Hindawi
Computational Intelligence and Neuroscience
Volume 2023, Article ID 1456971, 14 pages
https://doi.org/10.1155/2023/1456971

Research Article

Road-Type Classification with Deep AutoEncoder

Mohale E. Molefe and Jules R. Tapamo

School of Engineering, University of KwaZulu Natal, Durban, South Africa

Correspondence should be addressed to Jules R. Tapamo; tapamoj@ukzn.ac.za

Received 8 October 2022; Revised 30 January 2023; Accepted 15 February 2023; Published 14 March 2023

Academic Editor: Paolo Gastaldo

Machine learning algorithms are among the driving forces towards the success of intelligent road network systems design. Such algorithms allow for the design of systems that provide safe road usage, efficient infrastructure, and traffic flow management. One such application of machine learning in intelligent road networks is classifying different road network types that provide useful traffic information to road users. We propose a deep autoencoder model for representation learning to classify road network types. Each road segment node is represented as a feature vector. Unlike existing graph embedding methods that perform road segment embedding using the neighbouring road segments, the proposed method performs embedding directly on the road segment vectors. The proposed method performs embedding directly on the road segment vectors. Comparison with state-of-the-art graph embedding methods show that the proposed method outperforms graph convolution networks, GraphSAGE-MEAN, graph attention networks, and graph isomorphism network methods, and it achieves similar performance to GraphSAGE-MAXPOOL.

1. Introduction

Throughout the world, the number of vehicles and road users is increasing, and this has created traffic problems such as traffic congestion, accidents, and fuel cost. The rise of such traffic problems has led to the need to design and develop smart cities. Smart city design integrates physical, digital, and human systems in the built environment to facilitate the planning, construction, and management of the city's infrastructure [1]. Smart cities cover a wide range of applications within the transport and health industries. One key element of smart city design within the transport industry is the intelligent road network system design, which aims to ensure efficient traffic flow by minimising traffic problems. Intelligent road networks have also seen a wide range of applications in the domain of autonomous vehicles.

The use of intelligent road networks system is widely accepted in many countries, and its use is not only limited to traffic flow control and information but also expands to efficient infrastructure and road safety usage. Machine learning algorithms have been the driving force behind the successes of intelligent road networks; indeed, access to big data has opened doors to the development of various intelligent road network models. One such application of machine learning in road networks is classifying different road types. Road-type classification models are becoming important, as they can be embedded in interactive maps to provide helpful traffic information to road users. Other benefits of road-type classification include efficient traffic flow management, avoidance of congested routes, avoidance of routes where accidents are likely to occur, avoidance of routes with many intersections, and model integration to autonomous vehicles.

However, modelling road networks with machine learning is complex due to the lack of available feature extraction methods for representing road-types as feature vectors. Thus, researchers have introduced deep learning embedding methods to learn the spatial information of road networks to automatically extract features in the network data. These embedding methods are termed graph representation learning (GRL) as they rely on the spatial connection of different objects within the road network structure; thus, each object's feature vectors are constructed by leveraging its spatial connection with neighbouring objects. The main goal of GRL methods is to achieve automatic

feature extraction on the non-Euclidean graph data space without relying on the actual object attributes.

In this work, however, a method for representing different road types as feature vectors is proposed, such that machine learning classification algorithms can be trained and evaluated on these features. We take full advantage of the state-of-the-art baseline road network feature extraction method proposed in [2]. Furthermore, we introduce a deep autoencoder (DAE) embedding method to reduce the dimensions of feature vectors obtained by the baseline method. We then pass the feature vectors extracted by our DAE method to several machine learning classification algorithms; we select the classifier with the highest performance measure.

The rest of the paper is organised as follows. In Section 2, we present the literature study and related works. Section 3 provides the materials and methods used to build our model. In Section 4, we present the experimental results obtained by the proposed method, and we further compare the results to some of the state-of-the-art methods found in the literature. Finally, in Section 5, we conclude our work and provide recommendations for future work.

2. Background and Related Work

Graph theory is the fittest paradigm for modelling road networks, as it embraces all the topological information of any road network. Apart from the spatial road networks, graphs diagrammatically represent all transport networks, including highway, transits networks, air, and water. Thus, attributes such as speed, travel times, number of lanes, and head ways can be represented. A network's topological spatial structure is represented by graphs composed of lines and points. Lines are also called edges, while points are nodes or vertices. Therefore, graphs can represent the topology and spatial structure of a road network such that nodes represent intersections, dead-ends, and locations of interest on the roads, while edges represent the road segments between such nodes.

Machine learning in road networks has had many successes in facilitating important traffic information such as traffic forecasting [3–5], speed limit annotation [6–8], and travel time estimation [9–11]. However, machine learning in road networks for modelling road-type classification is often challenging due to a lack of attributes representing different road types. Thus, it sounds reasonable to apply deep learning methods to automatically learn the network's structure and represent every road segment by aggregating its neighbouring road segments. However, solving a learning problem on graphs is challenging. This is because many widely used data types, such as images and texts, are not structured as graphs. Also, the underlying connectivity patterns on the graph-structured data are more complex and non-Euclidean.

The fundamental solution to modelling complex, non-Euclidean patterns is to learn the graph representation from a low-dimensional Euclidean space using the GRL methods. Once the low-dimensional representations are learned, graphs-related problems such as node and link predictions can be achieved. Also known as graph embedding functions, the main goal of the GRL methods is to pack the properties of every road segment into a vector with a smaller dimension; this enables road segment similarity in the original complex graph space to be quantified in the embedded feature space using standard metrics. Several embedding methods have been proposed in the literature for modelling road networks. In [12], a hybrid graph convolution neural network (HGCN) method is proposed for traffic flow prediction in highway networks, where nodes represent the toll stations while edges represent road segments between two toll stations. In addition to modelling the spatial feature of the highway, the authors achieved better traffic flow prediction by considering factors such as time, space, weather conditions, and data type of each toll station.

It is worth noting that the HGCN method proposed in [12] uses local neighbourhood aggregation to learn the spatial connection of toll stations, and it cannot integrate road segment features into the learning process. This is valid since many state-of-the-art GRL methods rely on node features only. However, road segment features in road networks not only provide the connectivity information of two nodes but can also provide important, descriptive information that could be significant for the learning representation. To tackle this problem, the notion of relational fusion networks (RFN) is proposed in [13], for the speed limit classification and estimation tasks. RFN integrates edge information on the representation learning using the novel graph convolution operator. The RFN operator aggregates information over the relations between nodes instead of aggregating the information over neighbouring nodes.

To the best of our knowledge, the work proposed in [2] is the only available work in the literature that classifies different road types on a graph dataset extracted from Open Street Maps (OSMnx). Similar to RFN, the authors used the dual graph generated by the line graph transformation of the original graph to incorporate the edge features into the learning process. Thereafter, a method for generating road segment features is proposed based on information such as the length of the road segment, speed limit, and midpoint coordinates of the adjacent start and end nodes. The authors further compared the performance of learning representation using several embedding methods, including graph convolution networks (GCN) [14], GraphSAGE [15], graph attention networks (GAT) [16], and graph isomorphism network (GIN) [17] in inductive and transductive tasks, and in supervised and unsupervised learning tasks. In addition, a new GRL method, graph attention isomorphism network (GAIN), is proposed.

In our work, we attempt to improve the robustness of road segment features extracted in [2], by using the deep autoencoder (DAE) model as the embedding function; furthermore, we focus on the transductive and supervised learning settings only since these are the settings that achieved the highest accuracy in [2]. Unlike most graph embedding methods proposed in the literature, our DAE model does not construct the vector representation of the target road segment by aggregating over its neighbouring

segments; instead, it operates directly on the high dimensional feature vectors of each road segment and produces compact feature vectors in a much smaller dimensional space. We then pass these compact features into several machine learning algorithms and report the results using the microaveraged $f1$-score. Finally, we compare our highest f1-score to the f1-score obtained using the methods proposed in [2].

3. Materials and Methods

As depicted in Figure 1, our proposed method for road-type classification comprises 6 steps. First, we extract the original road network graph dataset of Linkoping city from OSMnx. Edges in the original graph represent the road segments, while nodes represent information such as intersections and crossroads. In the second step, we transform the original graph into a line graph representing road segments as nodes. In the third step, we use the original and transformed graphs to derive attributes and represent every road segment as a feature vector. To the best of our knowledge, steps 1 to 3 of our proposed method follow a similar procedure proposed in [2]. In step 4, we introduce the deep autoencoder model as the embedding function, and dimensionality reduction is performed. In step 5, we use the feature vectors obtained by our embedding function to train, validate, and test the deep neural networks, support vector machines, and K-nearest neighbor classifiers. We then select the classifier with the highest microaveraged $f1$-score and compare our obtained results to some of the state-of-the-art embedding methods for solving a similar task to ours.

3.1. Input Dataset. Similar to the transductive setting in [2], the input dataset used to conduct the experiments in our work is the road network graph dataset of Linkoping city. The dataset was extracted from OSMnx within a 14 km radius of the city centroid. The obtained graph dataset is represented as $G = (V, E)$, where V and E are set of nodes and set edges, respectively. Edges represent road segments, and nodes represent crossroads, intersections, and junctions. Some of the preprocessing steps on the obtained graph involved transforming G into an undirected graph, consolidating parallel edges, and intersections within a 10 m distance.

3.2. Line Graph Transformation. As depicted in Figure 2, the original graph G is converted to line graph $L(G)$, such that edges (road segments) in G become nodes in $L(G)$ and two edges (two road segments) that share a node (intersection) in G become an edge in $L(G)$. Transforming G to $L(G)$ has two significant advantages. Firstly, graph embedding methods in the literature are designed for nodes and not edges; thus, the transformed graph $L(G)$ has road segments as nodes. Secondly, nodes (cross-roads, intersections, and junctions) on the original graph do not have the essential information required for road-type classification tasks. Algorithm 1 gives the steps used to transform G to $L(G)$.

3.3. Class Distribution. Road segments in OSMnx are tagged with their corresponding road-type labels, thus allowing for a supervised classification task to be accomplished. However, 15 road-type labels are obtained, and some of these labels rarely occur on our obtained dataset. Therefore, the distribution of data is highly characterised by extreme class imbalances. To tackle this problem, we follow the same technique as in [2], where the authors merged and relabelled road types as shown in Table 1.

3.4. Feature Engineering. Feature generation of each road segment is conducted by extracting its descriptive attributes from the edges of the original graph and nodes of the transformed graph. Indeed, attributes such as the width, length, number of lanes, and speed limit of light vehicles and heavy vehicles provide useful road segment information required for feature generation. Nevertheless, we generate the road segment feature vectors using four main components as in [2] to compare the results fairly. As shown in Table 2, these four components yield a 58-dimensional feature vector for every road segment.

Let l represent the road segment length, (x, y) be the midpoint coordinates of two nodes in longitude and latitude directions, respectively, and $S = \{s_1, s_2, s_3, \ldots, s_m\}$ be the one hot encoding vector of m speed limits. Then, the final feature vector of each road segment is generated using Algorithm 2.

3.5. Embedding with Deep AutoEncoder. We introduce the deep autoencoder (DAE) model to achieve the embedding task. In contrast to the graph embedding methods found in the literature, where road segment vector representation is obtained by aggregating over the neighbouring road segments, our DAE model performs embedding directly on the high-dimensional features of each road segment. As shown in Figure 3, our DAE model comprises three crucial components: the encoder, the embedding space, and the decoder. The encoder component takes the D-dimensional road segment feature vectors as input and compresses these into the smaller dimension while preserving as much important information as possible. The preserved N-dimensional feature vectors (where $N \ll < D$) are stored in the embedding space. The decoder component aims to reconstruct the original D-dimensional road segment features by decompressing the N-dimensional features in the embedding space. Taking the above objectives of each component, we can therefore define the learning process of our DAE model into three steps. First, we compress the D-dimensional input road segment features (X) into N-dimensional feature space in the encoder component. Then, we reconstruct the output Y from the small dimension using the decoder component. Finally, we calculate the error difference between the original inputs and the reconstructed outputs and adjust the weight parameters to reduce this difference.

Our DAE model is a fully connected network with an input layer, four hidden layers, and an embedding space layer on the encoder component. The decoder component comprises four hidden layers and an output layer. The output layer has the same size as the input layer in the

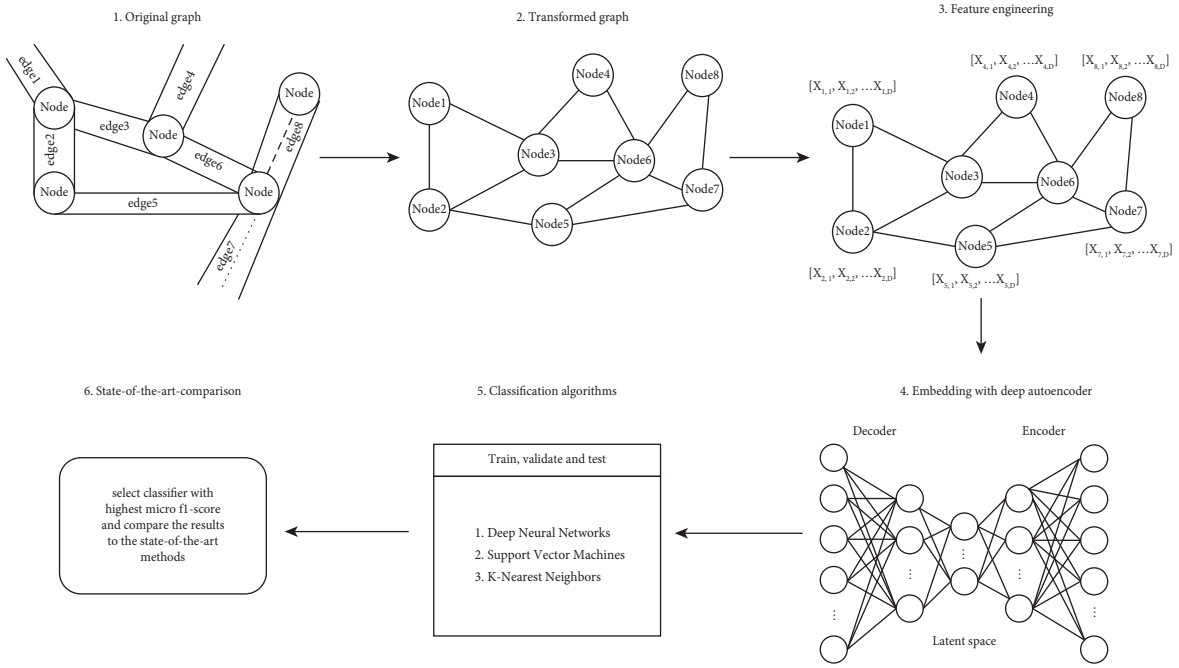

Figure 1: System diagram of the proposed method.

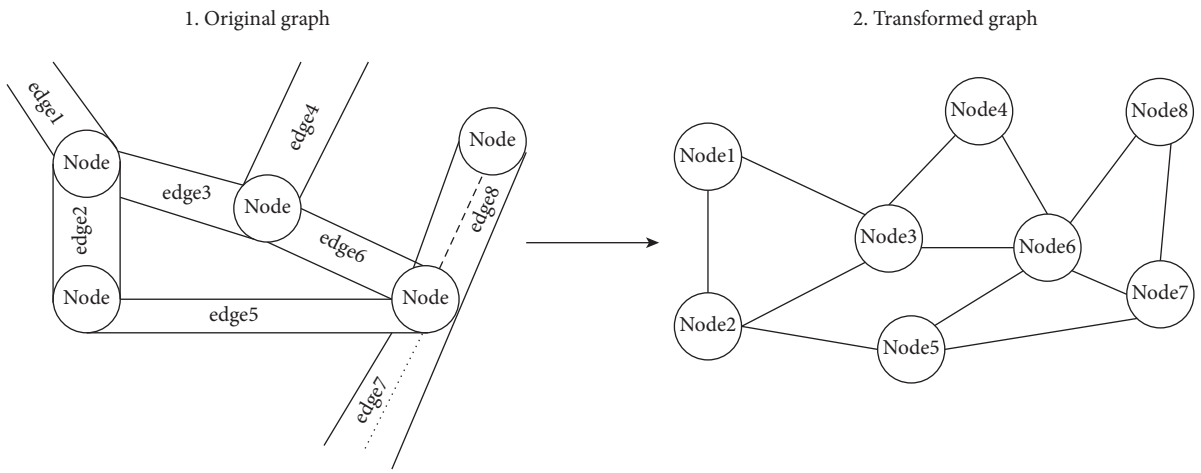

Figure 2: Line graph transformation: edges in the original graph are nodes in the transformed graph, and edges that share a node in the original graph become an edge in the transformed graph.

Require: Original graph G
Output: Transformed graph $L(G)$
(1) Construct $L(G)$ nodes.
(2) **for** each edge $e \in G$ **do**
(3) Set e as a node in $L(G)$.
(4) **end for**
(5) Construct $L(G)$ edges.
(6) **while** two edges $e_1, e_2 \in G$ share a node **do**
(7) Set as (e_1, e_2) an edge in $L(G)$.
(8) **end while**

Algorithm 1: Line graph transformation.

Table 1: Merged and relabelled road types.

Class	Merged road labels	Nodes count
Class 1	Highway, yes, primary, secondary, motorway-link, trunk-link, primary-link, secondary-link	1140
Class 2	tertiary, tertiary-link	951
Class 3	Road, planned, unclassified	922
Class 4	Residential	3012
Class 5	Living street	736

Table 2: Road segment feature generation.

Graph	Road segment attributes	Dimension
G	Road segment length	1-dimension
$L(G)$	Midpoint coordinates between two nodes in longitude and latitude	2-dimensions
$L(G)$	Subtraction of 20 equally spaced distanced points by the midpoint coordinates	40-dimensions
G	One hot encoding of speed limit	15-dimensions

Require: G and $L(G)$
Output: Feature vector for each road segment
(1) **for** each road segment $s \in L(G)$ **do**
(2) **Obtain** the length l_s of s from G.
(3) **Compute** midpoint coordinates (x_s, y_s) of s.
(4) **Obtain** geometry of s.
(5) **if** s does not have a geometry **then**
(6) **Convert** to line geometry.
(7) **Divide** l_s into 20 equally spaced distanced points $(lx_i, ly_i)_{i=1,2,\ldots,20}$.
(8) **for** $i = 1$ to 20 **do**
(9) **Subtract** (lx_i, ly_i) by midpoint coordinates (x_s, y_s).
(10) **end for**
(11) **else**
(12) **Divide** geometry into 20 equally spaced distanced points $(lx_i, ly_i)_{i=1,2,\ldots,20}$.
(13) **for** $i = 1$ to 20 **do**
(14) **Subtract** (lx_i, ly_i) by midpoint coordinates (x_s, y_s).
(15) **end for**
(16) **end if**
(17) **Obtain** one hot encoded vector S of the speed limits with m standard values.
(18) **Concatenate** features generated from steps 2 to 17.
(19) **end for**

Algorithm 2: Feature generation for each road segment.

encoder, while the size of the hidden layers in the decoder is similar to the size of the hidden layers in the encoder. We first normalise the road segment feature vectors on the encoder before feeding them to the input layer. Thereafter, we obtain the value of each neuron in the next compressed layer by computing the sum of products of values in the previous layer and their corresponding weight parameters. We then introduce nonlinearities to the network by applying the rectified linear unit (ReLU) activation function defined as $\mathrm{ReLU}(x) = \max(0, x)$. On the decoder, we decompress values in the embedding space layer and obtain values in the next decompressed layer using a similar procedure; again, the ReLU function is used as the activation function. Furthermore, we normalise the values in the output layer to be between 0 and 1 through the sigmoid function defined as $\mathrm{Sigmoid}(x) = 1/1 + e^{-x}$. This normalisation is important since input features are also normalised. Finally, we measure the error difference between values in the input layer and their corresponding values in the output layer. Therefore, our optimisation problem is finding the set of optimal weight parameters on the encoder component that achieves the smallest possible error difference. Finally, we extract features in the embedding space layer which we later use to train, validate, and test the machine learning algorithms. Algorithm 3 shows the step-by-step implementation of our DAE model for an embedding task. The reasons for choosing the number of hidden layers and corresponding sizes will be given in greater detail in Section 4.

3.6. Road Segment Classification. We use the obtained embedded features, Z, in N-dimensional feature space (where $N = 8$) to compare the performance of deep neural networks (DNN), support vector machines (SVM), and K-

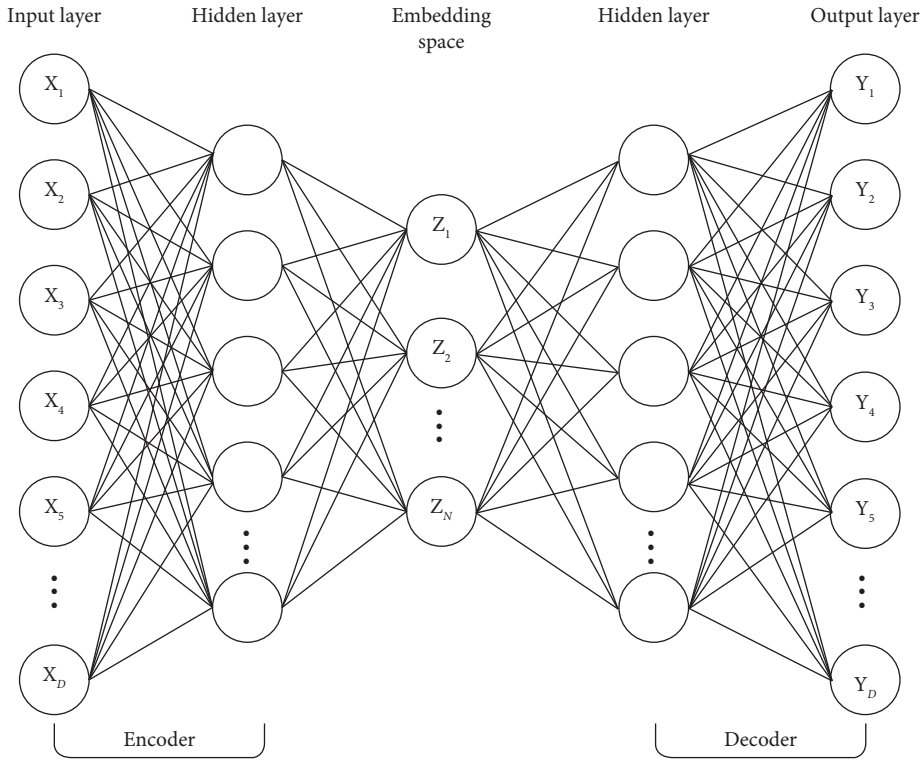

Figure 3: Illustration of the basic deep autoencoder model.

Require: Road segment features space: RSFS $\subset \mathbb{R}^D$
Outputs: Road segment embedded features space: RSEFS $\subset \mathbb{R}^N$
(1) Define encoder parameters:
(2) Input layer: $n = D$.
(3) Hidden layer1: $n_1 = n - 10$, Activation = ReLu.
(4) Hidden layer2: $n_2 = n_1 - 10$, Activation = ReLu.
(5) Hidden layer3: $n_3 = n_2 - 10$, Activation = ReLu.
(6) Hidden layer4: $n_4 = n_3 - 10$, Activation = ReLu.
(7) Embedding layer $n_5 = n_4 - 10 = N$, Activation = ReLu.
(8) Define decoder parameters:
(9) Hidden layer1: $n_1 = N + 10$, Activation = ReLu.
(10) Hidden layer2: $n_2 = n_1 + 10$, Activation = ReLu.
(11) Hidden layer3: $n_3 = n_2 + 10$, Activation = ReLu.
(12) Hidden layer4: $n_4 = n_3 + 10$, Activation = ReLu.
(13) Output layer: $n_5 = n_4 + 10 = D$, activation = Sigmoid.
(14) Define DAE model: model(encoder, decoder)
(15) **for** $X \in$ RSFS **do**
(16) **Fit** input feature vectors (X) to DAE model.
(17) **Initialise** weights randomly.
(18) **Obtain** reconstructed feature vectors (Y).
(19) **Compute** the error difference: $(X - Y)^2$
(20) **while** error difference is not converging **do**
(21) **Update** weight parameters.
(22) **end while**
(23) **Store** weights parameters.
(24) **Obtain** the embedding features vector (Z)
(25) RSEFS $\longleftarrow$ RSEFS $\cup \{Z\}$
(26) **end for**
(27) **Return** embedding space features RSEFS

Algorithm 3: Feature embedding with deep autoencoder.

nearest neighbors (K-NN) classifiers for road-type classification of road classes mentioned in Section 3.3. These classifiers were chosen for comparison as they are deemed adequate for multiclass classification tasks across various applications [18–23]. Furthermore, these classifiers represent three unique learning methods: the artificial neural networks, the hyperplane-based, and the instance-based learning methods.

The DNN classifier belongs to a family of artificial neural networks where the network's underlying parameters are fine-tuned to match a given class label for each input vector. The SVM is a hyperplane-based learning method that transforms nonlinearly separable input features into a high-dimensional feature space where input features can be separated linearly. The K-NN classifier belongs to the family of instance-based learning methods; unlike the SVM classifier, where two classes are trained simultaneously, the K-NN achieves multiclass classification tasks in one go, where feature vectors (with class labels) representing multiple classes are stored in a feature space. The K parameter is used to decide the class label of the unlabelled vector. Thus, comparing these three classifiers will signify the best learning method for a road-type classification task.

We initially divided the input features into train and test datasets. We perform the 10-fold cross-validation method on the training dataset to obtain optimal parameters for each classifier; then, we use the test dataset to obtain the microaveraged $f1$-score of each classifier based on the optimal parameters.

3.6.1. Deep Neural Networks. The DNN classifier is a fully connected network with the input layer, two or more hidden layers, and the output layer. The size of the input layer corresponds to the number of components (m) of road segment feature vectors ($X = (x_i)_{i=1,2,\ldots,m}$), and the size of the output layer corresponds to the number of road-type classes ($Y = (y_i)_{i=1,2,\ldots,n}$). The size and number of hidden layers are often fine-tuned for optimal results. The embedded road segment features are passed into the input layer; the outputs from the input layer are fed into the 2^{nd} layer, the outputs from the 2^{nd} layer are fed into the 3^{rd} layer, and so on, and ultimately the outputs from the $(L-1)^{th}$ layer are fed into the L^{th} layer; equations (1) and (2) are used to obtain the value of the i^{th} neuron of the l^{th} layer, u_i^l, by taking the sum of products of values previous layer $l-1$ and their corresponding weight parameters $W = (W_1, W_2, \ldots, W_L)$, where $W_i = (w_{i1}, w_{i2}, \ldots, w_{iS_i})$, and S_i is the size of the i^{th} layer.

$$u_i^1 = x_i, i = 1, 2, \ldots, S_1, S_1 = m, \tag{1}$$

$$u_i^l = \sum_{j=0}^{S_{l-1}} w_{l-1\,j} u_j^{l-1}, i = 1, 2, \ldots, S_l, l = 2, 3, \ldots, L, \tag{2}$$

where w_{i0} is the bias term, S_l is the size of the l^{th} layer, and $S_L = n$.

In equation (3), Sigmoid function, g, is applied to compress outputs to be between 0 and 1 and thus obtain probabilities that a given road segment feature vector belongs to a class.

$$v_i(X, W) = g\left(u_i^L\right), i = 1, 2, \ldots, m. \tag{3}$$

Equation (3) determines the predicted class label for each road segment input feature vector X. Given a training sample representation $(X^k, Y^k), k = 1, 2, \ldots N$, such that Y^k determines the class of X^k, and N is the size of the training sample. The training of our DNN classifier is made by first obtaining the predicted class label, $v_1(X^k, W), v_2(X^k, W), \ldots, v_n(X^k, W)$ based on the randomly initialised weight parameters W for each road segment vector, X^k. The error incurred between predicted outputs and actual class labels $Y^k = (y_1^k, y_2^k, \ldots, y_n^k)$ for all the training samples is measured by the following formula:

$$E(W) = \frac{1}{2} \sum_{1}^{N} \sum_{j=1}^{m} \left(y_i^k - v_i\left(X^k, W\right)\right)^2. \tag{4}$$

During the training process, the parameters vector W will be updated using the following formula:

$$W = W - \lambda \frac{\partial E}{\partial W}, \tag{5}$$

where λ is the learning rate parameter, and $\partial E/\partial W$ is the gradient calculated using the backpropagation. Algorithm 4 shows the steps used to classify road segment features using the DNN classifier.

3.6.2. Support Vector Machines. We perform the multiclass road-type classification task using the one vs. one support vector machines (SVM) formulation. In one vs. one SVM, we train two road-type classes at a time; thus, for M road-type classes, we obtain a total of $M(M-1)/2$ SVM classifiers; we then assign a class label to the unknown road segment feature vector based on the class with majority counts. For any pair of road-type classes with road segment features from the training dataset and the corresponding class labels $(x_1, y_1), (x_2, y_2), \ldots, (x_m, y_m)$ and $y \in \{-1, +1\}$, we construct an optimal hyperplane that separates the two classes with the largest possible margin as shown in Figure 4. This margin is defined as the distance between vectors nearest to the optimal hyperplane (support vectors) from both classes.

The H_1 and H_2 planes defined by $wv_i + b \geq 1$ and $wv_i + b \leq -1$ represent the boundaries for feature vectors that belong to two distinct road-type classes. The margin which must then be maximised for the optimal hyperplane is the distance: $d = 2/|w|$ between the H_1 and H_2 planes. We maximize d by solving a dual optimisation problem defined as

$$\min \frac{1}{2} w^2. \tag{6}$$

Require: m: size of input layer; n: size of output layer
Training set: $TrFS = \{(X^k, Y^k); \quad i = 1, 2, \ldots, m\}$
Test set: $TeFS = \{T_i; \quad i = 1, 2, \ldots, p\}$
Output: DNN structure: W_{Opt}: Optimal weight, L: number of layers
List of labels such that l_i is the class label of the Test set element t_i: $L = (l_i)_{i=1,2,\ldots,p}$
(1) Training phase:
(2) **Initialise** weight parameters structure W
(3) **Define** number of hidden layers, L, and corresponding sizes $(S_i)_{1,2,\ldots,L}$.
(4) **while** optimal parameters are not obtained **do**
(5) **for** each training sample $(X, Y) \in TrFS$ **do**
(6) **Calculate** neurons value using equations (1) and (2).
(7) **Obtain** the predicted output using equation (3).
(8) **end for**
(9) **Calculate** loss using equation (4).
(10) **Obtain** the updated weight parameters using equation (5).
(11) **end while**
(12) **Store** optimal weight parameters W_{Opt}.
(13) Classification phase:
(14) **for** each road segment vector $T^k \in TeFS$ **do**
(15) **Predict** class l_k of T^k
(16) **end for**
(17) **Return** L

Algorithm 4: Road segment classification using DNN.

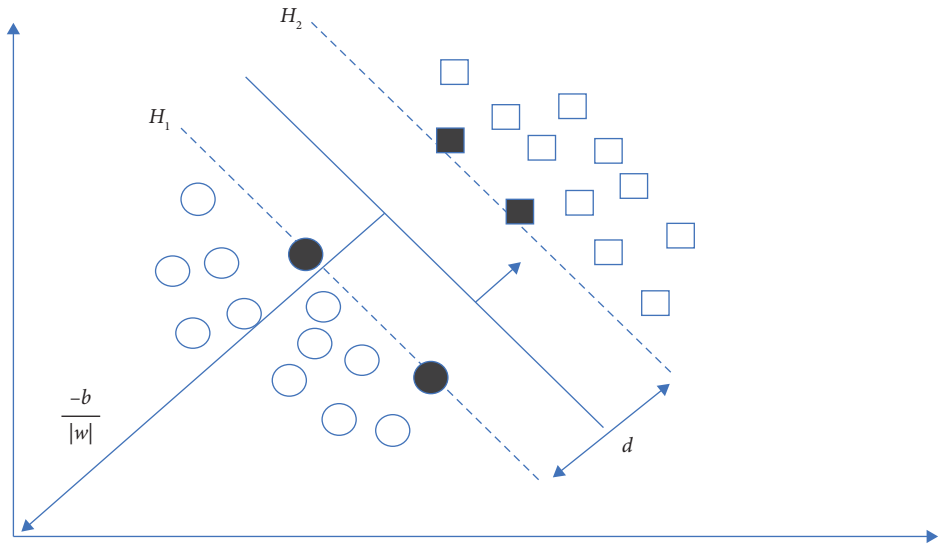

Figure 4: Application of the SVM classifier on two road-type classes.

Subject to the following constraints:

$$y_i = wv_i + b \geq 1, \forall i. \tag{7}$$

Furthermore, we introduce Lagrangian's multipliers to eliminate the constraints, and we obtain the dual SVM formulation defined as maximizing

$$L = \sum_i \alpha_i - \frac{1}{2} \sum_i \sum_j \alpha_i \alpha_j y_i y_j \left(v_i \cdot v_j\right)$$
$$\text{subject to} \sum_i \alpha_i y_i = 0, \alpha_i \geq 0. \tag{8}$$

Solving the dual SVM yields the coefficients of α_i, feature vectors where $\alpha_i > 0$ are the support vectors, and they lie directly on the H_1 and H_2 planes. The dual SVM problem is 0 for $\alpha_i = 0$. Thus, the SVM optimisation problem is affected only by the support vectors. The optimal hyperplane for assigning a class label to an unknown road segment feature vector v is done by evaluating the following function:

$$f(v) = \sum_i^m y_i \alpha_i \left(x_i^T \cdot v\right) + b. \tag{9}$$

Radial basis function (RBF) kernel is used to transform nonlinearly separable features to a higher features space φ

where they can be separated linearly. We compute the transformation by taking the dot product between any pairs of feature vectors using a Kernel function: $K(x_i, x\{\_j\}) = \phi(x_i) \cdot \phi(x_j)$. The RBF kernel is defined as follows:

$$K\left(x_i, x_j\right) = \exp\left(\frac{\left\|x_i - x_j\right\|^2}{2\sigma^2}\right). \tag{10}$$

Algorithm 5 outlines the steps used to classify road segment features using the SVM classifier.

3.6.3. K-Nearest Neighbors. K-nearest road segment feature vectors from the training dataset are used to assign a class label to the unknown feature vector. Thus, given road segment features with their corresponding class labels from the training dataset as $(x_1, y_1), (x_2, y_2), \ldots, (x_m, y_m)$, we calculate the distance between the unknown vector v and vectors in the training dataset using the Euclidean distance defined as

$$d(u, x_i) = \sqrt{\sum_{r=1}^{m} \left(c_r(x_i) - c_r(u)\right)^2}, \tag{11}$$

where $c_r(u)$ is the value of the r^{th} component of the vector u.

We then define a set $V = \{v_1, v_2 \ldots, v_K\}$ of K features from the training dataset nearest to the unknown feature to assign its class label. According to equation (12), the unknown feature vector u is assigned to a class that appears the most within a set of K-nearest feature vectors.

$$l_u \longleftarrow \underset{v \in V}{\operatorname{argmax}} \sum_{i=1}^{K} \delta(l_v, y_i)), \tag{12}$$

where y_i is the class of sample x_i, l_v is the class label of the vector v, and the function δ is defined as follows:

$$\delta(x, y) = \begin{cases} 1, & \text{if } x = y, \\ 0, & \text{otherwise.} \end{cases} \tag{13}$$

Algorithm 6 lists the steps used to classify road segment features using the K-NN classifier. Inputs are the feature vectors initially divided into training and test datasets. We use the training dataset to train the classifier and obtain the optimal K through cross-validation. We use the test dataset to obtain the accuracy of the K-NN classifier based on the optimal K value.

4. Experimental Results and Discussion

The Linkoping city road networks graph dataset [24] is used to train our DAE embedding method. Embedded features are used to train, validate, test, and compare DNN, SVM, and K-NN classifiers for road segment classification tasks. Classifier with the highest microaveraged $f1$-score is selected, and results achieved are compared to some state-of-the-art embedding methods found in the literature for solving a similar problem. Experiments are designed mainly to obtain optimal parameters on our DAE embedding method and classifiers.

4.1. Embedding with Deep AutoEncoder. Input to our DAE embedding method is a total of 6761, and 58-dimensional road segment feature vectors described in Section 3.3. First, the dataset is divided into a 70/30 split, where 70% of the dataset is used to train the DAE model while the remaining portion of the dataset obtains optimal parameters of the DAE method. Optimisation is achieved through the Adam optimiser, while the batch size and maximum iterations were chosen as 1024 and 500, respectively. As depicted in Table 3, we define several DAE models with varying numbers of hidden layers and sizes on the encoder and decoder component and the layer size on the embedding space. This is done to identify optimal parameters that achieve the lowest reconstruction error and the highest accuracy using the validation dataset.

We train each DAE model listed in Table 3 using different learning rates $\{1e^{-4}, 1e^{-3}, 1e^{-2}\}$, and we report the results in terms of the reconstruction error and accuracy based on the validation dataset after 500 iterations. We then select the DAE model with a corresponding learning rate that achieves the lowest average reconstruction error and the highest accuracy as our optimal DAE embedding method.

Figures 5–7 show the performances of three DAE models at increasing learning rates in terms of reconstruction error and validation accuracy. It can be observed that the DAE model with 5 hidden layers and embedding space of 4 units achieved the lowest reconstruction error and the highest accuracy of 0.0013 and 98.82%, respectively, at a learning rate of $1e^{-3}$. The DAE model with 4 hidden layers and embedding space of 8 units achieved the lowest reconstruction error and the highest accuracy of 0.000578 and 99.11%, respectively, at a learning rate of $1e^{-3}$. Finally, the DAE model with 3 hidden layers and embedding space of 10 units achieved the lowest reconstruction error and the highest accuracy of 0.000623 and 98.96%, respectively, at a learning rate of $1e^{-3}$. Based on these observations, we select our DAE embedding method using the model with 4 hidden layers and embedding space of 8 units as it achieves better performance.

4.2. Road-Type Classification. Input dataset to the DNN, SVM, and K-NN classifiers is 6761 road segment features of 8 dimensions obtained by our DAE embedding method. The dataset comprises 5 classes of merged and relabelled road types according to the method described in Section 3.3. We initially divided the dataset into a 70/30 split, where 70% of the data are used to train and validate the classifiers based on the 10-foldcross-validation method. We use the remaining 30% of the data to test the classifiers' performance in the microaveraged $f1$-score based on the optimal parameters obtained by the 10-foldcross-validation method. We obtain the micro $f1$-score of each classifier by first computing the confusion matrix; thereafter, we calculate the sums of the true positives (TP), false positives (FP), and false negatives (FN) across all the classes.

4.2.1. Classification with DNN. Road-type classification with DNN was achieved using the steps mentioned in Algorithm 4. Optimisation was performed using the Adam optimiser, and the ReLU function as the activation function.

Require: Training set: $TrFS = \{(x_i, y_i);\ i = 1, 2, \ldots, m\}$
Test set: $TeFS = \{t_i;\ i = 1, 2, \ldots, p\}$
Output: Parameters: b, α
List of labels such that l_i is the class label of the Test set element t_i: $L = (l_i)_{i=1,2,\ldots,p}$
(1) Classifier training.
(2) **Find** α that satisfies equation (8)
(3) **Compute** $w = \sum_{i=1}^{m} \alpha_i y_i z_i$
(4) **Find** the set S of indices i such that $\alpha_i > 0$
(5) **Compute** $b = 1/|S| \sum_{i \in S} (y_i - \sum_{j \in S} \alpha_j y_j z_j \cdot z_i)$
(6) Classifier testing
(7) **for** each road segment feature $t_i \in TeFS$ **do**
(8) Assign class label l_i to t_i using equation (9)
(9) **end for**
(10) **Return** L

Algorithm 5: Road segment classification using SVM.

Require: Training set: $TrFS = (x_i, y_i)_{i=1,2,\ldots,m}$
Test set: $(TeFS = t_i)_{i=1,2,\ldots,p}$
Output: List of labels such that l_i is the class label of the Test set element t_i: $L = (l_i)_{i=1,2,\ldots,p}$
(1) **for** $i = 1$ **to** p **do**
(2) **for** $j = 1$ **to** m **do**
(3) **Compute** the distance $d_j = d(x_i, t_j)$ of x_i to the element t_j of the training set
(4) **end for**
(5) **Compute** the set V of the K-nearest vectors to x_i based on $(d_l)_{l=1,2,\ldots,m}$
(6) **Compute** l_i the class label of x_i using equation (12)
(7) **end for**
(8) **Return** L

Algorithm 6: Road segment classification using K-NN.

Table 3: Hyperparameter settings for optimal DAE parameters.

No. of hidden layers	Layer size		
	Encoder	Embedding space	Decoder
5	{58, 49, 40, 31, 22, 13}	4	{13, 22, 31, 40, 49, 58}
4	{58, 48, 38, 28, 18}	8	{18, 28, 38, 48, 58}
3	{58, 46, 34, 22}	10	{22, 34, 46, 58}

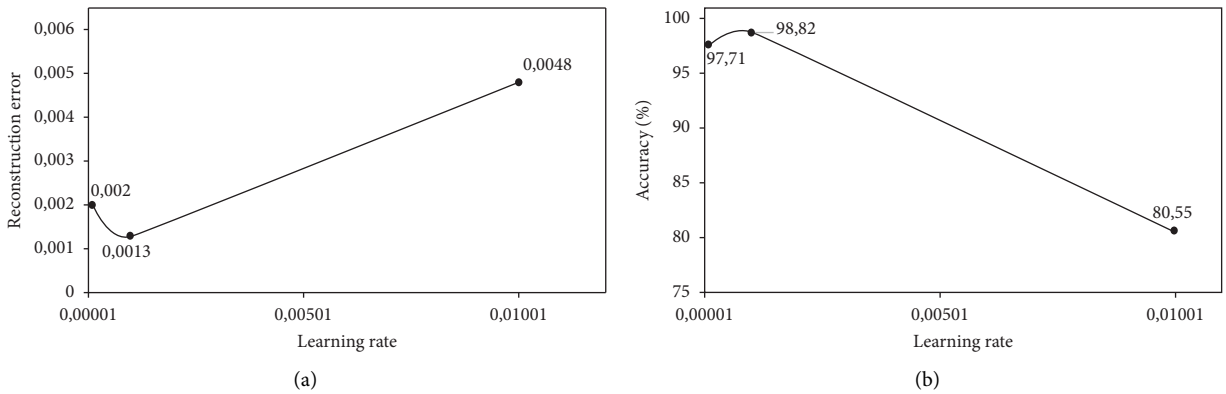

Figure 5: Performance of DAE model at 5 hidden layers of size {49, 40, 31, 22, 13} on the encoder and decoder; and embedding space of size 4: (a) reconstruction error at increasing learning rates and (b) validation accuracy at increasing learning rates.

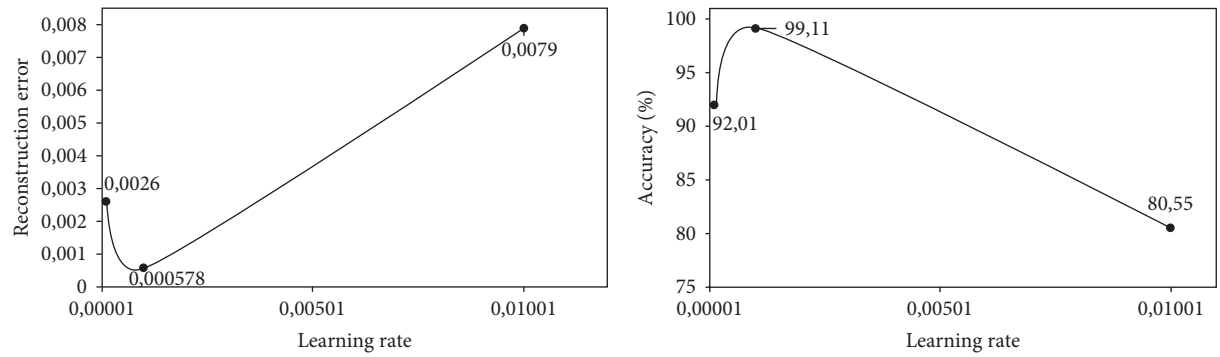

Figure 6: Performance of DAE model at 4 hidden layers of size {48, 38, 28, 18} on the encoder and decoder; and embedding space of size 8: (a) reconstruction error at increasing learning rates and (b) validation accuracy at increasing learning rates.

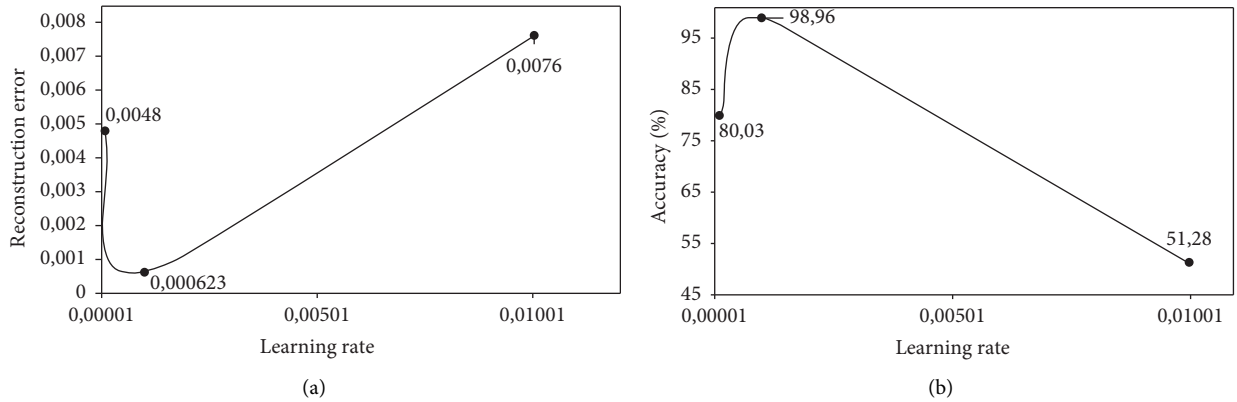

Figure 7: Performance of DAE model at 3 hidden layers of size {46, 34, 22} on the encoder and decoder; and embedding space of size 10: (a) reconstruction error at increasing learning rates and (b) validation accuracy at increasing learning rates.

Various learning rates and the number of hidden layers and sizes were investigated to identify the optimal parameters. The number of hidden layers and sizes was as follows: Layers $A = \{64, 128, 64\}$, Layers $B = \{64, 128, 128, 64\}$, Layers $C = \{400, 800, 800, 400\}$, and Layers $D = \{300, 500, 800, 800, 500, 300\}$. Through the 10-foldcross-validation method, we obtained optimal learning rate of $u = 0.001$ and six hidden layers (Layers D), respectively, as indicated in Table 4.

We used the obtained optimal parameters to train and test the DNN classifier, and we report the results obtained from the test dataset using the microaverage $f1$-score shown in Table 5.

4.2.2. Classification with SVM. Road-type classification with SVM was achieved using the steps mentioned in Algorithm 5. The one vs. one SVM formulation was used, thus giving a total of 10 classifiers. Through the 10-foldcross-validation method, we obtained optimal RBF kernel width of $\sigma = 5$ and the error term parameter of $c = 100$ as indicated in Table 6.

We then used the obtained optimal parameters to train and test the SVM classifier; we report the results obtained from the test dataset using the microaverage $f1$-score as shown in Table 7.

4.2.3. Classification with K-NN. Road-type classification with K-NN was performed using the steps mentioned in Algorithm 6. Through the 10-foldcross-validation method, we obtained the optimal K value of 5 as shown in Table 8.

We used the obtained optimal parameter to train and test the K-NN classifier, and we report the results obtained from the test dataset using the microaverage $f1$-score as shown in Table 9.

Tables 5–7 show the performances of three classifiers using microaveraged $f1$-score for road segment classification tasks. It can be observed that the DNN is the best-performing classifier with the micro $f1$-score of 80.16%. The second best performing classifier is the K-NN, with the micro $f1$-score of 76.13%. The SVM is the worst-performing classifier with the micro $f1$-score of 70.98%.

4.3. Comparison to Other Methods. We then compared the highest microaveraged $f1$-score obtained by our DAE model with some of the state-of-the-art embedding methods presented in [2] for the road-type classification task. Some of the similarities between our study and the study proposed in [2] are as follows: both studies were carried out using the Linkoping City road network graph dataset extracted from OSMnx, both studies transform the original graph to line graph to obtain more descriptive features for each road

Table 4: Average micro $f1$-score for DNN classifier across 10-fold CV at layers and u parameters.

$u = 0.0001$		$u = 0.001$		$u = 0.01$	
Hidden layer size	Avg. micro $f1$-score	Hidden layer size	Avg. micro $f1$-score	Hidden layer size	Avg. micro $f1$-score
Layers A	0.67	Layers A	0.77	Layers A	0.74
Layers B	0.71	Layers B	0.77	Layers B	0.69
Layers C	0.71	Layers C	0.79	Layers C	0.76
Layers D	0.74	**Layers D**	**0.81**	Layers D	0.78

Bold values indicate optimal DNN classifier parameters (hidden layers and learning rate) and the corresponding micro averaged $f1$-score.

Table 5: Micro $f1$-score for DNN classifier at optimal parameters.

Class	TP	FN	FP	Micro $f1$-score
Class 1	279	63	55	
Class 2	162	123	73	
Class 3	201	76	85	$f1 = TP/TP + 1/2(TP + FN) * 100 = 80.16\%$
Class 4	811	93	145	
Class 5	172	49	42	
Total	1625	404	400	

Table 6: Average micro $f1$-score for SVM classifier across 10-fold CV at c and σ parameters.

$c = 1$		$c = 10$		$c = 100$	
Sigma: σ	Avg. micro $f1$-score	Sigma: σ	Avg. micro $f1$-score	Sigma: σ	Avg. micro $f1$-score
0.0001	0.43	0.0001	0.55	0.0001	0.57
0.01	0.57	0.01	0.57	0.01	0.62
0.2	0.62	0.2	10.62	0.2	0.65
2	0.62	2	0.64	2	0.67
5	0.62	5	0.63	**5**	**0.69**

Bold values indicate optimal SVM classifier parameters (sigma and error term) and the corresponding micro averaged $f1$-score.

Table 7: Micro $f1$-score for SVM classifier at optimal parameters.

Class	TP	FN	FP	Micro $f1$-score
Class 1	301	41	113	
Class 2	150	135	91	
Class 3	128	149	49	$f1 = \mathrm{TP/TP} + 1/2(\mathrm{TP} + \mathrm{FN}) * 100 = 70.98\%$
Class 4	851	53	326	
Class 5	16	205	20	
Total	1446	583	599	

Table 8: Average micro $f1$-score for K-NN classifier across 10-fold CV at each K parameter.

K Parameter	Avg. micro $f1$-score
1	0.73
3	0.74
5	**0.76**
7	0.75
9	0.74
11	0.74

Bold values indicate optimal KNN classifier parameter (K value) and the corresponding micro averaged $f1$-score.

segment, and both studies similarly construct a 58-dimensional feature vector representing each road segment. Training of embedding methods for both studies is achieved using 500 iterations and 1024 batch size while optimisation is achieved using the Adam optimiser. The major difference between the two studies is how embedding is achieved. Our DAE method achieves embedding by reducing the dimensionality of each road segment feature vector, while the methods proposed in [2] achieve embedding on each road segment vector by aggregating information from its neighbouring road segments. The final embedded vector obtained from our DAE method has 8 dimensions, while the methods proposed in [2] have the final embedded vectors obtained from one of the output dimensions {64, 128, 56}.

Table 10 shows the comparison of the performance of the methods in terms of the micro $f1$-score. Our DAE embedding method achieves the micro $f1$-score of more than 20% when compared to raw features (original 58-dimensional features without embedding). Furthermore, our DAE method outperforms the GCN, GSAGE-MEAN, GAT, and GIN methods by micro $f1$-score of 22%, 18%, 5%,

Table 9: Micro $f1$-score for K-NN classifier at optimal parameters.

Class	TP	FN	FP	Micro $f1$-score
Class 1	309	33	108	
Class 2	155	130	81	
Class 3	169	108	66	$f1 = \mathrm{TP}/\mathrm{TP} + 1/2(\mathrm{TP} + \mathrm{FN}) * 100 = 76.31\%$
Class 4	749	155	144	
Class 5	166	55	81	
Total	1548	481	480	

Table 10: Method comparisons using micro $f1$-score.

Method	Micro $f1$-score (%)
Raw features [2]	59
GCN [2]	58
GSAGE-MEAN [2]	62
GSAGE-MEANPOOL [2]	81
GSAGE-MAXPOOL [2]	80
GSAGE-LSTM [2]	81
GAT [2]	75
GIN [2]	78
GAIN [2]	81
DAE (proposed)	**80**

Bold values indicate the micro averaged $f1$-score obtained by our proposed method (DAE).

and 2%, respectively. We also observe that our DAE method achieves the same micro $f1$-score of 80% as the GSAGE-MAXPOOL method. Finally, our DAE methods fall short by micro $f1$-score of 1% compared to the GSAGE-MEANPOOL and GAIN methods.

One of the reasons why the proposed method outperforms the state-of-the-art graph embedding methods is that the two methods perform embedding differently. Our DAE acknowledges that not all 58 features representing each road segment are necessary; thus, it performs embedding by reducing the dimensionality of each road segment from 58 dimensions to 8 dimensions representing the most prominent features. On the graph embedding methods, embedding on each road segment feature vector is performed by using the feature vectors of neighbouring road segments; while this allows for modelling spatial connection of road segments, the fact that some features in each road segment are not necessary is ignored, thus yielding less performance compared to our DAE method.

5. Discussion

This study presents a novel representation learning method for a road-type classification task. Compared to other methods found in the literature, which normally perform embedding on each road segment by aggregating information from neighbouring road segments, our method performs embedding by reducing the dimensionality of each road segment while preserving only the important features using the deep autoencoder (DAE) model. To compare the methods fairly, we conducted our experiments using the Linkoping city road networks graph dataset extracted from OSMnx. We then used the same line graph transformation and feature engineering methods as in [2] to represent road segments as nodes and obtain more descriptive features of each road segment, respectively. We then passed the road segment vectors to our DAE embedding methods, obtaining more robust features at much smaller dimensions than the original ones.

We then passed the vectors obtained by our DAE embedding method to the deep neural networks (DNN), support vector machines (SVM), and K-nearest neighbor classifier (K-NN) classifiers to select best performing classifier using the microaveraged $f1$-score. As shown in Tables 5–9, we demonstrated that the DNN is the best performing classifier for road-type classification task of the vectors obtained by our DAE method. We compared our DAE method to some of the state-of-the-art methods experimented in [2] for solving a similar task. These methods include graph convolution networks (GCN), GraphSAGE (MEAN, MEANPOOL, MAXPOOL, and LSTM), graph attention networks (GAT), graph isomorphism networks (GIN), and graph attention isomorphism networks (GAIN).

In Table 10, we demonstrated that our method outperforms the GCN, GSAGE-MEAN, GAT, and GIN methods while achieving similar performance to GSAGE-MAXPOOPL. Furthermore, we observed that our method falls short by 1%, compared to the GSAGE-MEANPOOL and GAIN methods. It is worth mentioning that GSAGE-MEANPOOL and GAIN embedding methods achieve the best performances at much larger dimensions of the embedded feature vectors compared to our method, which achieves comparable performance at a much smaller dimension of the embedded vectors. We also note from Tables 5–9 that merging and relabelling different road types using the method shown in Table 1 is not ideal as several classes (class 2 and class 3) are characterised by many false negatives across all three classifiers, thus, resulting in low micro $f1$-score in all classifiers.

6. Conclusion

This paper proposes a novel deep autoencoder (DAE) embedding method for road-type classification tasks. We used the state-of-the-art feature extraction method found in the literature and represented each road segment as a feature vector. We then applied our DAE embedding method and obtained embedded road segment features which we later used to train, validate, and test several machine learning classifiers. We compared our results to several state-of-the-art graph embedding methods and demonstrated that our method outperforms some of these methods while achieving comparable results to others. It is worth noting that our method performs embedding by reducing the dimensionality of each road segment vector. In contrast, the

graph embedding methods in the literature achieve road segment embedding using the neighbouring road segment features. Therefore, future work will employ a double embedding technique where the vectors obtained by our DAE method are fed as inputs to the graph embedding methods proposed in the literature.

Data Availability

The datasets analyzed during the current study are available at https://planet.openstreetmap.org/.

Conflicts of Interest

The authors declare that they have no conflicts of interest.

Acknowledgments

The authors would like to thank the authors in [2] for publicly making the road network extraction and feature engineering scripts available for experimentation purposes.

References

[1] J. Sahoo and M. Rath, "Study and analysis of smart applications in smart city context," in *Proceedings of the 2017 International Conference on Information Technology (ICIT)*, pp. 225–228, Bhubaneswar, India, December 2017.

[2] Z. Gharaee, S. Kowshik, O. Stromann, and M. Felsberg, "Graph representation learning for road type classification," *Pattern Recognition*, vol. 120, Article ID 108174, 2021.

[3] H. R Deekshetha, A. V. S. Madhav, and K. T Amit, "Traffic prediction using machine learning," in *Evolutionary Computing and Mobile Sustainable Networks*, pp. 969–983, Springer, Berlin, Germany, 2022.

[4] X. Yin, G. Wu, J. Wei, Y. Shen, H. Qi, and B. Yin, "Deep learning on traffic prediction: methods, analysis, and future directions," *IEEE Transactions on Intelligent Transportation Systems*, vol. 23, no. 6, pp. 4927–4943, jun 2022.

[5] J. J. Vázquez, J. Arjona, MaP. Linares, and J. Casanovas-Garcia, "A comparison of deep learning methods for urban traffic forecasting using floating car data," *Transportation Research Procedia*, vol. 47, pp. 195–202, 2020.

[6] P. Szwed, "Speed limits can be determined from geospatial data with machine learning methods," in *International Conference on Artificial Intelligence and Soft Computing*, pp. 431–442, Springer, Berlin, Germany, 2019.

[7] M. Yan, M. Li, H. He, and J. Peng, "Deep learning for vehicle speed prediction," *Energy Procedia*, vol. 152, pp. 618–623, 2018.

[8] S. Modi, J. Bhattacharya, and P. Basak, "Multistep traffic speed prediction: a deep learning based approach using latent space mapping considering spatio-temporal dependencies," *Expert Systems with Applications*, vol. 189, Article ID 116140, 2022.

[9] L. Pereira Masiero, M. A. Casanova, and M. T. M de Carvalho, "Travel time prediction using machine learning," in *Proceedings of the 4th ACM SIGSPATIAL International Workshop on Computational Transportation Science*, pp. 34–38, New York; NY, USA, November 2011.

[10] F. Goudarzi, "Travel time prediction: comparison of machine learning algorithms in a case study," in *Proceedings of the 2018 IEEE 20th International Conference on High Performance Computing and Communications; IEEE 16th International Conference on Smart City; IEEE 4th International Conference on Data Science and Systems (HPCC/SmartCity/DSS)*, pp. 1404–1407, Exeter, UK, June 2018.

[11] X. Song, C. Zhang, and J. Q. Y James, "Learn travel time distribution with graph deep learning and generative adversarial network," in *Proceedings of the 2021 IEEE International Intelligent Transportation Systems Conference (ITSC)*, pp. 1385–1390, Indianapolis, IN, USA, September 2021.

[12] F. Yang, H. Zhang, and S. Tao, "Hybrid deep graph convolutional networks," *International Journal of Machine Learning and Cybernetics*, vol. 13, no. 8, pp. 2239–2255, 08 2022.

[13] T. S Jepsen, C. S. Jensen, and T. D Nielsen, "Relational fusion networks: Graph convolutional networks for road networks," *IEEE Transactions on Intelligent Transportation Systems*, vol. 23, no. 1, pp. 418–429, 2022.

[14] N. Thomas, "Kipf and Max Welling. Semi-supervised classification with graph convolutional networks," 2016, https://arxiv.org/abs/1609.02907.

[15] W. L. Hamilton, R. Ying, and J. Leskovec, "Inductive representation learning on large graphs," *Advances in neural information processing systems*, vol. 30, 2017.

[16] P. Veličković, G. Cucurull, A. Casanova, A. Romero, L. Pietro, and Y. Bengio, "Graph Attention Networks," *stat*, vol. 1050, 2017.

[17] K. Xu, W. Hu, J. Leskovec, and S. Jegelka, "How powerful are graph neural networks?," 2018, https://arxiv.org/abs/1810.00826.

[18] C. J. C. Burges, "A tutorial on support vector machines for pattern recognition," *Data Mining and Knowledge Discovery*, vol. 2, no. 2, pp. 121–167, 1998.

[19] C. Cortes and V. Vapnik, "Support-vector networks," *Machine Learning*, vol. 20, no. 3, pp. 273–297, 1995.

[20] T. M. Mitchell, *Machine Learning*, McGraw-Hill, New York, NY, USA, 1997.

[21] B. Chen, Q. Huang, Y. Chen, L Cheng, and R. Chen, "Deep neural networks for multi-class sentiment classification," in *Proceedings of the 2018 IEEE 20th International Conference on High Performance Computing and Communications; IEEE 16th International Conference on Smart City; IEEE 4th International Conference on Data Science and Systems (HPCC/SmartCity/DSS)*, pp. 854–859, IEEE, Exeter, UK, June 2018.

[22] R. J. Samworth, "Optimal weighted nearest neighbour classifiers," *The Annals of Statistics*, vol. 40, no. 5, pp. 2733–2763, 2012.

[23] Andreas Miroslaus Wichert and Luis Sa-Couto, *Machine Learning-A Journey to Deep Learning: With Exercises and Answers*, World Scientific, Singapore, 2021.

[24] OpenStreetMap contributors Planet dump retrieved from, https://www.openstreetmap.org, 2022.