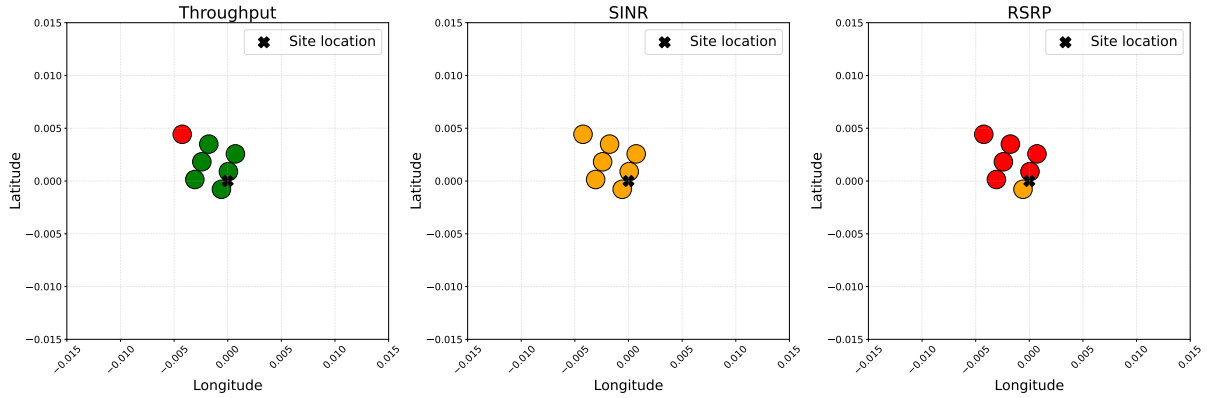

Figure 5.4: Example of an image-based representation of a single mobile cell. The cell site is centered and the image is rotated so the beam direction points north. Tiles are shown as circles, and KPI values are shown using color. The three images represent different KPIs: throughput doominance, SINR, and RSRP.

5.1.2 Graph-based representation

As an alternative representation method, the data from each mobile cell can be modeled as an *attributed graph*. The motivation behind this approach is twofold. First, there is the question of efficacy of using color to encode different magnitudes of the KPI measurements. Second, there is the complexity introduced by having multiple KPI layers (or images) per mobile cell. In contrast, attributed graphs offer a unified way to represent all KPIs in a single structure.

An attributed graph is designed to capture both the spatial structure of the tiles and their performance characteristics. The nodes in the graph correspond to individual tiles. Each node is associated with a feature vector that includes information about the tile's position relative to the site, as well as its performance metrics. Specifically, the node features include:

- Distance from the site in meters (m).
- Relative angle from the site in degrees.
- Throughput class: '0' for *bad*, '1' for *good* performance.
- SINR class: '0' for *weak*, '1' for *medium*, '2' for *good*, '3' for *excellent* performance.
- RSRP class: '0' for *weak*, '1' for *medium*, '2' for *good*, '3' for *excellent* performance.

Edges are created between *neighboring tiles* that are within a distance of no more than 300 meters from each other. There are no edge features, as encoding the exact distance between the centers of the tiles does not add value. An example graph is shown in Figure 5.5, corresponding to the mobile cell shown in Figures 5.3 and 5.4 to illustrate the connection between the two formats.

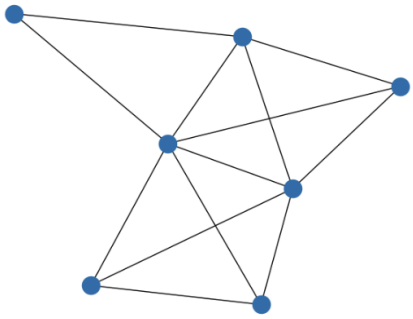

Figure 5.5: Example of an graph-based representation of a single mobile cell. Tiles are represented as nodes, and KPI measurements as node features. Edges are drawn between neighboring tiles.

As will be described later, the graph-based representations are also used to create embedding vectors. For this reason, standard data preparation techniques such as adding self-loops and normalizing node features (specifically, using min-max scaling) were applied before passing the graph data into a graph neural network.

5.2 Embedding techniques

The previous section presented two representation methods for mobile cell data: *image-based* and *graph-based* representations. This section explores strategies for learning *embedding vectors* from the aforementioned representations. These *latent representations* will be used as input to different clustering algorithms, identifying clusters of cells with similar performance characteristics.

5.2.1 Methods for image-based representations

The methods applied to the image-based representations of the cell data include: pretrained convolutional neural network and convolutional autoencoder.

Pretrained CNN To obtain embeddings from the mobile cell image-based representations a pretrained VGG-16 convolutional neural network was used. VGG-16 (Simonyan and Zisserman 2015) is a type of a deep convolutional network with 16 layers. It was pretrained on the ImageNet dataset (Deng et al. 2009), which contains 1,000 object classes. The last classification layer of the network was removed, so that it can be used as a feature extractor. The output of the model was a 2,048-dimensional vector, which was further reduced to a 25-dimensional embedding using *principal component analysis* (PCA) (Jolliffe 2005). PCA is a statistical technique used to reduce the dimensionality of data by transforming it into a new set of uncorrelated variables called *principal components*, which retain most of the original variance.

It's important to note that the pretrained CNN architecture could only process a single KPI

image-representation (i.e. 3 channels). In other words, the embeddings were derived based on the throughput dominance performance of the mobile cells. Additionally, the data was filtered to contain only images with at least one red throughput dominance tile, resulting in 2207 images in total. Since images contain only red or green tiles, green tiles were removed for simplicity. This directs the model's focus to the position of red tiles, which is more relevant for detecting spatial patterns. The entire pipeline is illustrated in Figure 5.6.

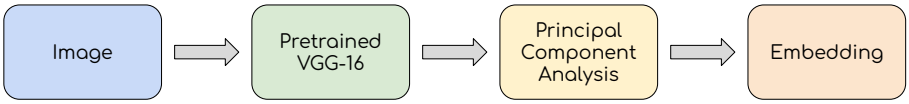

Figure 5.6: An illustration of the pipeline with a pretrained CNN

Convolutional autoencoder A convolutional autoencoder (CAE) was trained on the dataset of mobile cell image-based representations. As described in Section 3.2, CAEs fall under unsupervised learning architectures, in which the model learns to reconstruct the input data using a *decoder*, based on a compressed representation produced by the *encoder*. Both the encoder and decoder consist of three convolutional layers (the decoder uses transposed convolutions). The compressed representation is optimized to retain the most important features needed to reconstruct the input data, and it is used as an embedding to represent the mobile cell. The full pipeline is shown in Figure 5.7.

The CAE architecture can process both single KPI image-representations (i.e. 3 channels) and multiple KPI image-representations (i.e. 9 channels composed of the stacked throughput dominance, SINR and RSRP image layers). Accordingly, two variants of the CAE were trained. The first variant was trained on throughput dominance images containing at least one red tile, similar to the pretrained CNN scenario. The second variant was trained on 9-channel images, i.e., stacked throughput dominance, SINR, and RSRP layers. In both settings, the models were trained by minimizing the MSE loss between the original and reconstructed images. Training was performed on a single L40S GPU using the Adam optimizer (Diederik P. Kingma and Ba 2017). 90% percent of the dataset was used for training, and the remaining 10% for validation (testing).

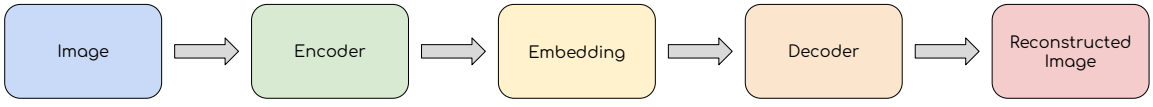

Figure 5.7: An illustration of the pipeline with a convolutional autoencoder (CAE)

5.2.2 Methods for graph-based representations

The methods applied to the graph-based representations of the cell data include: FEATHER, graph autoencoder, variational graph autoencoder and the DGLC framework.

FEATHER As described in Section 3.9, FEATHER is a method that uses characteristic functions to describe the distribution of features in the neighborhood of a node, using random-walk weights. These node level features are pooled by mean pooling to create graph level embeddings. FEATHER was considered as an example of a traditional method for learning embeddings on graphs, similar to the popular graph2vec (Narayanan et al. 2017). Compared to graph2vec, FEATHER is better suited for attributed graphs, such as those used in this thesis.

The implementation used was the one provided by the KarateClub library[I], with default parameters (i.e., mean pooling), which produced embeddings of length 500. As briefly mentioned earlier, basic preprocessing was applied before feeding the graphs into FEATHER. This included simple checks for `NaN` and `Inf` values. Additionally, all node features were min-max scaled so that their values lie between 0 and 1.

Graph autoencoder As described in Section 3.6, graph autoencoders (GAEs) are unsupervised neural network architectures used to learn node embeddings from graph-structured data. Graph-level embeddings are constructed by averaging the node-level embeddings, i.e. by mean pooling.

The GAE architecture consists of two main components: an encoder and a decoder. In this setting, the encoder was composed of two graph convolutional layers (`GCNConv`), aiming to learn how to map the nodes into latent representations. The decoder reconstructs the adjacency matrix of the graph from the embeddings and is implemented as an inner product. The implementation used was the one provided by PyTorch-Geometric[II].

Before training the GAE, self-loops were added to every node in all graphs. Moreover, graphs without edges were dropped. All node features were min-max scaled to fall between 0 and 1. Training was done on a single L40S GPU using the Adam optimizer. 90% percent of the dataset was used for training, and the remaining 10% for validation (testing). The validation set was used to select the optimal length of the graph-level embeddings.

Variational graph autoencoder The variational graph autoencoder (VGAE) extends the GAE by introducing a probabilistic framework, based on the VAE architecture, as described in Section 3.7. As opposed to deterministic node embeddings, the VGAE model learns a distribution over latent variables for each node. Similarly, graph-level embeddings are obtained by averaging

[I] https://karateclub.readthedocs.io/

[II] https://github.com/pyg-team/pytorch_geometric/blob/master/examples/autoencoder.py

node-level embeddings, i.e. by mean pooling. The VGAE implementation used was the one provided by PyTorch-Geometric[III].

In a similar manner as with the GAE, self-loops were added, graphs without edges were excluded and node features were min-max scaled. Training was done on a single L40S GPU using the Adam optimizer. 90% percent of the dataset was used for training, and the remaining 10% for validation (testing). The validation set was used to select the optimal length of the graph-level embeddings.

DGLC The DGLC framework is an end-to-end framework to learn graph-level embeddings and perform clustering simultaneously, as outlined in Section 3.7. The implementation was taken from the original GitHub repository[IV] provided by the authors of the paper.

Training was performed on a single L40S GPU using the Adam optimizer. 90% of the dataset was used for training, and the remaining 10% for validation (testing). It is important to note that the number of clusters to discover was one of the hyperparameters provided to the model. Other parameters included the number of layers in the GNN, the dimension of the hidden layer, and the dimension of the cluster projector (i.e., the embedding size). Variants with different numbers of clusters were trained, and the validation set was used to determine the optimal values for the hidden layer and cluster projector dimensions.

5.3 Clustering algorithms

This section describes the clustering algorithms used to cluster the embeddings produced by the methods described in the previous section for *image-based* and *graph-based* representations, respectively. The clustering algorithms used include: *k-means*, *DBSCAN* and *Spectral clustering*.

k-means As presented in Section 3.10.1, *k-means* is one of the most popular unsupervised clustering algorithms, which partitions data into k clusters by minimizing within-cluster variance (i.e., the sum of squared Euclidean distances). An important disadvantage of this algorithm is that the number of clusters k needs to be specified in advance. To mitigate this issue and to guide the choice of k, both *clustering coefficients* and the *elbow method* (Thorndike 1953) are used.

As explained in Section 3.11, the Silhouette, Calinski-Harabasz (CH) and Davies-Bouldin (DB) scores are suitable for evaluating the clustering performance. By analyzing the trends of these three scores for different values of k, an appropriate k can be chosen. On the other hand, the elbow method considers the *inertia*, i.e. the sum of squared distances of samples to their closest cluster center, and can be used as a heuristic to determine the number of clusters k. The method consists of plotting the inertia as a function of the number of clusters k, and choosing the value of

[III] https://github.com/pyg-team/pytorch_geometric/blob/master/examples/autoencoder.py
[IV] https://github.com/JinyuCai95/DGLC

k where the curve inflects (i.e. the *elbow point*) as optimal. However, this method is considered subjective and unreliable (Ketchen and Shook 1996; Schubert 2023), especially in settings where a clear elbow point can not be identified.

DBSCAN Section 3.10.2 provides a detailed description of DBSCAN, one of the most well-known and widely-used clustering algorithms. Unlike k-means, DBSCAN is a density-based algorithm that groups together points in close proximity (i.e., in high-density regions), while marking as outliers points found alone or with few neighbors (i.e., in low-density regions). Additionally, DBSCAN does not require the number of clusters k to be specified a priori, which can be seen as an advantage. However, there are two other important parameters that influence the clusters discovered by the algorithm: ε and `min_samples`. The ε parameter refers to the maximum distance between two points for one to be considered a neighbor of the other, while `min_samples` refers to the minimum number of points in a neighborhood of a point required to form a cluster (i.e., the minimum number of points required to form a dense region).

Typically, the `min_samples` parameter is set to be at least $D + 1$, where D represents the dimensionality of the data points in the dataset. (Ester et al. 1996) propose to select `min_samples` as 4 for 2-dimensional datasets, while (Sander et al. 1998) suggest a generalization of $2 \cdot D$ for D dimensions. However, (Schubert et al. 2017) state that for larger, higher-dimensional datasets or datasets with many duplicates and noise, increasing `min_samples` can improve performance.

The value of ε parameter is often more challenging to choose. If ε is chosen to be too small, a large portion of the dataset will not be clustered (i.e., it will be assigned as outliers). On the other hand, if ε is set too large, the clusters will merge, resulting in majority of the data points belonging to the same cluster. (Ester et al. 1996) propose a heuristic for choosing ε: the *k-dist graph*. First, the distances to the k-nearest neighbor of each point in the dataset are computed, where k is set to `min_samples` $- 1$ as per (Sander et al. 1998). Next, the distances are ordered in descending order and shown on a graph. The distance where the graph inflects (i.e., the *elbow*, *knee* or *valley point*) is considered a good choice for ε.

It is important to note that, due to the nature and size of the dataset at hand, we also experimented with values of ε and `min_samples` beyond the outlined recommendations.

Spectral clustering Spectral clustering requires specifying the number of clusters k in advance, which poses a challenge as in many other clustering approaches. Similarly as in k-means, clustering coefficients can help. However, there exists a specific heuristic to Spectral clustering, which is the *eigengap heuristic*. Let $\lambda_1 \leq \lambda_2 \leq \cdots \leq \lambda_n$ be the eigenvalues of the Laplacian matrix $\mathbf{L}$. The goal is to choose k such that $\lambda_1, \lambda_2, \ldots \lambda_k$ are very small, but λ_{k+1} is relatively large, as described in (Luxburg 2007).

It should be noted that the algorithm was used with a fully connected similarity graph constructed using a Gaussian, (i.e. radial basis function) kernel as described in Section 3.10.3.

5.4 Evaluation

The main challenge in this thesis is *evaluation*. The goal is to identify clusters of mobile cells based on their performance characteristics. Specifically, cells that share common spatial performance patterns should be grouped together. However, the dataset does not contain ground truth labels linking mobile cell data to specific spatial patterns or radio network issues. This makes evaluation particularly difficult, as no standard metric (e.g., accuracy, or F1 score) can be applied to measure the results obtained by the methods described earlier. Moreover, defining a domain-specific quantitative metric in an unsupervised setting is challenging. Measuring an absolute distance or similarity between data from two mobile cells is not straightforward, since similarity can be based on different factors. One possibility could be to use a pretrained foundation model trained on large amounts of mobile cell performance data, both of which are not available.

Therefore, a two-step qualitative evaluation procedure is proposed. First, an evaluation pipeline consisting of various visualizations and clustering coefficients was applied. Secondly, examples from the clusters were sent to network engineers at Swisscom for interpretation. The evaluation relied primarily on the first step, as the second step was slow and time-consuming. Expert input was used mainly to validate claims and observations from the initial analysis.

The evaluation pipeline includes manual inspection, clustering coefficients, as well as projections of the clusters in a two-dimensional space with PCA, t-SNE and UMAP, radial sector maps and custom cluster statistics. Each of the components is described in detail below.

Manual inspection The first step of the evaluation pipeline involves visualizing a sample of cells from each cluster.

Clustering coefficients The Silhouette, Calinski-Harabasz, and Davies-Bouldin scores are used to evaluate clustering performance.

PCA, t-SNE and UMAP visualizations PCA (Jolliffe 2005) is a linear dimensionality reduction technique effective for discovering global patterns, while t-SNE (Maaten and G. Hinton 2008) and UMAP (McInnes, Healy, and Melville 2020) are nonlinear techniques that preserve both global and local structures (e.g., clusters, patterns, and other relationships between nearby points). Visualizations based on these three methods are used to analyze the shape of the clusters in the projected two-dimensional space. Data points are colored according to their cluster membership. The plots can be also used to compare the results of different clustering approaches.

Adjusted Rand Index (ARI) and Adjusted Mutual Information (AMI) scores The ARI and AMI scores, as described in Sections 3.11.4 and 3.11.5 respectively, are used to compare the results obtained by different clustering approaches (k-means, DBSCAN and Spectral clustering).

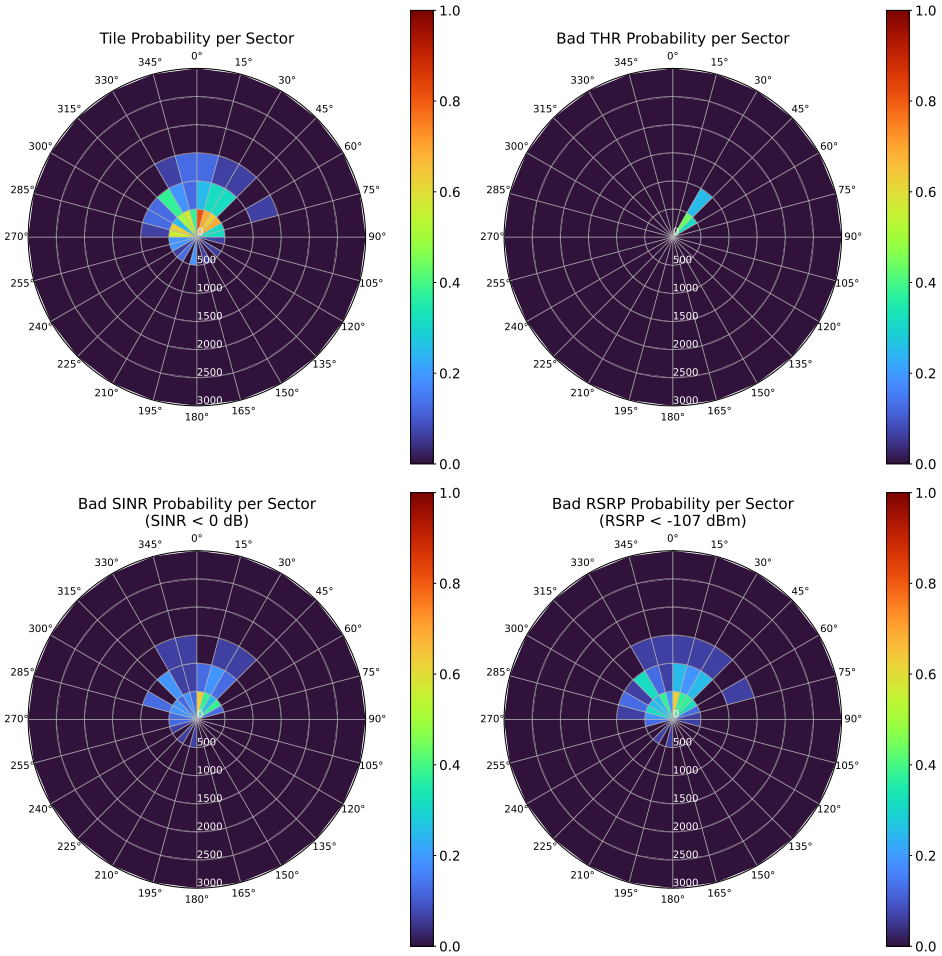

Figure 5.8: Example of a radial sector map

They are convenient because they can quantify the agreement between two independent label assignments on the same dataset in the absence of ground truth labels.

Radial sector maps Radial sector maps are polar heatmaps used to characterize mobile cells found in each cluster. The area shown on the maps is divided in sectors, defined by angle and distance from the cell site. The first map shows the probability that a cell has a tile in each sector. The remaining three maps display the probabilities of cells having bad tiles based on different performance metrics: throughput dominance, SINR and RSRP. An example is shown in Figure 5.8.

Custom metrics The last step of the evaluation pipeline consists of visualizing custom metrics: average SINR, RSRP, cell range, and fraction of red tiles.

6 Results

This section presents the results obtained from the methods described in Section 5. As previously discussed, due to the absence of ground truth labels, an evaluation pipeline based on multiple visualizations and scores was proposed to assess and compare the performance of different approaches.

Specifically, the results are given for the two different types of data representations, i.e., *image-based* and *graph-based* representations, along with the respective methods applied to each. First, the choice of parameters for the three clustering algorithms used (k-means, DBSCAN, and Spectral clustering) is briefly explained. Then, visualization methods using PCA, t-SNE, and UMAP are applied to display the clusters obtained by each clustering algorithm in the projected 2D space. The Adjusted Rand Index (ARI) and Adjusted Mutual Information (AMI) compare the results across the algorithms for each representation method. Furthermore, the discovered spatial patterns are presented using radial sector maps. Finally, where applicable, expert feedback is provided by radio network engineers at Swisscom, validating some of the results and linking them to potential issues.

Table 6.1 summarizes the parameters used for each clustering algorithm by method, i.e., k for k-means and Spectral clustering, and ε and `min_samples` for DBSCAN. The detailed process for selecting these parameters is explained in Section 5.3, and an example of the workflow is detailed in Section 6.1.1 for the embeddings obtained from the pretrained CNN. For each of the subsequent methods, the parameters, together with the resulting relevant information such as the

Table 6.1: Clustering parameters by embedding-learning method

Method	**k-means**	**DBSCAN**	**Spectral clustering**
Pretrained CNN	$k = 3$	$\varepsilon = 0.5$, `min_samples` $= 10$	$k = 30$
Single-layer CAE	$k = 30$	$\varepsilon = 17$, `min_samples` $= 10$	$k = 30$
Multi-layer CAE	$k = 30$	`NA`	$k = 30$
FEATHER	$k = 15$	$\varepsilon = 0.2$, `min_samples` $= 20$	$k = 30$
VGAE	$k = 6$	$\varepsilon = 0.45$, `min_samples` $= 20$	$k = 6$
DGLC	$k = 10,\ 20,\ 30$	`NA`	$k = 10,\ 20,\ 30$

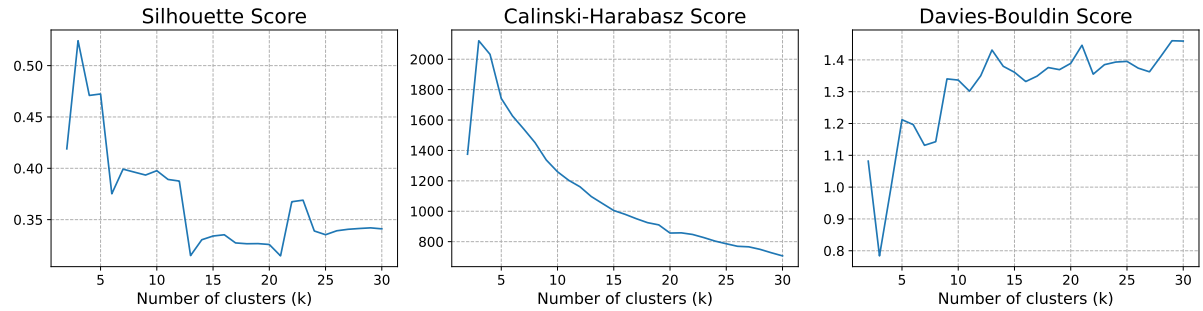

Figure 6.1: Silhouette, Calinski-Harabasz and Davies-Bouldin scores for different values of k of the k-means algorithm (Pretrained CNN)

number of clusters, number of outliers, and clustering coefficients (Silhouette, Calinski–Harabasz, and Davies-Bouldin scores), are briefly reported. Although heuristics such as the elbow method (k-means), k-dist graph (DBSCAN), and eigengap heuristic (Spectral clustering) exist for selecting optimal parameters, they serve merely as guidelines and can be subjective or unreliable (Ketchen and Shook 1996; Schubert 2023). That is, sometimes the choice of the parameters is dataset dependent and may rely on expert opinion or manual parameter search.

6.1 Image-based representations

The results from the methods applied to the image-based representation of the mobile cell data are presented below, starting with the pretrained CNN, followed by the convolutional autoencoder architecture for single-layer and multi-layer KPI. For the pretrained CNN, the process of selecting the optimal parameters for the clustering algorithms is described in detail. For the convolutional autoencoder, the descriptions are more concise.

6.1.1 Pretrained CNN

As described in Section 5, k-means, DBSCAN, and Spectral clustering were applied to the embeddings obtained from the pretrained CNN combined with PCA.

k-means Figure 6.1 shows the Silhouette, Calinski-Harabasz (CH) and Davies-Bouldin (DB) scores for different values of k of the k-means algorithm. As described in Section 3.11, higher values of the Silhouette and CH scores and lower values of the DB score indicate better clustering performance. From the plots, we observe that as k increases, the Silhouette and CH scores decrease, while the DB score increases, which suggests that smaller k produce better clusters. The highest Silhouette and CH scores, and the lowest DB score is attained for $k = 3$. Furthermore, the left panel Figure 6.2 shows the elbow method. While a clear inflection point cannot be seen, the rate of decrease of the *inertia* (Section 5.3) slows down between $k = 3$ and $k = 8$. Based on

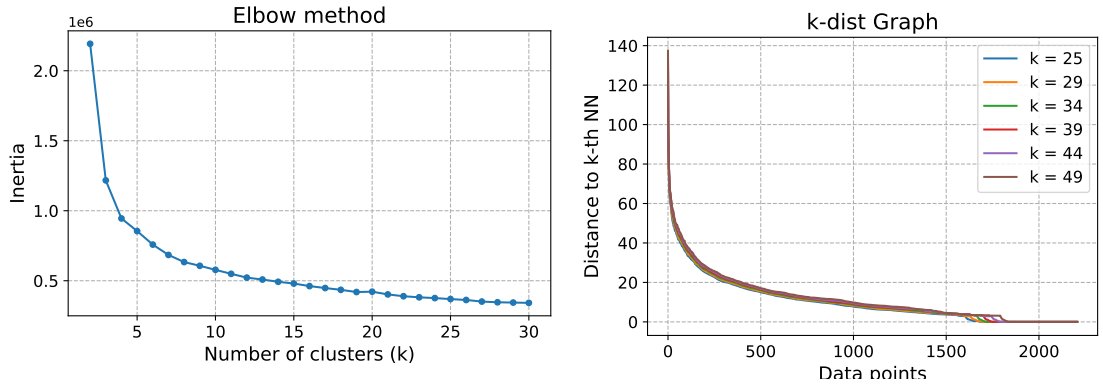

Figure 6.2: Elbow method for k-means (left) and k-dist graph for DBSCAN (right) (Pretrained CNN)

the described observations, the number of clusters was set to 3. When $k = 3$, the Silhouette, CH and DB are 0.52, 2121.65 and 0.78, respectively. resulting. Refer to Section 3.11 for more details on these scores.

DBSCAN Figure 6.2 (right) shows the k-dist graph for selecting an optimal value of ε for DBSCAN for different values of k (`min_samples` - 1). As described in Section 5.3, `min_samples` should be set to at least $D + 1$ or $2 \cdot D$, where D is the dimensionality of the embeddings. In this setting, $D = 25$, so `min_samples` should be at least 26. The plot does not reveal a clear inflection point for any of the curves. An interesting observation is the steep, stair-like drop that occurs around $\varepsilon = 3$ in all curves. Additionally, the relatively large distances to the k-th nearest neighbor, as shown on the y-axis, suggest that the space is sparse.

Figure 6.3 explores values of ε around 3 by showing the number of clusters and the fraction of outliers. Intuitively, ε should not be chosen too large as it will cause different clusters to merge together. Based on these observations, ε can be set to 2.8 and `min_samples` to 26, resulting in 8 clusters and 71% of the data labeled as outliers. However, it is important to keep in mind that the heuristics are just guidelines. They can provide some intuition on how to choose the parameters for ε and `min_samples`, but their recommendations should not be taken as definitive. Observing the visualizations produced by PCA, t-SNE and UMAP and the radial sector maps, setting `min_samples` to smaller values than recommended can sometimes help identify clusters with fewer points. Since the dataset is relatively small and we are looking for distinct patterns or anomalies, detecting groups of outliers can be more valuable. Hence, manually tuning the parameters can be a reasonable approach. After some experimentation, `min_samples` was set to 10, while ε was set to 0.5 for DBSCAN, resulting in 18 clusters and 66.92 % of outliers.

Spectral clustering The same reasoning applies to Spectral clustering. The Silhouette and DB scores shown in Figure 6.4 suggest that larger values of k produce better clustering results.

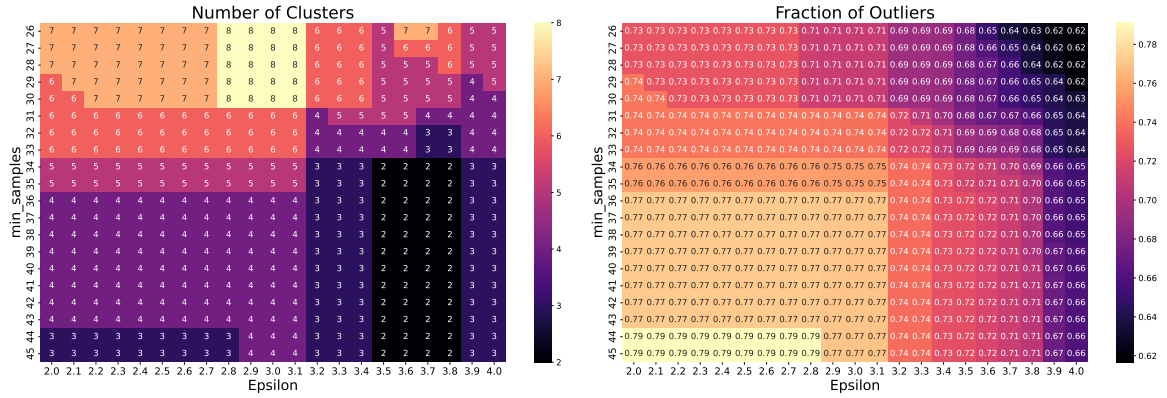

Figure 6.3: Number of clusters and fraction of outliers as functions of ε and `min_samples` (Pretrained CNN)

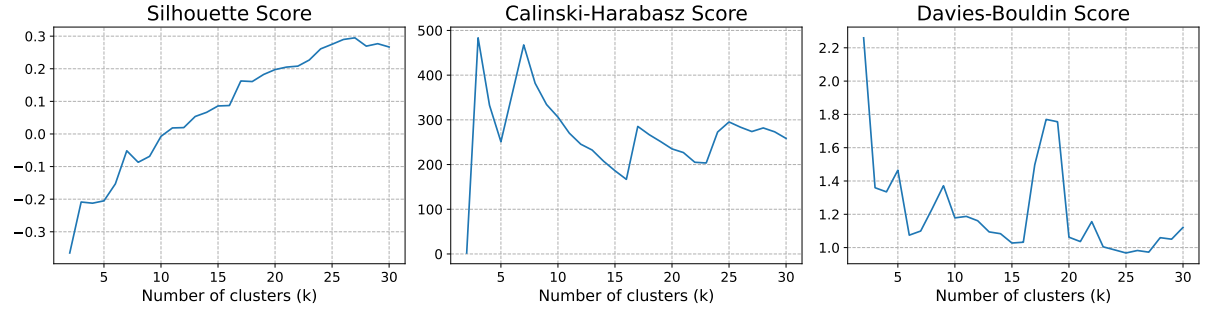

Figure 6.4: Silhouette, Calinski-Harabasz and Davies-Bouldin scores for different values of k of the Spectral clustering algorithm (Pretrained CNN)

Additionally, the eigengap heuristic shown in Figure 6.5 identifies an absurd value of $k = 641$ clusters for a dataset of 2207 points. Applying the same reasoning as DBSCAN, the number of clusters k was set to 30.

PCA, t-SNE and UMAP visualizations Figure 6.6 shows PCA, t-SNE and UMAP projections of the clusters produced by k-means, DBSCAN and Spectral clustering. Black data points in the DBSCAN-related plots represent outliers. DBSCAN and Spectral clustering identify many small, well-separated groups, while k-means produces larger clusters. Furthermore, t-SNE and UMAP reveal clearer separation between clusters than PCA.

AMI and ARI scores Tables 6.2 and 6.3 give the ARI and AMI scores between the clustering assignments given by k-means, DBSCAN and Spectral clustering. Both metrics show low similarity between k-means and the other methods, and moderate similarity between DBSCAN and Spectral clustering.

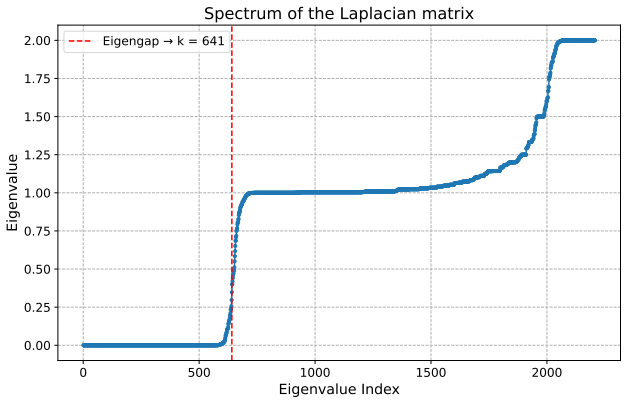

Figure 6.5: Eigengap heuristic for Spectral clustering (Pretrained CNN)

Table 6.2: Adjusted Rand Index (ARI) between clustering methods (Pretrained CNN)

	k-means	DBSCAN	Spectral
k-means	1.00	0.09	0.15
DBSCAN	0.09	1.00	0.19
Spectral	0.15	0.19	1.00

Table 6.3: Adjusted Mutual Information (AMI) between clustering methods (Pretrained CNN)

	k-means	DBSCAN	Spectral
k-means	1.00	0.30	0.40
DBSCAN	0.30	1.00	0.57
Spectral	0.40	0.57	1.00

Observations Figures 6.7, 6.8, and 6.9 show some of the spatial patterns using the radial sector maps described in Section 5.4. For readability, only a few maps are shown. All radial sector maps are shown in Appendix A.1.

Figure 6.7 shows the maps for all three clusters discovered by k-means. No clear spatial pattern emerged, the clusters appear mixed and the probabilities per sector remain low. One can hypothesize that since k is small, k-means likely merges distinct patterns together.

Figure 6.8 shows a few of the patterns found by the DBSCAN algorithm. The clusters correspond to different directions and distances of the red tiles relative to the cell site. Figures 6.10, 6.11, and 6.12 show a few examples of mobile cells from Clusters 9, 14 and 17. While DBSCAN produces clusters where elements share problematic (red) tiles at the same physical location, it can also generate multiple clusters with similar radial sector maps. Sometimes the radial sector maps show low probabilities across all regions, suggesting that several different patterns are grouped into the same cluster. In most clusters, the mobile cells contain only one or a few red tiles. Mobile cells with more red tiles and less defined structures were considered outliers.

Figure 6.9 shows a few clusters identified by Spectral clustering. This algorithm does not account for outliers and finds patterns similar to those detected by DBSCAN. Specifically, these patterns include different directions and distances of the red tiles relative to the cell site. Sometimes, it detects very fine-grained differences; for example, Clusters 12 and 13 in Figure 6.9 both consist of

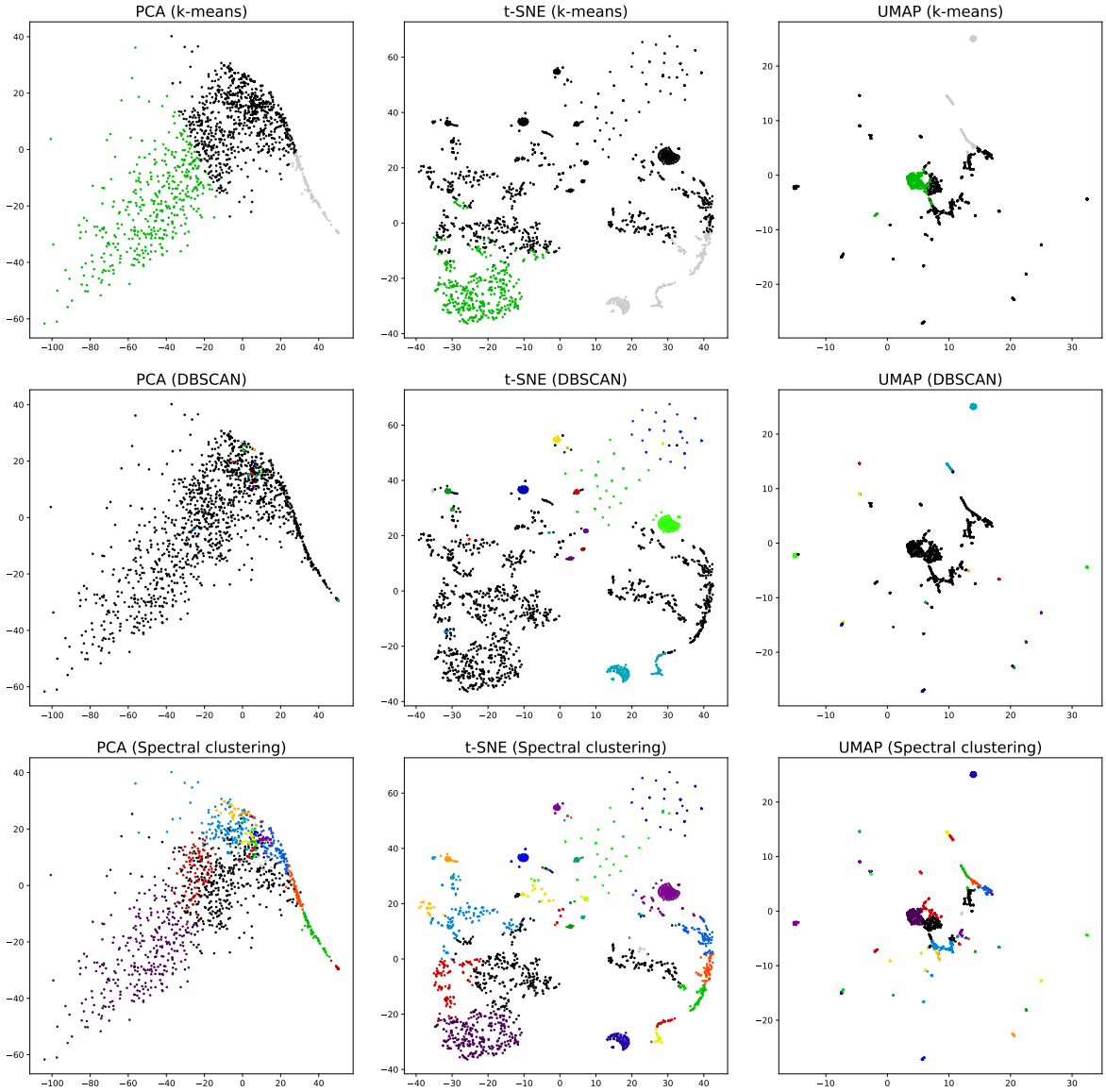

Figure 6.6: Clustering results on pretrained CNN embeddings visualized with PCA (left), t-SNE (center), and UMAP (right), using k-means (top row), DBSCAN (middle row), and Spectral clustering (bottom row).

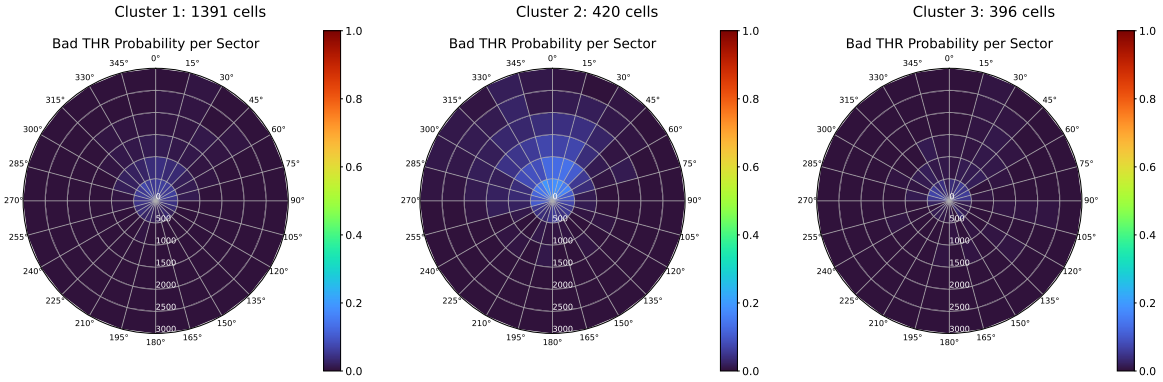

Figure 6.7: Spatial performance patterns discovered using k-means, shown as radial sector maps of bad throughput probability per sector (Pretrained CNN)

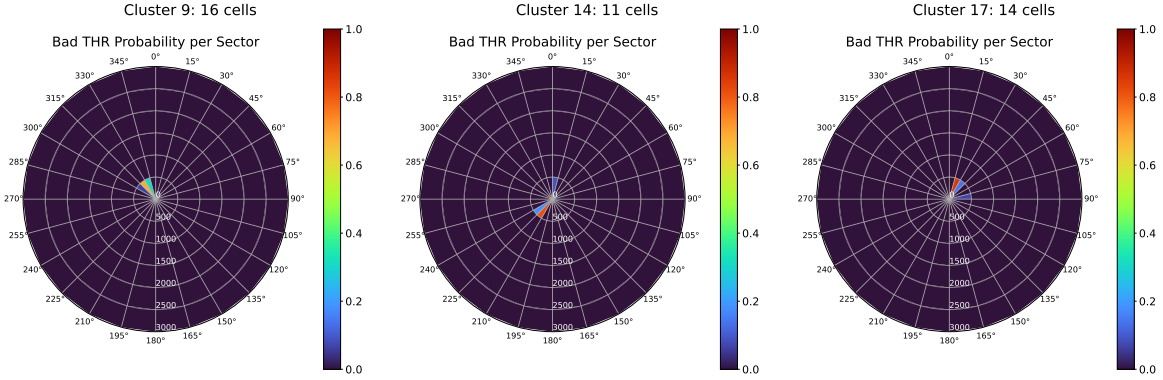

Figure 6.8: Spatial performance patterns discovered using DBSCAN, shown as radial sector maps of bad throughput probability per sector (Pretrained CNN)

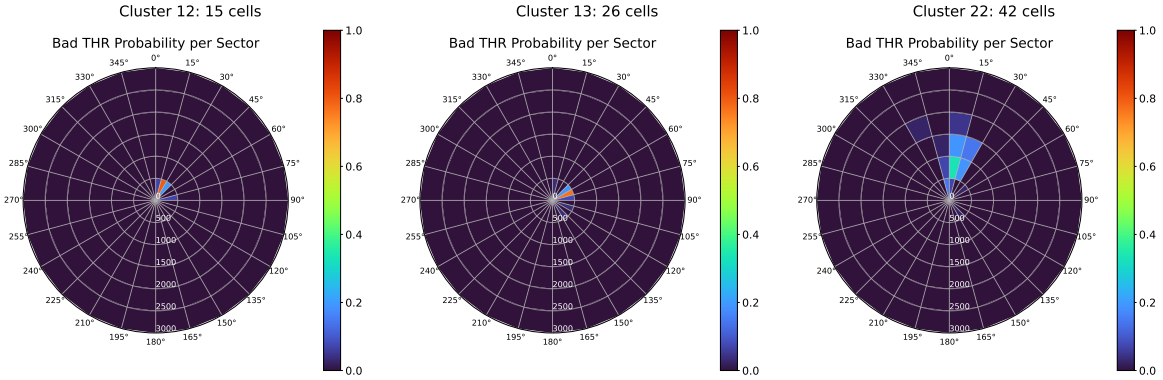

Figure 6.9: Spatial performance patterns discovered using Spectral clustering, shown as radial sector maps of bad throughput probability per sector (Pretrained CNN)

Figure 6.10: Example mobile cells from Cluster 9 discovered by DBSCAN (Pretrained CNN)

Figure 6.11: Example mobile cells from Cluster 14 discovered by DBSCAN (Pretrained CNN)

cells with red tiles at the front of the beam on the right side. However, for Cluster 12 the azimuth ranges between 15° and 30°, while for Cluster 13 it ranges between 60° and 75°. Additionally, compared to DBSCAN, Spectral clustering detects directional patterns where the red tiles are further away from the site, such as Cluster 22 shown in Figure 6.9 (as well as clusters 23 and 26 shown in the Appendix A.1).

Interpretations Radio network engineers inspected some of the clusters discovered by DBSCAN and Spectral clustering. The distance and direction of red tiles becomes interesting when they are located outside the coverage area of the cell (given by its range, beam direction and sector angle), as is the case with Cluster 14 discovered by DBSCAN (Figure 6.11). This may indicate the presence of an object (e.g., a building) reflecting the signal in an unintended direction, but more importantly, it can suggest issues with one of the neighboring cells. Specifically, upon examining examples from Cluster 14, engineers identified problems with the tilt and power configurations of

Figure 6.12: Example mobile cells from Cluster 17 discovered by DBSCAN (Pretrained CNN)

neighboring cells. These misconfigurations caused the neighboring cells to fail to properly serve the area represented by the red tiles, which was instead being served by the cell of interest.

6.1.2 Convolutional Autoencoder

As described in Section 5.2.1, two variants of the CAE architecture were trained. The first variant (Single-layer KPI) is trained on throughput dominance images, while the second (Multi-layer KPI) is trained 9-channel images, i.e., stacked throughput dominance, SINR, and RSRP layers. Using the validation set, the hidden layer size (i.e., embedding size) was set to 64 for both the first and the second variant. Further details on hyperparameter tuning are provided in Appendix A.2.1.

Single-layer KPI CAE

The parameters for each clustering algorithm were selected following the procedure described in Sections 5.3 and 6.1.1. For k-means, the number of clusters was set to 30, resulting in Silhouette, DB, and CH scores of 0.09, 54.1, and 1.79, respectively. Unlike in Section 6.1.1, the Silhouette, CH, and DB indices did not agree on the optimal value of k. Specifically, the Silhouette score increased with larger k, while the CH and DB scores decreased. Similarly, the elbow method did not reveal a clear inflection point. For DBSCAN, ε and `min_samples` were set to 17 and 10, respectively, resulting in 9 clusters and 88.45% outliers. Tuning these parameters was challenging. According to the heuristics, if `min_samples` was set to at least $D + 1$ (i.e., 65 in this case), for ε smaller than 39 the algorithm produced zero clusters. Increasing ε resulted in one to two clusters, which was undesirable. Therefore, the parameters were manually tuned, as described earlier, by examining visualizations from PCA, t-SNE, and UMAP alongside the patterns revealed by the radial sector maps. For Spectral clustering, the eigengap heuristic suggested choosing k as 2030, which is an absurdly large value. However, as previously discussed, it was set to 30, yielding Silhouette, CH, and DB scores of -0.05, 35.55, and 2.07, respectively.

PCA, t-SNE and UMAP visualizations Figure 6.13 shows PCA, t-SNE and UMAP visualizations of the cluster assignments given by the 3 algorithms. Data points colored black in the DBSCAN-related plots represent outliers.

AMI and ARI scores Tables 6.4 and 6.5 give the ARI and AMI scores between the clustering assignments given by k-means, DBSCAN and Spectral clustering, respectively.

Observations Figures 6.14, 6.15, and 6.16 show some of the discovered spatial patterns using the radial sector maps described in Section 5.4. For readability, only a few maps are shown per clustering algorithm.

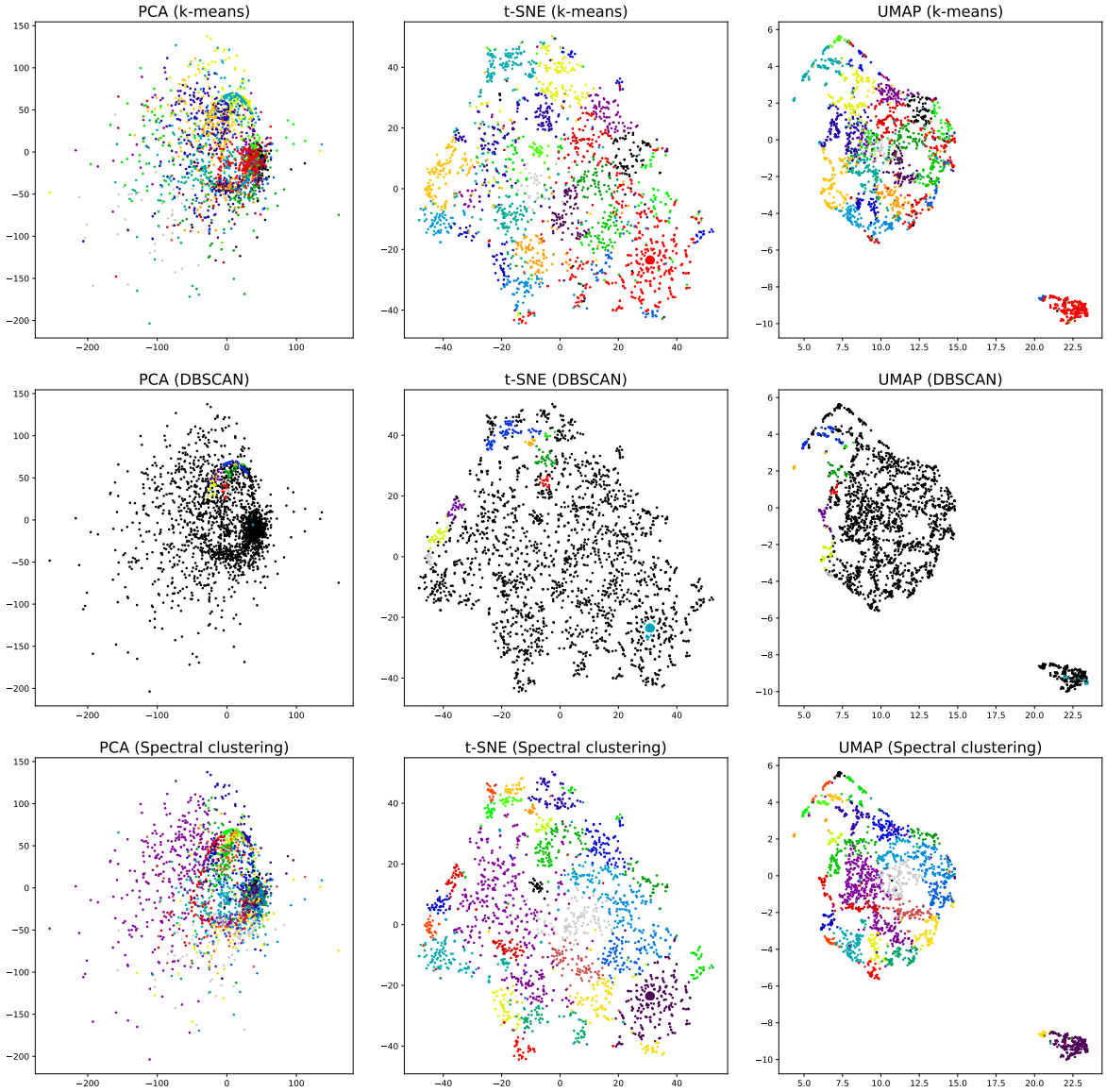

Figure 6.13: Clustering results on Single-layer KPI CAE embeddings visualized with PCA (left), t-SNE (center), and UMAP (right), using k-means (top row), DBSCAN (middle row), and Spectral clustering (bottom row).

Table 6.4: Adjusted Rand Index (ARI) between clustering methods (Single-layer KPI CAE)

	k-means	DBSCAN	Spectral
k-means	1.00	0.00	0.36
DBSCAN	0.00	1.00	0.02
Spectral	0.36	0.02	1.00

Table 6.5: Adjusted Mutual Information (AMI) between clustering methods (Single-layer KPI CAE)

	k-means	DBSCAN	Spectral
k-means	1.00	0.13	0.57
DBSCAN	0.13	1.00	0.21
Spectral	0.57	0.21	1.00

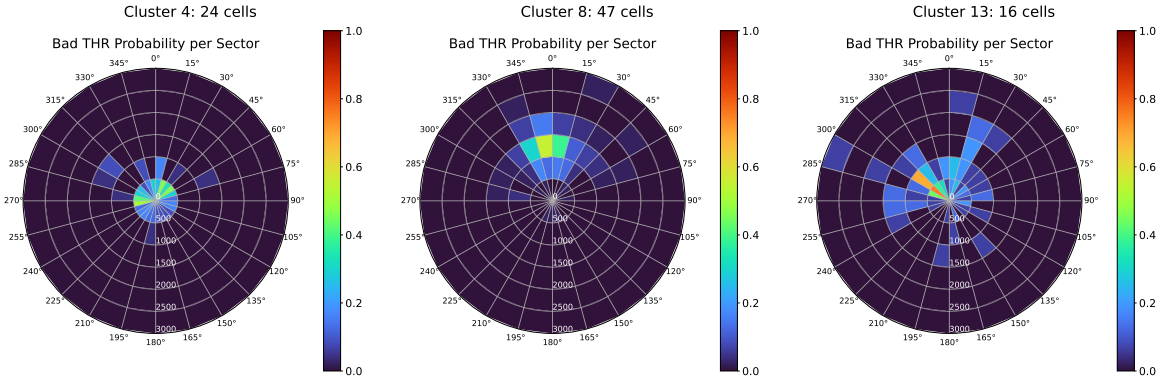

Figure 6.14: Spatial performance patterns discovered using k-means, shown as radial sector maps of bad throughput probability per sector (Single-layer KPI CAE)

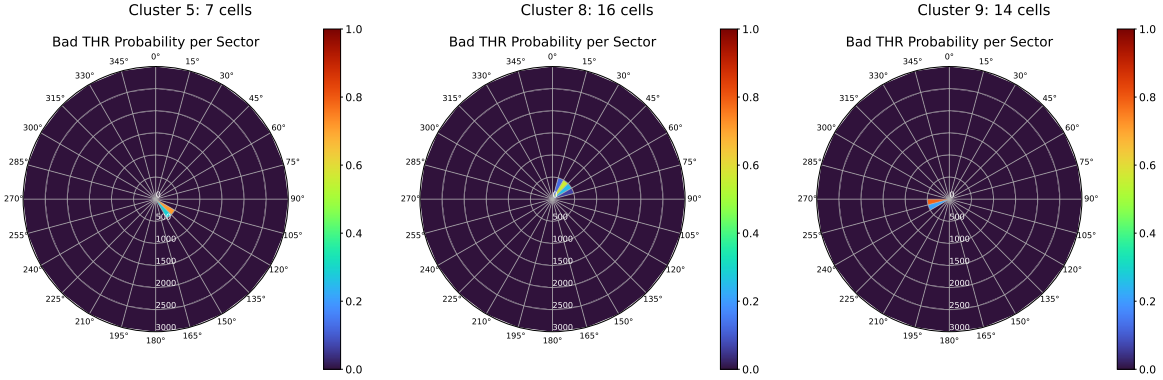

Figure 6.15: Spatial performance patterns discovered using DBSCAN, shown as radial sector maps of bad throughput probability per sector (Single-layer KPI CAE)

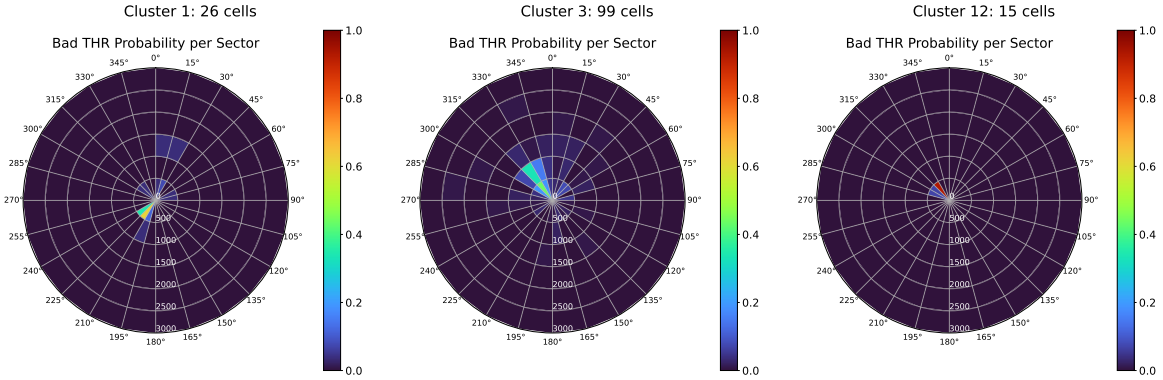

Figure 6.16: Spatial performance patterns discovered using Spectral clustering, shown as radial sector maps of bad throughput probability per sector (Single-layer KPI CAE)

Figure 6.17: Example mobile cells from Cluster 4 discovered by k-means (Single-layer KPI CAE)

Figure 6.18: Example mobile cells from Cluster 8 discovered by k-means (Single-layer KPI CAE)

Figure 6.14 shows a few interesting performance patterns discovered by k-means (i.e., Clusters 4, 8 and 24). Similarly as before (i.e., Section 6.1.1), the majority of patterns correspond to different angular directions and distances of red tiles respective to the cell site. However, the mobile cells in Cluster 4 are characterized by a lot of red tiles positioned in all directions around the site, as shown in the selected examples in Figure 6.17. Similarly, the mobile cells in Cluster 8 are characterized by one or few red tiles located in front of the beam of the antenna at distances between 1000 and 1500 meters, as shown in Figure 6.18. Finally, Cluster 13 corresponds to a pattern characterized by a high concentration of red tiles in front of the antenna beam, particularly at azimuth angles between 300 and 315 degrees (Figure 6.19).

Figures 6.15 and 6.16 show some of the patterns discovered by DBSCAN and Spectral clustering. Similarly as before, both algorithms identify patterns corresponding to different directions and distances of red tiles relative to the mobile cell site. DBSCAN produces only 9 clusters and a high percentage of outliers, failing to capture the full diversity of possible patterns. Spectral clustering, on the other hand, produces results that are similar to those of k-means. All three algorithms still produce mixed clusters, i.e., clusters with seemingly no pattern, as well as duplicate clusters, i.e., clusters with similar patterns.

Multi-layer KPI CAE

The Multi-layer KPI CAE is the first approach that considers all three layers of KPIs, i.e., throughput dominance, SINR and RSRP. In other words, this model was trained on images consisting of 9 channels, i.e., 3 channels per KPI metric.

Figure 6.19: Example mobile cells from Cluster 13 discovered by k-means (Single-layer KPI CAE)

As before, the first step of the analysis was to set the optimal parameters for each clustering algorithm. For k-means, the Silhouette and CH scores decreased as k increased, which suggests that lower values of k produce better clusters. However, the DB score also decreased for larger values of k indicating that higher values of k produce better clusters, which was contradictory. Furthermore, the elbow method showed no clear inflection point. Regarding DBSCAN, if `min_samples` was set to at least $D + 1$, the algorithm discovered zero to one clusters for different values of ε. Relaxing `min_samples` and experimenting with ε in some cases produced three to six clusters; however, this was accompanied by an astonishing number of outliers (above 95%). Similarly, the eigengap heuristic for Spectral clustering suggested an absurd number of 9956 clusters for a dataset of 10443 points.

Observations Unfortunately, the visualizations with PCA, t-SNE, and UMAP did not show clear cluster separations either. Figure 6.20 shows the projections of the data embeddings given by each of these methods. Similarly, the radial sector maps did not reveal specific patterns for the performance metrics, unlike in Sections 6.1.1 and 6.1.2. Consequently, only k-means and Spectral clustering were considered, as even manual tuning failed to produce meaningful results with DBSCAN due to the high number of outliers. In both approaches, k was set to 30. The only aspect the resulting clusters appeared to reflect was the shape and size of the area covered by the cell, indicating its range. This was validated by the first of the four radial sector maps described in Section 5.4, as well as by manual inspection of the cells assigned to each cluster. Examples are shown in Figure 6.20. The ARI and AMI scores between the cluster assignments produced by k-means and Spectral clustering were 0.2 and 0.47, respectively.

Interpretations According to the radio network engineers, the shape and size of the area covered by a mobile cell is not particularly useful on its own. It could become relevant when combined with information about the cell's location. For example, a large coverage area might be acceptable if the cell is located outside of an urban area. However, the same pattern inside a city could indicate a problem.

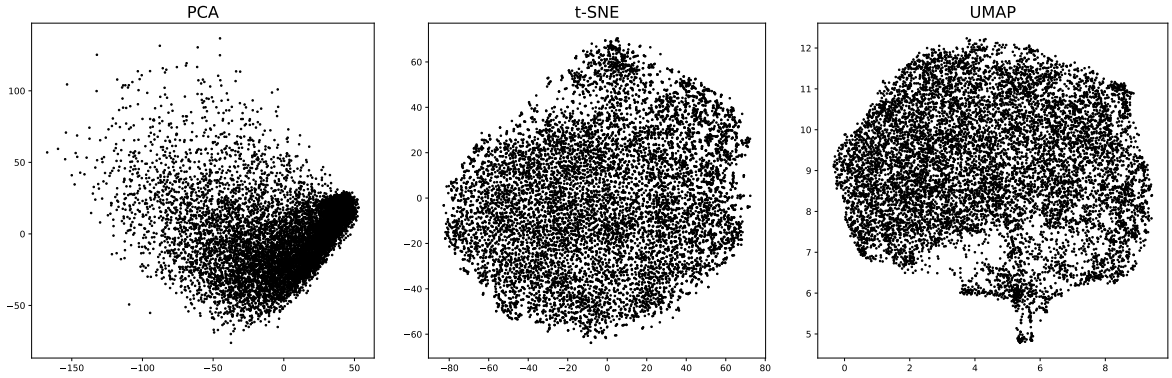

Figure 6.20: Multi-layer KPI CAE embeddings visualized with PCA (left), t-SNE (center), and UMAP (right)

6.2 Graph-based representations

The results from the methods applied to the graph-based representation of the mobile cell data are presented below, starting with FEATHER, followed by the graph autoencoder, variational graph autoencoder, and the DGLC framework. As with the image-based representations, k-means, DBSCAN, and Spectral clustering were applied to the graph-level embeddings produced by each of these approaches to generate clusters.

6.2.1 FEATHER

Similarly as Section 6.1.2, the heuristics for determining the optimal parameters of k-means, DBSCAN and Spectral clustering were not particularly useful. Furthermore, the radial sector maps showed no patterns to guide parameter selection, making the process even more challenging. Fortunately, the t-SNE visualization showed isolated blobs of data points, so the objective during parameter selection was to separate them as good as possible and analyze what they correspond to. Figure 6.22 shows PCA, t-SNE, and UMAP visualizations of the cluster assignments produced by the three algorithms.

k-means The Silhouette score decreased and the DB score increased with larger values of k, indicating that lower values produce better clusters. However, the CH score increased with k, suggesting the opposite. The elbow method showed no clear inflection point. Based on manual parameter tuning, aiming for a compromise between the clustering coefficients and taking into account the visualizations from PCA, t-SNE, and UMAP, k was set to 15. The resulting Silhouette, CH, and DB scores were 0.41, 18335.45, and 0.76, respectively.

DBSCAN Following the heuristics, setting `min_samples` to at least $D + 1$ or $2D$ resulted in 0 to 2 clusters for different values of ε, which was undesirable. Therefore, ε and `min_samples`

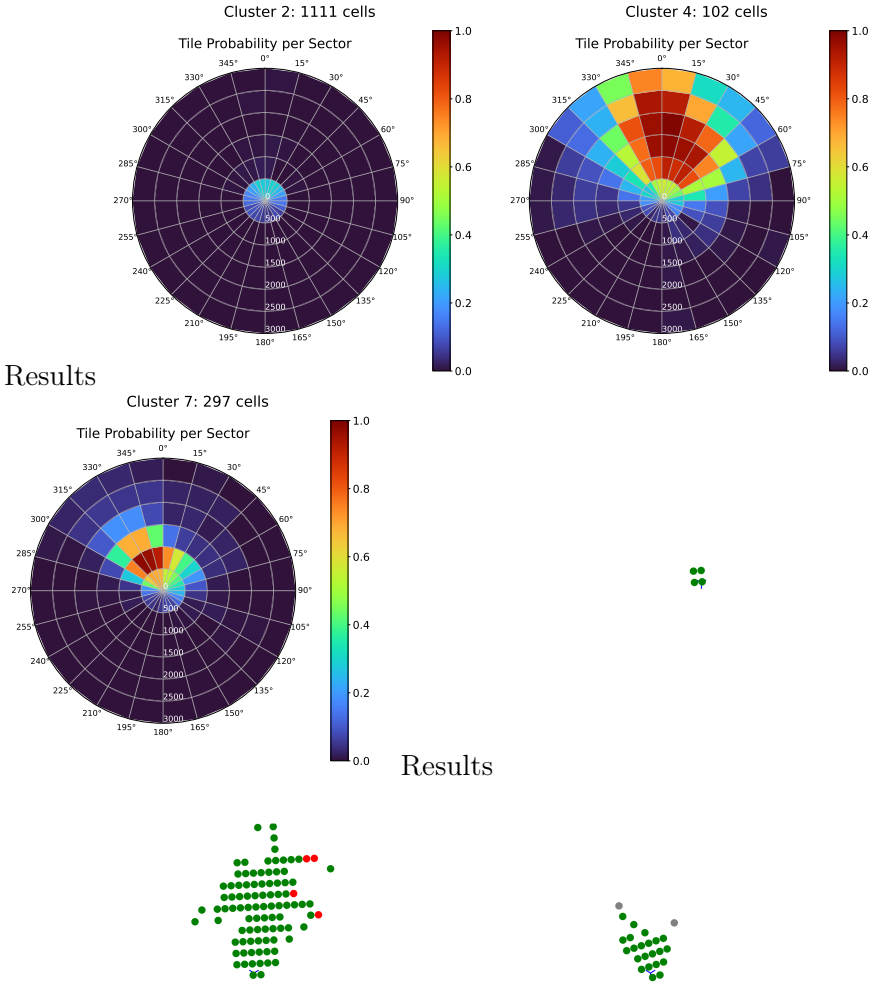

Figure 6.21: Spatial performance patterns discovered using k-means, shown as radial sector maps of tile probability per sector (Multi-layer KPI CAE) in the top row. The bottom row shows example mobile cells from each cluster, using the image corresponding to throughput dominance.

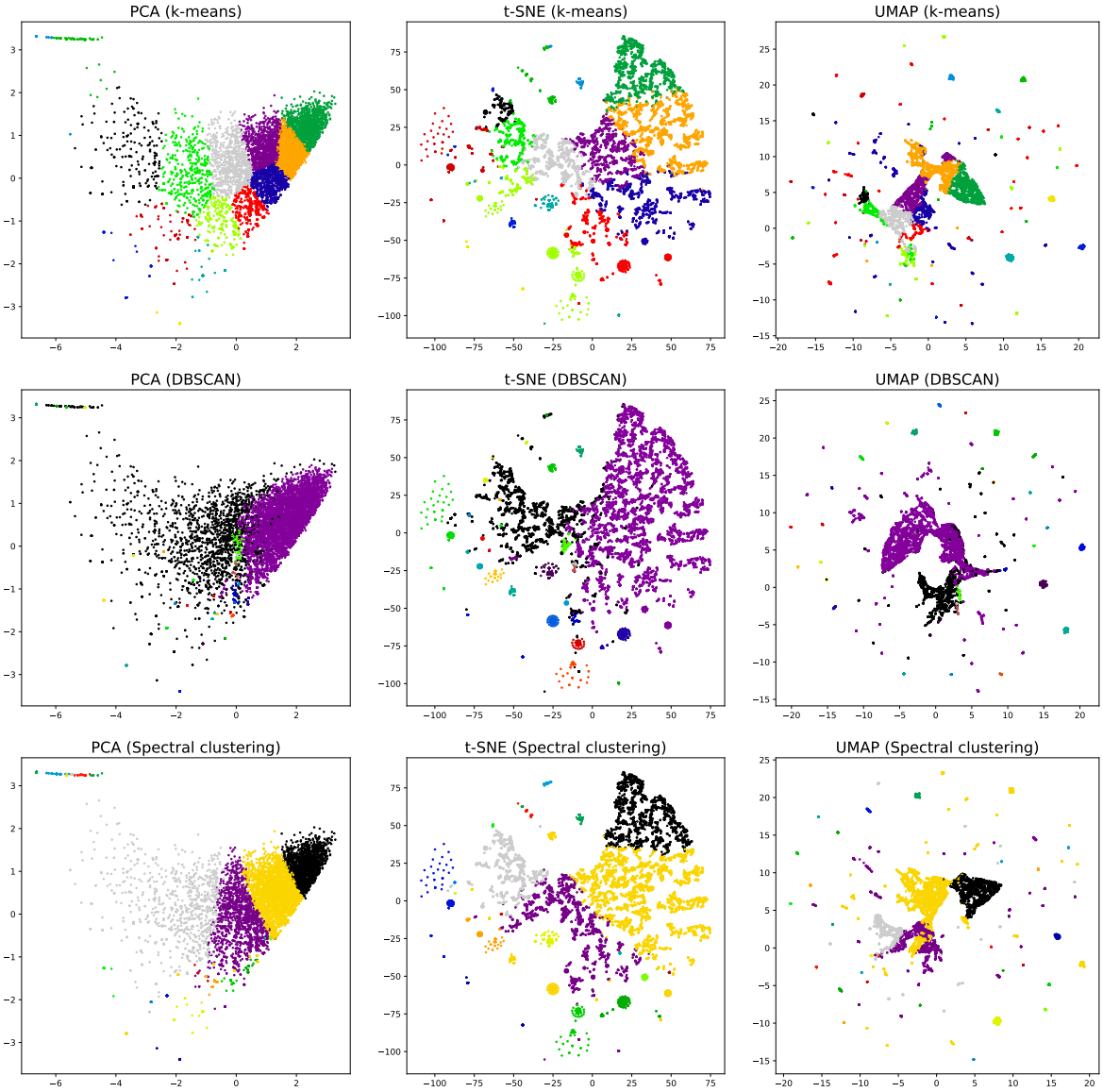

Figure 6.22: Clustering results on FEATHER embeddings visualized with PCA (left), t-SNE (center), and UMAP (right), using k-means (top row), DBSCAN (middle row), and Spectral clustering (bottom row).

were determined by examining the number of clusters produced and the fraction of outliers for each combination. Finally, based on manual tuning and observing the cluster assignments via the visualizations with PCA, t-SNE, and UMAP, ε was set to 0.2 and `min_samples` to 20. This resulted in 29 clusters and 19.32% outliers.

Spectral clustering The eigengap heuristic suggested setting k to 7, resulting in Silhouette, CH and DB scores of -0.35, 672.79 and 1.14 respectively. However, observing the cluster assignments via the visualizations with PCA, t-SNE and UMAP, k was set to 30, improving each of the Silhouette, CH and DB scores to 0.22, 3161.38, and 0.82, respectively.

Observations The clusters discovered by each of the algorithms mainly correspond to the size of the area covered by the cell, similarly as in Section 6.1.2, i.e., the number of tiles or nodes in the graphs. It is clear that the clusters do not reflect performance patterns, as was the case before in Sections 6.1.1 and 6.1.2 with varying locations of red throughput dominance, SINR, or RSRP tiles. Beyond grouping the graphs by the number of nodes, some clusters correspond to *isomorphic* graphs (i.e., graphs with the same structure). Figure 6.23 shows examples of clusters linked to specific graph structures discovered by DBSCAN.

6.2.2 (Variational) Graph Autoencoder

As described in Section 5.2.2, the embedding size was set to 64 for the GAE and 16 for the VGAE, using the validation set. Further details on hyperparameter tuning are provided in Appendices A.3.1 and A.4.1. As neither method produced interpretable results, only the results from the VGAE are described below (the outcome was similar for the GAE). Following the procedure in Sections 6 and 6.1.1, the first step was to select the clustering parameters. Figure 6.24 shows PCA, t-SNE, and UMAP visualizations of the resulting cluster assignments, while Tables 6.6 and 6.7 report the corresponding ARI and AMI scores.

k-means For $k = 6$, the Silhouette and CH scores attained their maximum, while the DB score reached its second lowest value. As the elbow method likewise displayed a subtle inflection point in the same range, k was set to 6. The resulting Silhouette, CH and DB scores were 0.22, 1486.69 and 1.51, respectively.

DBSCAN The number of clusters and the fraction of outliers for different combinations of ε and `min_samples` showed that for ε values larger than 1, all points were assigned to a single cluster (i.e., one cluster and 0% outliers), even when `min_samples` was as low as 3. For `min_samples` equal to $D + 1$ (17 in this setting), the k-dist graph didn't show a clear inflection point, however the rate of decrease significantly dropped between 0.7 and 1 for ε. Nevertheless, the corresponding range of ε and `min_samples` resulted in 1 cluster and 0 to 4% of outliers which was undesirable.

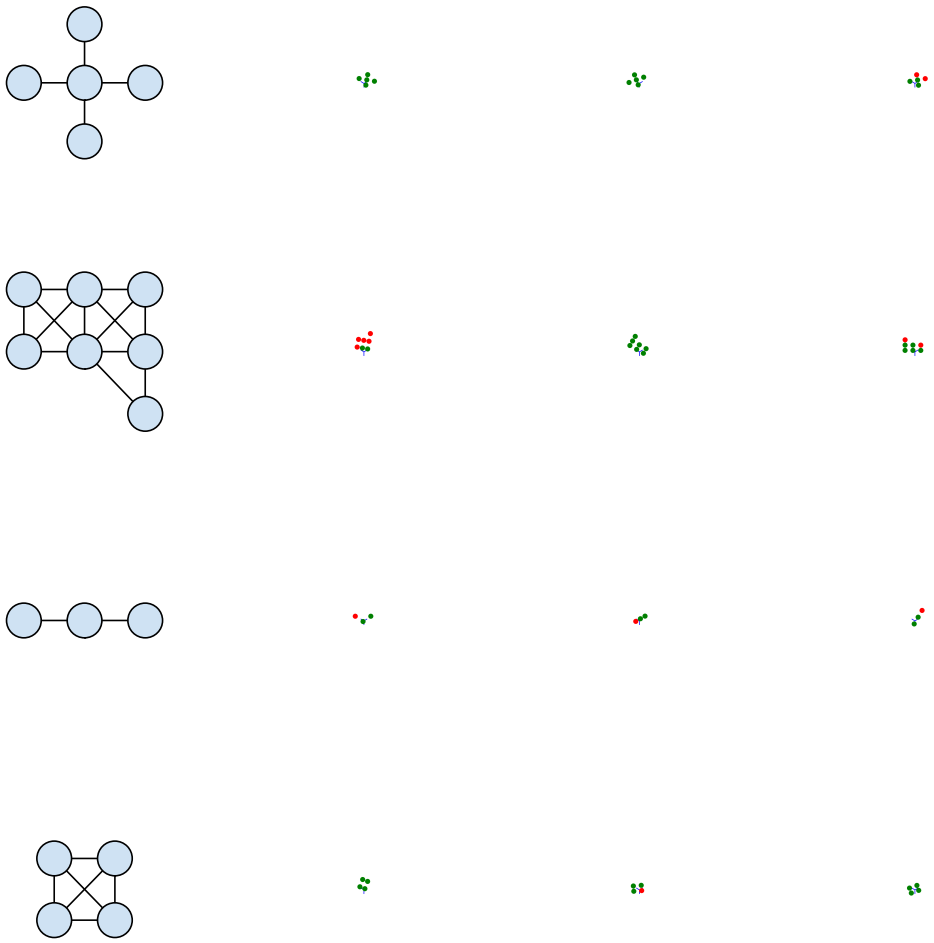

Figure 6.23: Examples of clusters corresponding to isomorphic graph structures discovered by DBSCAN (FEATHER). Each row corresponds to a different cluster and shows the graph structure along with a few examples.

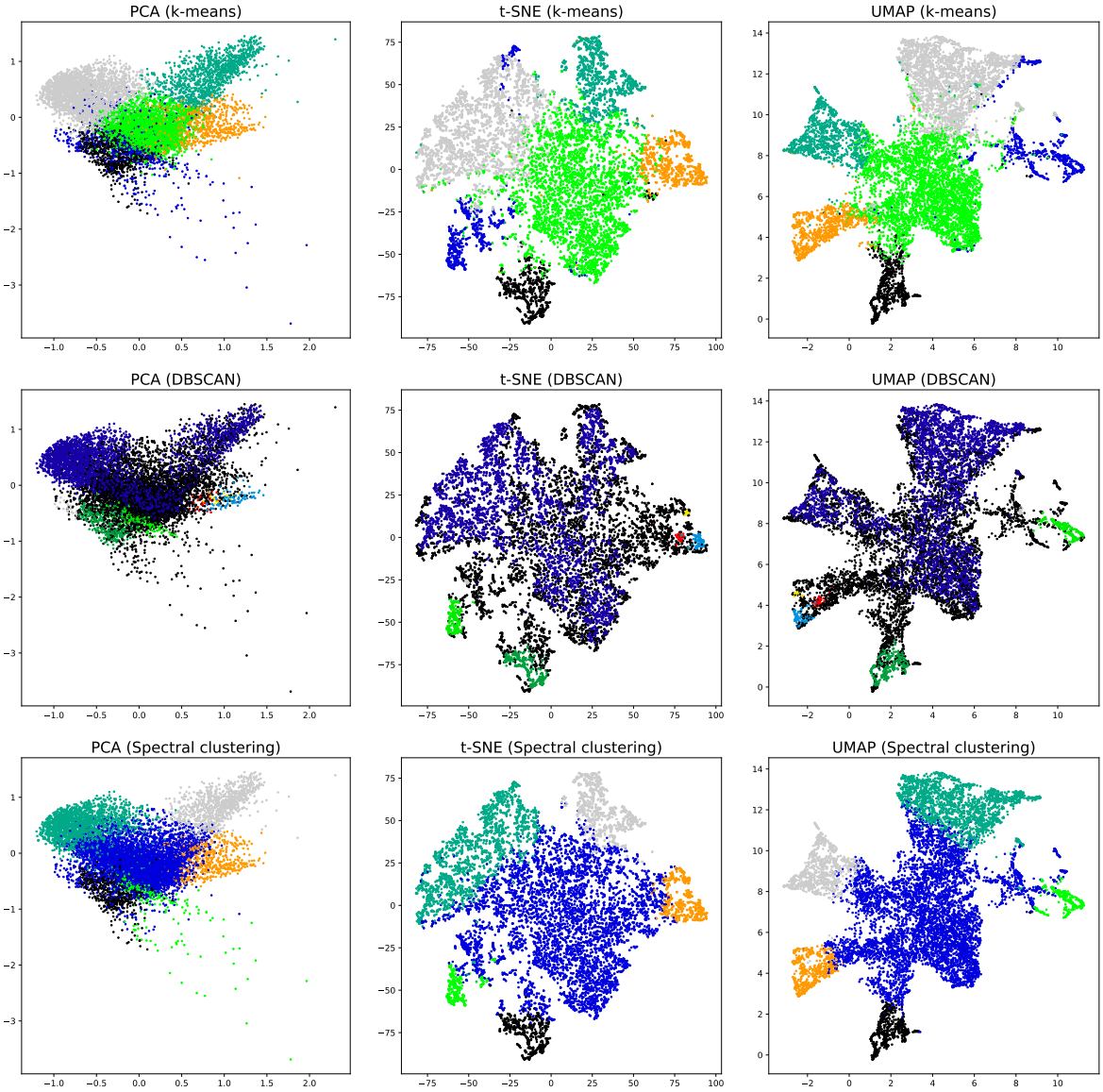

Figure 6.24: Clustering results on VGAE embeddings visualized with PCA (left), t-SNE (center), and UMAP (right), using k-means (top row), DBSCAN (middle row), and Spectral clustering (bottom row).

Table 6.6: Adjusted Rand Index (ARI) between clustering methods (VGAE)

	k-means	DBSCAN	Spectral
k-means	1.00	0.13	0.52
DBSCAN	0.13	1.00	0.18
Spectral	0.52	0.18	1.00

Table 6.7: Adjusted Mutual Information (AMI) between clustering methods (VGAE)

	k-means	DBSCAN	Spectral
k-means	1.00	0.26	0.62
DBSCAN	0.26	1.00	0.30
Spectral	0.62	0.30	1.00

The parameter selection was very challenging, as neither the visualizations with PCA, t-SNE and UMAP, nor the radial sector maps could guide the choice. Finally, ε was set to 0.45 and `min_samples` to 20, resulting in 7 clusters and 50.58% of outliers.

Spectral clustering The eigengap heuristic suggested setting the number of clusters k to 1, for which the Silhouette, CH and DB scores could not be computed. Analyzing the trend for different values of k showed that the maximum of the Silhouette and the minimum of the DB scores were likewise achieved at $k = 1$. However, the maximum of the CH score was achieved at $k = 6$. Since analyzing a single cluster is not very informative, similarly as with k-means, k was set to 6.

Observations Unfortunately, the clusters produced by the three algorithms appear mixed and are extremely difficult to interpret.

6.2.3 DGLC

As described in Section 5.2.2 the number of clusters was one of the hyperparameters of the model to be specified apriori. This makes the analysis easier, as the number of clusters is 'known', i.e. it doesn't need to be determined using the heuristics, clustering coefficients or visualizations as with the other methods. Three different models were trained with 10, 20 and 30 clusters, respectively. It is important to note that in this setting only k-means and Spectral clustering were used, as they allow the number of clusters to be specified in advance, unlike DBSCAN. Table 6.8 gives the Silhouette, CH and DB scores for each configuration. The Silhouette score in all scenarios is very close to zero indicating that the clusters are overlapping. Furthermore, Table 6.9 shows the ARI and AMI indices between the cluster assignments produced by k-means and Spectral clustering for k equal to 10, 20 and 30, respectively.

Observations The clusters produced by the algorithms in all settings seem to correspond to the size of the area covered by the cell, i.e. the number of tiles, similarly as in Sections 6.1.2 and 6.2.1, although mixed. Figure 6.25 shows bar plots of the average number of tiles per cell per cluster (with standard deviation) for k-means and Spectral clustering for different number

Table 6.8: Clustering coefficients for k-means and Spectral clustering (DGLC)

Number of clusters (k)	k-means			Spectral clustering		
	Silhouette	CH	DB	Silhouette	CH	DB
10	0.09	932.86	1.95	-0.16	357.65	2.06
20	0.07	1338.18	1.52	-0.12	505.11	1.98
30	0.03	594.94	1.85	-0.14	207.72	2.34

Table 6.9: ARI and AMI indices between k-means and Spectral clustering cluster assignments for different numbers of clusters (DGLC)

Number of clusters (k)	ARI	AMI
10	0.07	0.31
20	0.10	0.37
30	0.21	0.47

of clusters. Furthermore, the radial sector maps show no conclusive patterns in terms of the positions of the red throughput dominance, SINR or RSRP tiles. The only exception was when the number of clusters was set to 30, as both k-means and Spectral clustering identified a few clusters with a higher fraction of red throughput dominance tiles surrounding the mobile cell site. Figure 6.26 shows the radial sector maps in this setting, together with a few examples for both k-means and Spectral clustering.

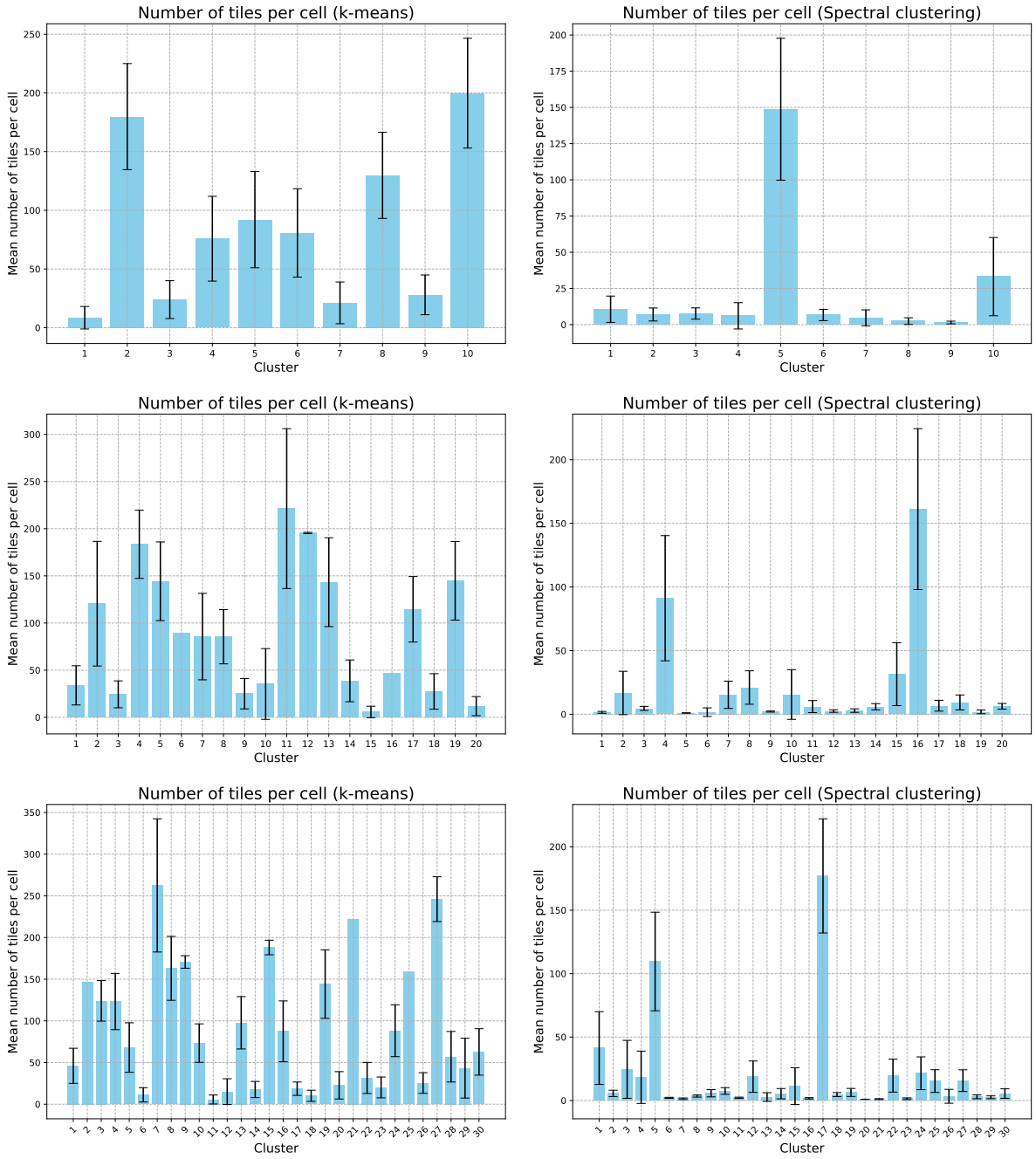

Figure 6.25: Average number of tiles per mobile cell per cluster for 10 (top row), 20 (middle row), and 30 (bottom row) clusters produced by k-means (left) and Spectral clustering (right) (DGLC).

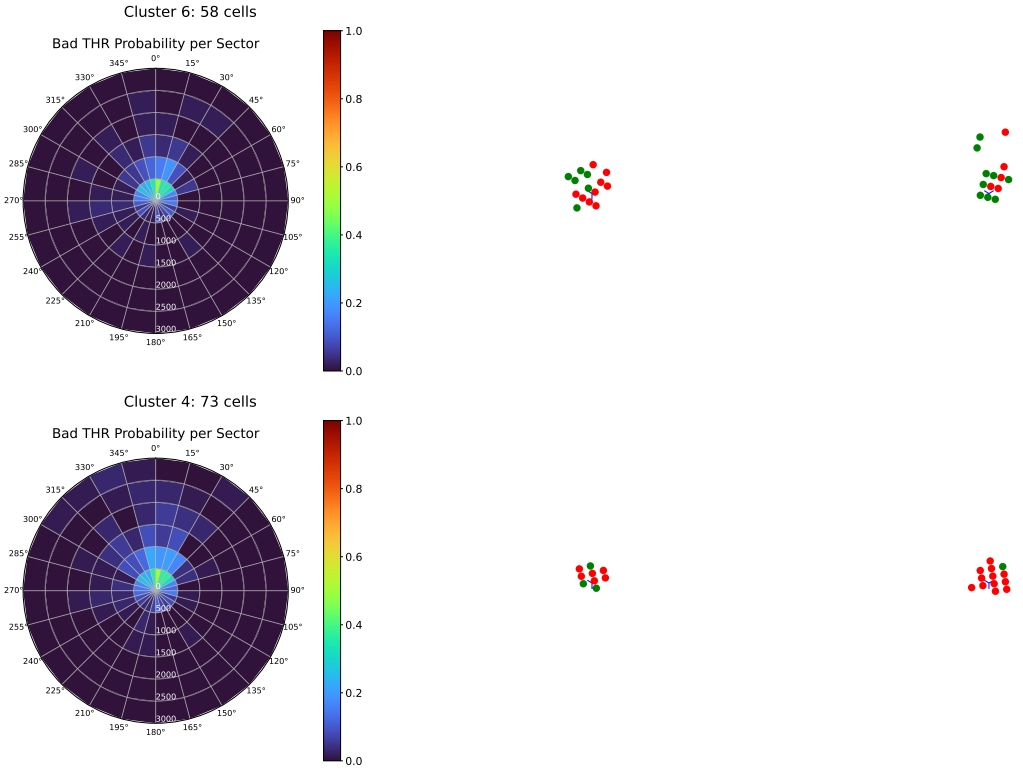

Figure 6.26: Radial sector maps of bad throughput probability for Cluster 6 (k-means; top row) and Cluster 4 (Spectral clustering; bottom row), along with two example mobile cells (DGLC).

7 Discussion

Mobile network optimization is a promising area of research that can benefit from modern applications of machine learning. As discussed, network tuning relies on the expertise of radio network engineers, who analyze performance data to detect and resolve radio issues. However, this process is repetitive and time-consuming. The goal of this thesis was to explore unsupervised machine learning techniques for clustering a set of mobile cells based on their similarity. By grouping mobile cells with similar characteristics, network engineers could analyze entire clusters simultaneously, determine whether there is a shared problem, and apply a common solution, saving valuable time.

7.1 Summary of results

The main challenge during this project was evaluation. The dataset of mobile cell performance data contained no ground-truth labels, making it difficult to assess the validity of the clusters produced by a single approach, let alone compare multiple methods. As presented, the objective was to discover different spatial performance patterns; however, throughout the entire thesis, the notion of a pattern remained vague.

Nevertheless, using the evaluation pipeline described in Section 5.4, results from different embedding learning methods and clustering algorithms were analyzed. The pretrained CNN (Section 6.1.1 and single-layer KPI CAE (Section 6.1.2) identified the clearest patterns, mainly related to the direction and distance of red tiles relative to the mobile cell site. However, these patterns were only useful to radio network engineers when the red tiles were outside the coverage area of the cell (determined by its range, beam direction, and sector angle). The multi-layer CAE (Section 6.1.2) grouped cells based on the size and shape of their coverage areas, as indicated by the tiles. On the other hand, embedding learning methods on graphs were less successful. The GAE and VGAE (Section 6.2.2) produced results that were not interpretable, while the clusters from FEATHER (Section 6.2.1) and DGLC (Section 6.2.3) mostly reflected the size of the graphs, similar to the multi-layer KPI CAE. FEATHER also managed to completely isolate isomorphic graphs (i.e., graphs with identical structure) in some clusters, but this was not useful for the purposes of this thesis.

Regarding the clustering algorithms, hyperparameter selection was generally difficult, suggesting that the data was not well-suited for clustering. Heuristics and clustering coefficients usually did not provide meaningful guidance. There was no clear consensus on which clustering algorithm performed best: in some scenarios, DBSCAN outperformed k-means, but in others it failed completely.

Additionally, the idea of grouping mobile cells together does not necessarily make the task of network engineers easier. Specifically, during the validation phase, engineers still needed to examine each of the cells carefully and determine what makes them similar. In other words, explainability was missing, making the proposed semi-automatic process unreliable. Just because mobile cells had a red tile in the same position did not mean they necessarily had a problem, or that they shared a common problem. Engineers always look at neighboring cells, something that was not considered in this project.

7.2 Future directions

During this thesis, several alternative directions were proposed but not pursued due to the lack of available data. Some of them are listed below and may serve as potential directions for future work.

Predict the problem Instead of clustering mobile cells and hoping that elements of a cluster could be linked to similar radio issues, one could try to predict the radio issue directly. This approach would require a dataset of mobile cell performance data labeled with the corresponding problems. To collect this dataset, network engineers could indicate the issue directly from a predefined set (e.g., *back-lobe phenomenon*). However, this assumes that all possible patterns are known in advance. Alternatively, engineers could provide a textual description on what they observe, which could then be processed by a large language model (LLM) to synthesize a set of labels.

Predict the action Network tuning is a process that involves two main steps: *detection* and *mitigation*. The detection phase involves analyzing the performance of a mobile cell using various metrics and tools to determine whether an issue is present. The mitigation phase consists of deciding which action to take to resolve the identified problem. These resolutions include increasing or decreasing the mobile cell's power, adjusting the tilt (upwards, downwards or sideways), or tuning neighboring cells. Hence, one could try to predict which changes to the configuration are the most likely to improve performance. This direction requires a dataset of mobile cell performance characteristics together with the changes made to their configuration parameters. Fortunately, such a dataset would be relatively easy to collect, since before-and-after configuration parameters (e.g., power in watts, tilt in degrees) can be automatically recorded. The predictions can target individual configuration parameters, indicating whether each should

be increased, decreased, or left unchanged.

Predict the result Building on the previous ideas, one could aim to predict future performance characteristics, such as the footprint of a mobile cell after a configuration change. When the number of possible changes is small, such a classifier could achieve the same outcome as directly predicting the action, since one could simply select the action that leads to the best performance. Since tuning is often iterative, it can take several weeks to fully optimize a group of mobile cells. Training such a model would require before-and-after performance data, along with the specific configuration changes that were made.

Interactions between cells All of the presented directions can be approached with varying levels of complexity (e.g., considering only a single mobile cell, or including its neighboring cells as well). Predicting the correct action or the outcome can be challenging, as mobile cells are rarely independent. There are often mutual interferences and interactions that affect performance.

8 Conclusion

This thesis addresses the problem of unsupervised pattern discovery for mobile network tuning optimization. Specifically, the goal was to cluster mobile cells based on the similarity between their performance measured by different KPIs. The motivation behind was to propose a semi-automatized tool, aiming to reduce the workload of radio network engineers. Different data representation methods, embedding learning techniques and clustering algorithms were used to achieve this goal. Due to the lack of ground truth labels, the setting was fully unsupervised, which made evaluation challenging.

Throughout the duration of this thesis, the notion of a *spatial performance pattern* remained vague, which made it difficult for the project to have a clear question and rather resulted in a thesis with an exploratory nature. Although promising, the applied techniques failed to capture performance patterns useful to identify radio network issues. In scenarios where some patterns were identified, those were either irrelevant to the radio network engineers or linked to a problem with a neighboring cell. Concretely, the pretrained CNN and the single-layer KPI CAE identified the clearest patterns, mainly related to the direction and distance of red tiles relative to the cell site, while the multi-layer KPI CAE grouped cells by the size and shape of their coverage areas. Graph-based embedding methods were less successful: the GAE and VGAE produced uninterpretable results, while FEATHER and DGLC mostly captured the graph size, with FEATHER occasionally isolating isomorphic graphs in separate clusters.

Future work could benefit from labeled data, which would make evaluation more reliable and allow comparing methods in a more meaningful way. The project showed that while clustering can reveal similarities between mobile cells, these alone are not enough to support network optimization.

Bibliography

Raivio, Kimmo et al. (Apr. 2003). "Analysis of Mobile Radio Access Network Using the Self-Organizing Map". In: pp. 439–451. ISBN: 1-4020-7418-2.

Gómez-Andrades, Ana et al. (2016). "Automatic Root Cause Analysis for LTE Networks Based on Unsupervised Techniques". In: *IEEE Transactions on Vehicular Technology* 65.4, pp. 2369–2386.

Liu, Xuewen et al. (2019). "KQIs-Driven QoE Anomaly Detection and Root Cause Analysis in Cellular Networks". In: *2019 IEEE Globecom Workshops (GC Wkshps)*, pp. 1–6.

Wang, Shaoxuan and Ramon Ferrús (2021). "Extracting Cell Patterns From High-Dimensional Radio Network Performance Datasets Using Self-Organizing Maps and K-Means Clustering". In: *IEEE Access* 9, pp. 42045–42058.

Zhang, Wuyang et al. (Mar. 2019). "Self-Organizing Cellular Radio Access Network with Deep Learning". In.

Li, Shuyang, Gianluca Francini, and Enrico Magli (2023). "Temporal dynamics clustering for analyzing cell behavior in mobile networks". In: *Computer Networks* 223, p. 109578. ISSN: 1389-1286.

Lu, Shun et al. (2022). "Mobile Networks Classification Based on Time-Series Clustering". In: *2022 IEEE 5th International Conference on Electronics and Communication Engineering (ICECE)*, pp. 65–71.

Mazguła, Jakub, Dariusz Król, and Ireneusz Jabłoński (2024). "Temporal and Multivariate Similarity Clustering of 5G Performance Data". In: *IEEE Access* 12, pp. 114137–114145.

Shibli, Ali and Tahar Zanouda (2024). "Context-Aware Mobile Network Performance Prediction Using Network & Remote Sensing Data". arXiv: 2405.00220 [cs.LG].

Fukushima, Kunihiko (Apr. 1980). "Neocognitron: A self-organizing neural network model for a mechanism of pattern recognition unaffected by shift in position". In: *Biological Cybernetics* 36.4, pp. 193–202.

LeCun, Y. et al. (1989). "Backpropagation Applied to Handwritten Zip Code Recognition". In: *Neural Computation* 1.4, pp. 541–551.

Dubey, Shiv Ram, Satish Kumar Singh, and Bidyut Baran Chaudhuri (2022). "Activation Functions in Deep Learning: A Comprehensive Survey and Benchmark". arXiv: 2109.14545 [cs.LG].

Lecun, Y. et al. (1998). "Gradient-based learning applied to document recognition". In: *Proceedings of the IEEE* 86.11, pp. 2278–2324.

Krizhevsky, Alex, Ilya Sutskever, and Geoffrey E. Hinton (May 2017). "ImageNet classification with deep convolutional neural networks". In: *Commun. ACM* 60.6, pp. 84–90. ISSN: 0001-0782.

Simonyan, Karen and Andrew Zisserman (2015). "Very Deep Convolutional Networks for Large-Scale Image Recognition". arXiv: 1409.1556 [cs.CV].

He, Kaiming et al. (2015). "Deep Residual Learning for Image Recognition". arXiv: 1512.03385 [cs.CV].

Guo, Xifeng et al. (2017). "Deep Clustering with Convolutional Autoencoders". In: *Neural Information Processing*. Lecture notes in computer science. Cham: Springer International Publishing, pp. 373–382.

Kipf, Thomas N. and Max Welling (2017). "Semi-Supervised Classification with Graph Convolutional Networks". arXiv: 1609.02907 [cs.LG].

Veličković, Petar et al. (2018). "Graph Attention Networks". arXiv: 1710.10903 [stat.ML].

Kipf, Thomas N. and Max Welling (2016). "Variational Graph Auto-Encoders". arXiv: 1611.07308 [stat.ML].

Kingma, Diederik P and Max Welling (2022). "Auto-Encoding Variational Bayes". arXiv: 1312.6114 [stat.ML].

Rezende, Danilo Jimenez, Shakir Mohamed, and Daan Wierstra (2014). "Stochastic Backpropagation and Approximate Inference in Deep Generative Models". arXiv: 1401.4082 [stat.ML].

Cai, Jinyu et al. (2023). "Deep Graph-Level Clustering Using Pseudo-Label-Guided Mutual Information Maximization Network". arXiv: 2302.02369 [cs.LG].

Sun, Fan-Yun et al. (2020). "InfoGraph: Unsupervised and Semi-supervised Graph-Level Representation Learning via Mutual Information Maximization". arXiv: 1908.01000 [cs.LG].

Nowozin, Sebastian, Botond Cseke, and Ryota Tomioka (2016). "f-GAN: Training Generative Neural Samplers using Variational Divergence Minimization". arXiv: 1606.00709 [stat.ML].

Rumelhart, David E., Geoffrey E. Hinton, and Ronald J. Williams (Oct. 1986). "Learning representations by back-propagating errors". In: *Nature* 323.6088, pp. 533–536. ISSN: 1476-4687.

Xie, Junyuan, Ross Girshick, and Ali Farhadi (2016). "Unsupervised Deep Embedding for Clustering Analysis". arXiv: 1511.06335 [cs.LG].

Rozemberczki, Benedek and Rik Sarkar (2020). "Characteristic Functions on Graphs: Birds of a Feather, from Statistical Descriptors to Parametric Models". arXiv: 2005.07959 [cs.LG].

Lloyd, S. (1982). "Least squares quantization in PCM". In: *IEEE Transactions on Information Theory* 28.2, pp. 129–137.

Ester, Martin et al. (1996). "A density-based algorithm for discovering clusters in large spatial databases with noise". In: *Proceedings of the Second International Conference on Knowledge Discovery and Data Mining*. KDD'96. Portland, Oregon: AAAI Press, pp. 226–231.

Luxburg, Ulrike von (Dec. 2007). "A tutorial on spectral clustering". In: *Stat. Comput.* 17.4, pp. 395–416.

Shi, Jianbo and J. Malik (2000). "Normalized cuts and image segmentation". In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 22.8, pp. 888–905.

Ng, Andrew Y., Michael I. Jordan, and Yair Weiss (2001). "On Spectral Clustering: Analysis and an algorithm". In: *ADVANCES IN NEURAL INFORMATION PROCESSING SYSTEMS*. MIT Press, pp. 849–856.

Rousseeuw, Peter J. (1987). "Silhouettes: A graphical aid to the interpretation and validation of cluster analysis". In: *Journal of Computational and Applied Mathematics* 20, pp. 53–65. ISSN: 0377-0427.

Caliński, Tadeusz and Harabasz JA (Jan. 1974). "A Dendrite Method for Cluster Analysis". In: *Communications in Statistics - Theory and Methods* 3, pp. 1–27.

Davies, David L. and Donald W. Bouldin (1979). "A Cluster Separation Measure". In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* PAMI-1.2, pp. 224–227.

Rand, William M (Dec. 1971). "Objective criteria for the evaluation of clustering methods". In: *J. Am. Stat. Assoc.* 66.336, p. 846.

Vinh, Nguyen Xuan, Julien Epps, and James Bailey (2009). "Information theoretic measures for clusterings comparison: is a correction for chance necessary?" In: *Proceedings of the 26th Annual International Conference on Machine Learning*. ICML '09. Montreal, Quebec, Canada: Association for Computing Machinery, pp. 1073–1080. ISBN: 9781605585161.

Pearson, Karl and Francis Galton (1895). "VII. Note on regression and inheritance in the case of two parents". In: *Proceedings of the Royal Society of London* 58.347-352, pp. 240–242. eprint: https://royalsocietypublishing.org/doi/pdf/10.1098/rspl.1895.0041.

Spearman, C (Jan. 1904). "The proof and measurement of association between two things". In: *Am. J. Psychol.* 15.1, p. 72.

Kuckartz, Udo et al. (Sept. 2013). *Statistik*. 2nd ed. Vs Verlag Fur Sozialwissenschaften.

Deng, Jia et al. (2009). "ImageNet: A large-scale hierarchical image database". In: *2009 IEEE Conference on Computer Vision and Pattern Recognition*, pp. 248–255.

Jolliffe, Ian (Oct. 2005). "Principal Component Analysis". In: *Encyclopedia of Statistics in Behavioral Science*. Chichester, UK: John Wiley & Sons, Ltd.

Kingma, Diederik P. and Jimmy Ba (2017). "Adam: A Method for Stochastic Optimization". arXiv: 1412.6980 [cs.LG].

Narayanan, Annamalai et al. (2017). "graph2vec: Learning Distributed Representations of Graphs". arXiv: 1707.05005 [cs.AI].

Thorndike, Robert L (Dec. 1953). "Who belongs in the family?" In: *Psychometrika* 18.4, pp. 267–276.

Ketchen Jr, David J and Christopher L Shook (June 1996). "The application of cluster analysis in strategic management research: An analysis and critique". In: *Strategic Manage. J.* 17.6, pp. 441–458.

Schubert, Erich (June 2023). "Stop using the elbow criterion for k-means and how to choose the number of clusters instead". In: *ACM SIGKDD Explorations Newsletter* 25.1, pp. 36–42. ISSN: 1931-0153.

Sander, Jörg et al. (June 1998). "Density-based clustering in spatial databases: The algorithm GDBSCAN and its applications". In: *Data Min. Knowl. Discov.* 2.2, pp. 169–194.

Schubert, Erich et al. (July 2017). "DBSCAN Revisited, Revisited: Why and How You Should (Still) Use DBSCAN". In: *ACM Trans. Database Syst.* 42.3. ISSN: 0362-5915.

Maaten, Laurens van der and Geoffrey Hinton (2008). "Visualizing Data using t-SNE". In: *Journal of Machine Learning Research* 9.86, pp. 2579–2605.

McInnes, Leland, John Healy, and James Melville (2020). "UMAP: Uniform Manifold Approximation and Projection for Dimension Reduction". arXiv: 1802.03426 [stat.ML].

Helber, Patrick et al. (2019). "EuroSAT: A Novel Dataset and Deep Learning Benchmark for Land Use and Land Cover Classification". arXiv: 1709.00029 [cs.CV].

A Appendix

A.1 Pretrained CNN

A.1.1 Radial sector maps

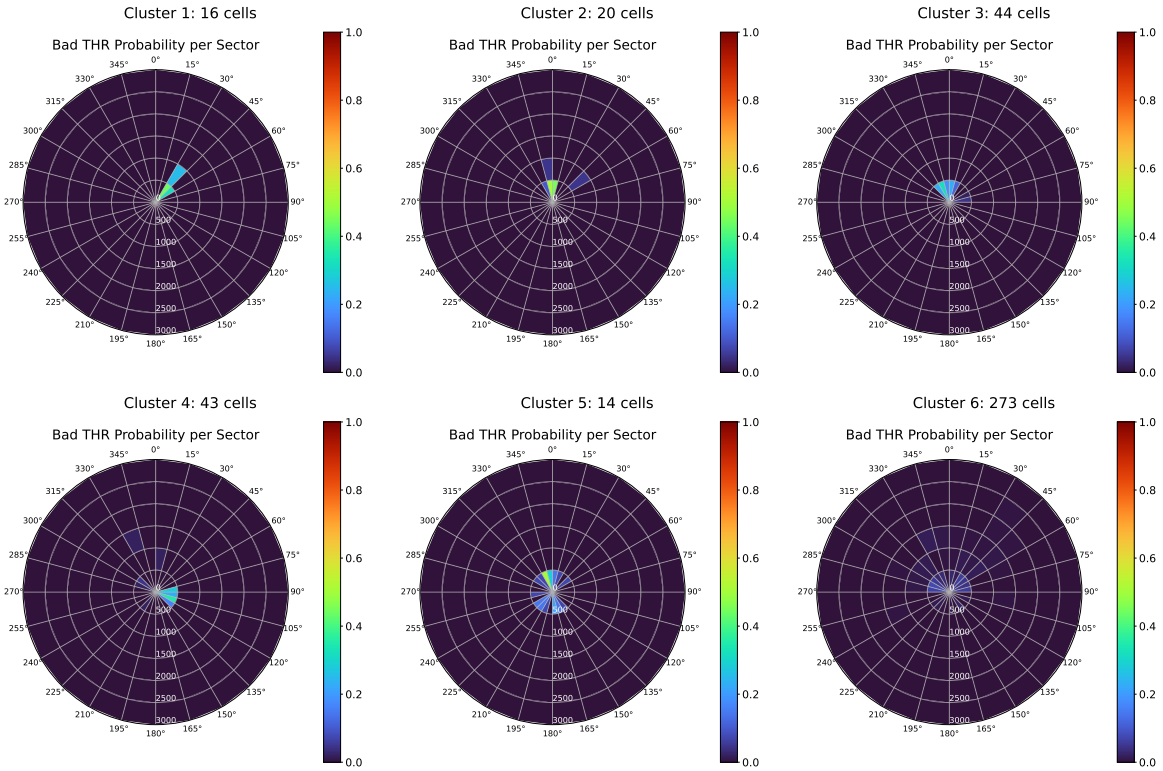

Figure A.1: Spatial performance patterns discovered using DBSCAN on embeddings from the Pretrained CNN, shown as radial sector maps of bad throughput probability per sector for each cluster (1–6).

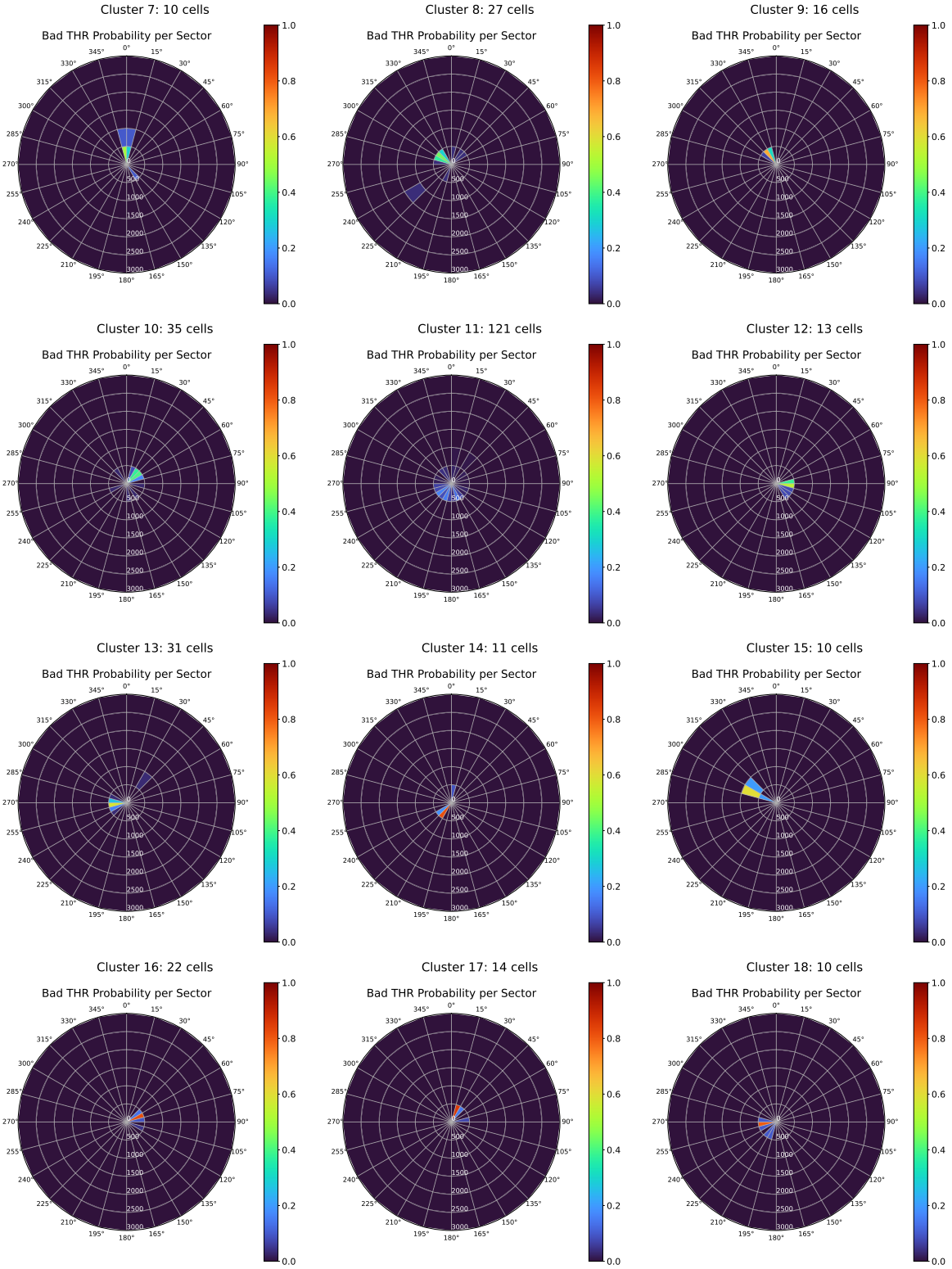

Figure A.2: Spatial performance patterns discovered using DBSCAN on embeddings from the Pretrained CNN, shown as radial sector maps of bad throughput probability per sector for each cluster (7–18).

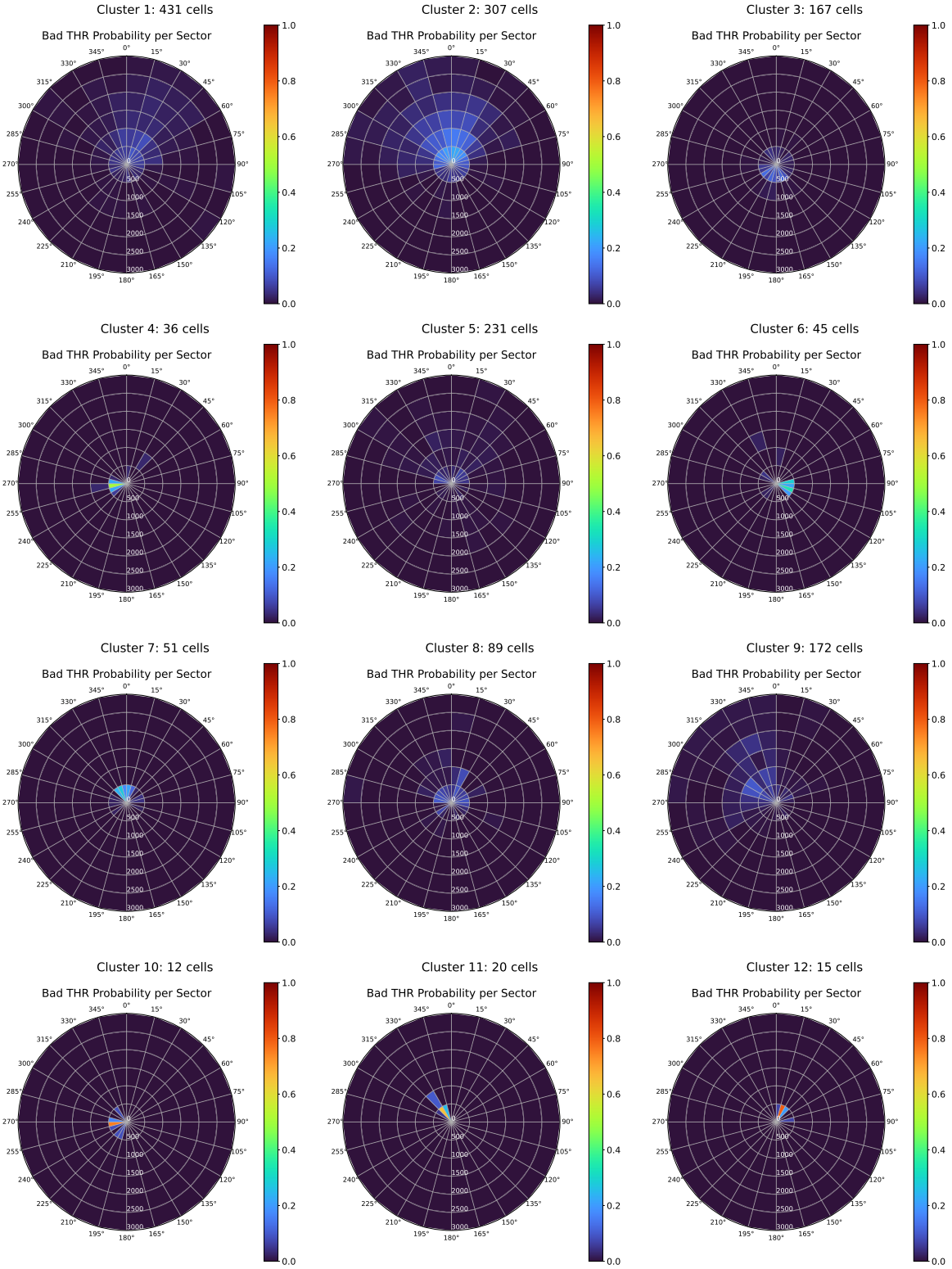

Figure A.3: Spatial performance patterns discovered using Spectral clustering on embeddings from the Pretrained CNN, shown as radial sector maps of bad throughput probability per sector for each cluster (1–12).

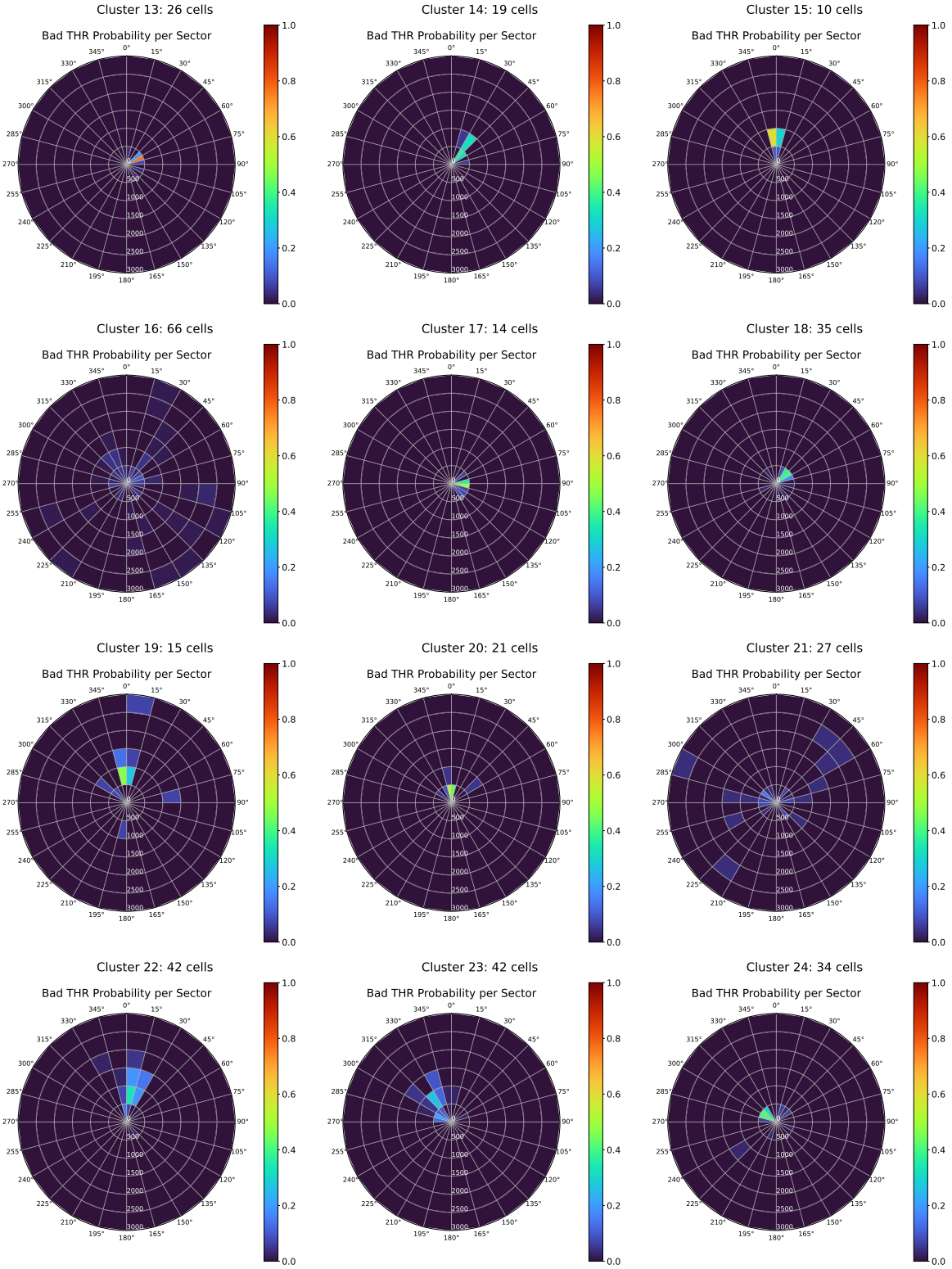

Figure A.4: Spatial performance patterns discovered using Spectral clustering on embeddings from the Pretrained CNN, shown as radial sector maps of bad throughput probability per sector for each cluster (13–24).

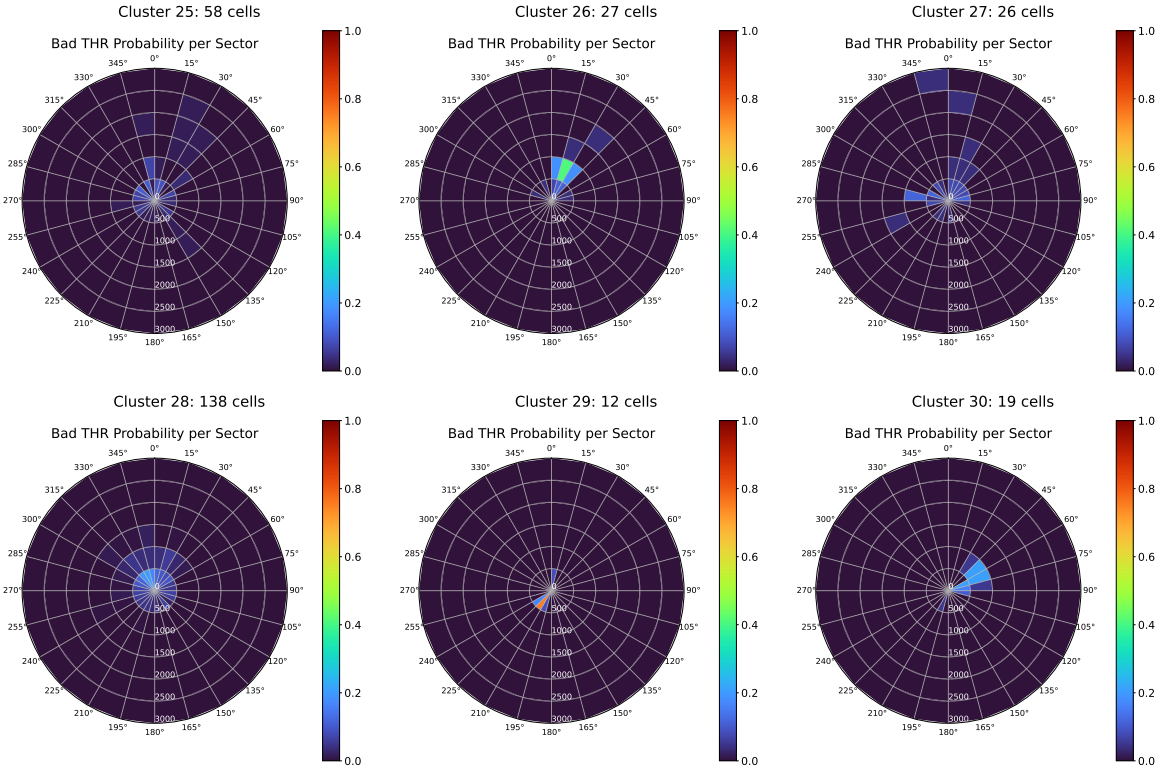

Figure A.5: Spatial performance patterns discovered using Spectral clustering on embeddings from the Pretrained CNN, shown as radial sector maps of bad throughput probability per sector for each cluster (25–30).

A.2 Convolutional Autoencoder

A.2.1 Hyperparameter tuning

In this section, the training and validation losses for different configurations of the CAE architecture are presented. The main hyperparameter to select was the hidden dimension size, which corresponds to the embedding vector size. The models were trained for up to 100 epochs using the Adam optimizer with a learning rate of 0.001. The checkpoint with the best validation loss was saved. Table A.1 presents the training and validation MSE for each configuration. Rows with the lowest validation losses are highlighted in bold.

Table A.1: Training and validation results for different Single-layer KPI CAE configurations

Hidden Dimension	Batch Size	Train Loss (MSE)	Validation Loss (MSE)
8	16	0.000594	0.000692
8	32	0.000687	0.00074
16	16	0.000432	0.000504
16	32	0.000594	0.000618
32	16	0.000352	0.000432
32	32	0.001130	0.001107
64	**16**	**0.00032**	**0.00039**
64	32	0.000671	0.000818

Table A.2: Training and validation results for different Multi-layer KPI CAE configurations

Hidden Dimension	Batch Size	Train Loss (MSE)	Validation Loss (MSE)
8	16	0.005934	0.005764
16	16	0.005460	0.005710
32	16	0.005032	0.005322
64	**16**	**0.004506**	**0.005028**

Figures A.7 and A.7 show the original and reconstructed images from the models with the best validation losses.

A.3 Graph Autoencoder

A.3.1 Hyperparameter tuning

In this section, the training and validation losses for different configurations of the GAE architecture are presented. The main hyperparameter to select was the hidden dimension size, which corresponds to the embedding vector size. The models were trained for up to 100 epochs using the Adam optimizer with a learning rate of 0.01. The checkpoint with the best validation loss was saved. Table A.3 presents the training and validation binary cross entropy for each configuration.

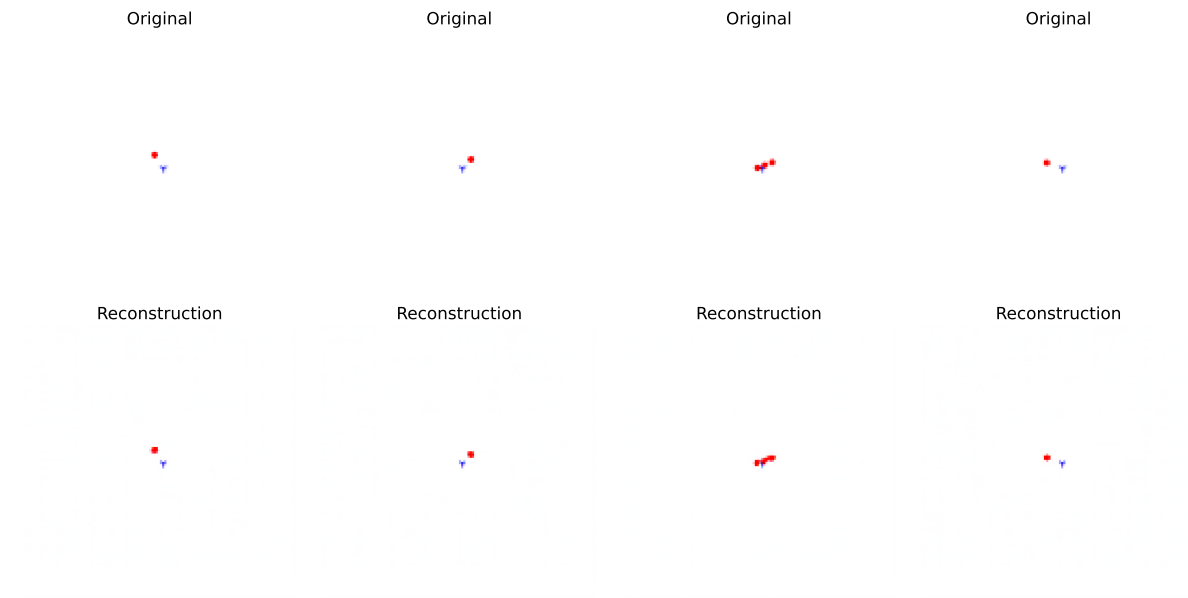

Figure A.6: Original images from the test set and their reconstructions produced by the single-KPI CAE model with a hidden dimension of 64 and batch size of 16.

Rows with the lowest validation losses are highlighted in bold.

Table A.3: Training and validation results for different GAE configurations

Hidden Dimension	Batch Size	Train Loss	Validation Loss
8	32	0.9683	0.9600
16	32	0.9322	0.9204
32	32	0.9115	0.9025
64	**32**	**0.9108**	**0.8975**
128	32	0.9329	0.9096

A.4 Variational Graph Autoencoder

A.4.1 Hyperparameter tuning

In this section, the training and validation losses for different configurations of the VGAE architecture are presented. The main hyperparameter to select was the hidden dimension size, which corresponds to the embedding vector size. The models were trained for up to 100 epochs using the Adam optimizer with a learning rate of 0.01. The checkpoint with the best validation loss was saved. Table A.4 presents the training and validation binary cross entropy for each configuration. Rows with the lowest validation losses are highlighted in bold.

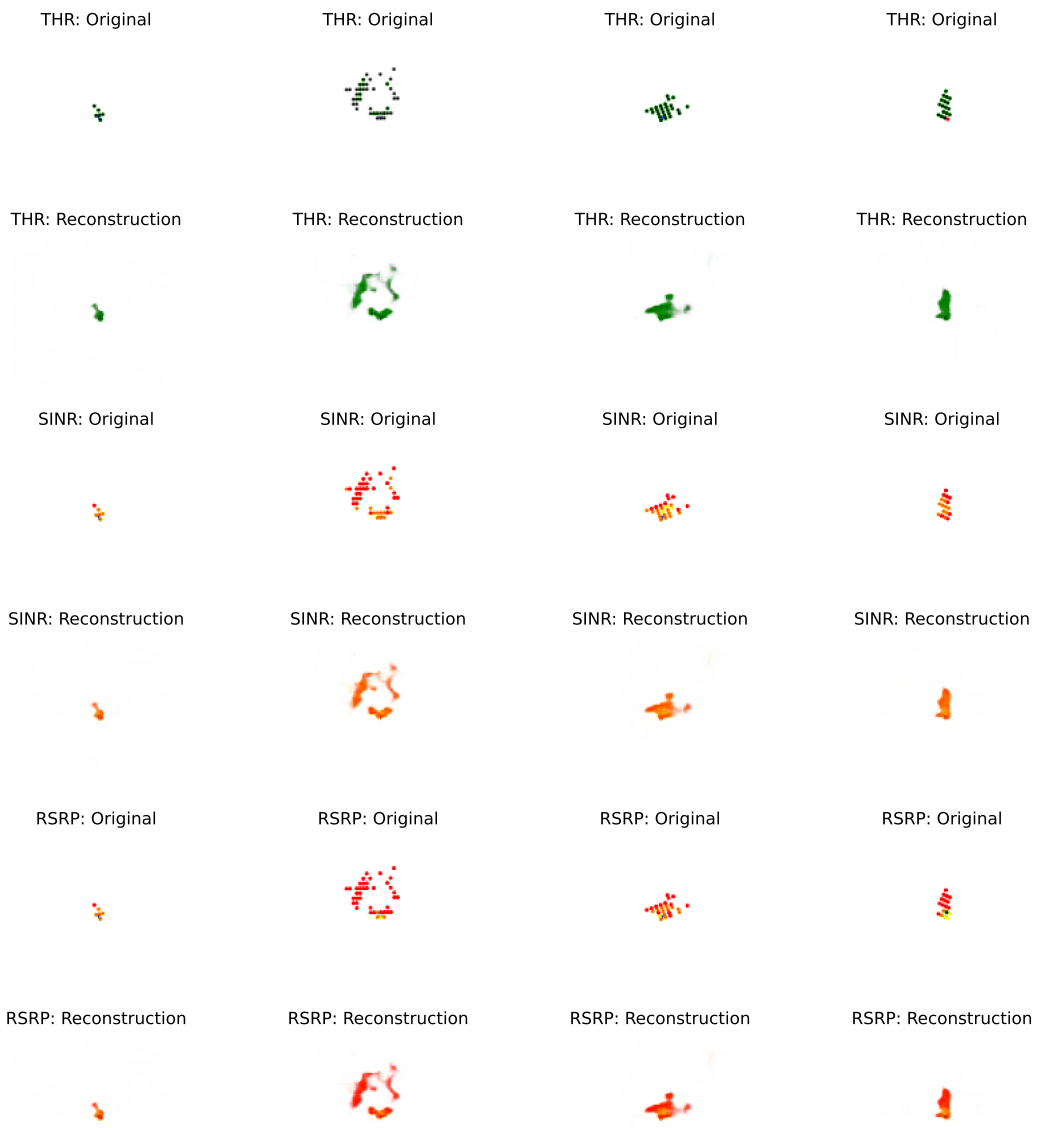

Figure A.7: Original images from the test set and their reconstructions produced by the multi-KPI CAE model with a hidden dimension of 64 and batch size of 16.

Table A.4: Training and validation results for different VGAE configurations

Hidden Dimension	Batch Size	Train Loss	Validation Loss
8	32	0.9985	0.9835
16	**32**	**0.9749**	**0.9643**
32	32	0.9854	0.9699
64	32	1.0425	1.0238
128	32	1.1601	1.1351

A.5 Deep Graph Level Clustering (DGLC)

A.5.1 Hyperparameter tuning

In this section, the training and the validation losses for different configurations of the DGLC framework are presented. The main hyperparameter to select was the size of the clustering embedding dimension, which corresponds to the size of the graph-level embedding vector. The models were trained up to 100 epochs, using the Adam optimizer with a learning rate of 10^{-5}. The batch size was set to 16 and the dimension of the hidden dimension was set to 64. The number of layers of the GNN was 4. As the number of clusters was to be specified apriori, three different models were trained with 10, 20, and 30 clusters, respectively. Table A.5 presents the training and validation loss for each configuration. Rows with the lowest validation losses are highlighted in bold.

Table A.5: Training and validation results for different DGLC configurations

Number of clusters	Clustering embedding dimension	Train Loss	Validation Loss
10	**16**	**-15.0987**	**-8.3578**
10	32	-15.0868	-4.1977
10	64	-15.1042	-3.1101
20	**16**	**-15.0301**	**-7.6374**
20	32	-15.0727	2.1005
20	64	-15.1246	-0.4739
30	16	-15.0051	-3.8088
30	**32**	**-14.8062**	**-4.8191**
30	64	-15.0305	7.7953

A.6 Satellite images

As mentioned in Section 5.2.1, the geographical context (i.e. the map) was removed during the preprocessing steps of the image-based representations for simplicity. However, as discussed by (Shibli and Zanouda 2024) the geographical location (e.g., satellite imagery) can provide

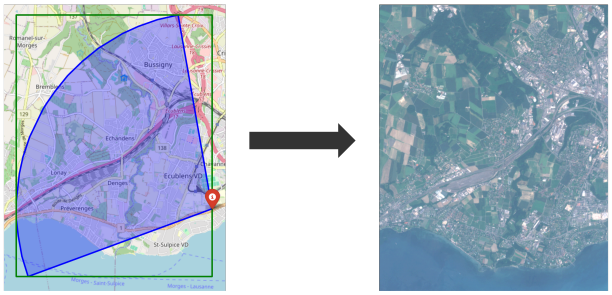

Figure A.8: Illustration of Steps 1 and 2 of the preliminary experimental setup.

information about the topography, urban density and foliage, which influence the performance of mobile cells. (Shibli and Zanouda 2024) partition the set of mobile cells into different clusters based on the similarity between their coverage areas. They train a separate KPI prediction model for each cluster and show that this approach improves performance compared to training a single common model. Similarly as their approach, the preliminary experimental setup consisted of the following steps:

1. Calculate the *bounding box* coordinates for the coverage area of each mobile cell based on the cell range, beam direction, and sector angle. The bounding box is the minimal rectangular area, with sides parallel to the meridians and parallels, which contains the coverage area of the cell. The bounding box is given by two pairs of latitude and longitude.
2. Extract the satellite images corresponding to the bounding box of the coverage areas of each mobile cell using the Sentinel Hub API[I]. An example is given in Figure A.8.
3. Use a pretrained ResNet-50 (He et al. 2015) model on the EuroSAT (Helber et al. 2019) dataset to extract embeddings corresponding to the coverage area of each mobile cell. The last classification of the model was removed, so that the model can be used for feature extraction. PCA was applied to reduce the size of the resulting embeddings.
4. Use a clustering algorithm (e.g., k-means) to partition the set of mobile cells based on the similarity between their coverage ares.

The naive hypothesis behind this experimental setup was that the resulting clusters will reveal spatial performance patterns, which was not the case. However, there is potential in using the geographical context to discover performance issues. As an example, the interpretation by the network engineers in Section 6.1.2 suggests that some behaviors might be considered *normal* if the mobile cell is located outside an urban area, but problematic otherwise. Further research work could adopt the same setting as this thesis and combine it with the information about the geographical location, or it can build on the ideas discussed in Section 7 and train separate prediction models as (Shibli and Zanouda 2024).

[I]https://dataspace.copernicus.eu/analyse/apis/sentinel-hub