FACULTY OF ENGINEERING OF THE UNIVERSITY OF PORTO

Node embedding and machine learning towards road network feature prediction

2022/2023

by: Mário Dixo de Sousa

Master in Informatics and Computing Engineering

supervisor: Ana Paula Rocha

July of 2023

Node embedding and machine learning towards road network feature analysis

Mário Dixo de Sousa

Master in Informatics and Computing Engineering

Resumo

Quando comparando a performance de pneus em diferentes mercados e regiões, existe pouca informação sobre como os veículos são operados. Fatores como a topografia e o clima de uma região, qualidade das estradas, tipo e modelo de um veículo, e o comportamento de um condutor têm um grande impacto na degradação dos pneus. A rede de estradas de uma região, por exemplo, tem uma influência substancial no comportamento dos condutores e na forma como os veículos são operados.

Assim sendo, uma comparação direta da informação sobre a performance dos pneus em várias regiões, sem considerar os diferentes atributos da rede de estradas, é desbalanceada e subjetiva. Para introduzir objetividade na comparação, são recolhidos dados de redes de estradas, do OpenStreetMap[20] numa vasta região, e derivados embeddings adequados para representar os mesmos.

No entanto, dados puramente geoespaciais não contêm o mesmo nível de informação sobre as características, como o tipo de superfície ou os sinais de trânsito, para diferentes regiões. Ao longo deste estudo, são usados network embeddings para extrair informação estrutural sobre esta rede, de forma a permitir futuramente que aplicações de machine learning analisem as características das redes de estradas para diferentes fins como completar informação em falta temporariamente num banco de dados da rede de estradas ou avaliar a performance dos pneus como exemplificado previamente, uma vez que a rede de estradas é um tipo de rede espacial onde arestas podem ser associadas a informação qualitativa como o tipo de estrada.

Com este estudo, pretende-se investigar o quão adequado são network embeddings baseadas em redes neuronais para obter características em falta sobre as estradas. Para tal, serve de inspiração outro estudo que verificou a funcionalidade de network embeddings para um cenário semelhante e expando o mesmo através de uma abordagem diferente e mais detalhada.

Abstract

When comparing tire performance from different markets and regions one has little data about how the vehicles were operated. Factors, such as the region's topography and weather, quality of the roads, vehicle type/model, and driver's behavior have a big impact on the rubber loss of the tires. The road networks of a region for instance, have a substantial influence on the drivers' behavior and how the vehicles are operated.

Hence, a direct comparison of tire performance data for various regions without taking the road network information is biased and subjective. To introduce some objectivity in the comparison, we study the road network data from OpenStreetMap for a vast region and derive proper embeddings to represent them.

However, purely geospatial data does not contain the same level of information of road characteristics, such as surface type and traffic signs, for different regions. In the course of this study, network embeddings are used to extract structural information from the network to enable future machine learning applications to analyze a detailed road network and its characteristics for its goals (such as completing temporarily missing data in a road networks database or evaluating the tires performance as exemplified previously), as road networks are a type of spatial network where edges may be associated with qualitative information such as road type.

This study aims to investigate how suitable are neural network based network embeddings to obtain missing road features. For that, inspiration is taken from another case study that verified how suitable network embeddings were for a similar scenario [1] and expand on it using a different and more detailed approach.

Keywords: **road network, machine learning, feature learning, network embedding, spatial networks, neural networks**

Acknowledgements

This project, Node Embedding and Machine Learning Towards Road Network Feature Analysis, had multiple people contributing for its completion, and for that, I take this section to thank them all for the help and support they provided me so that I could surpass the challenges that happened throughout the fulfilment of my thesis.

To the thesis supervisor Ana Paula Rocha, for providing me great guidance when dealing with such a foreign topic, I thank her for her reliable feedback and for helping me in approaching some obstacles during development from a different point of view.

To the company of *Continental* for the suggestion of such a rich and unexplored topic, despite this work not having further relationship with them.

To the company of *Ascendi* for their willingness to allow me to use some of their data related to the topic worked on. However this was not used as it would create bias in the results due to some conflicting characteristics of datasets.

To my family and friends, who supported me and accompanied the development of this work. To my parents, Alexandra and António, my brother Eduardo, and Bárbara, the deepest of my thanks for their constant incentive and depriving themselves of some of their free time and providing support when I needed the most.

Contents

List of Figures

List of Tables

Listings

1. Introduction

1.1 Background and motivation

The road system is a complex spatial network comprising various characteristics that significantly impact vehicle performance, including road surface, road type, speed limits, and road signalling. Unlike other factors like driving style or climate, the above mentioned features affect all vehicles in a similar manner, primarily by exerting wear and tear on tires.

In today's era, where land transportation is an integral part of daily life for many people, readily accessible information about road characteristics could lead to substantial improvements in the quality of services provided by tire companies. By having access to comprehensive data on road conditions across different regions, tire companies can enhance their offerings and, in turn, increase the longevity of tires for individual customers.

Unfortunately, acquiring such information is not easily achievable, as the most detailed databases containing these features are predominantly tailored for Global Positioning System (GPS) applications. Consequently, purely geospatial data lacks the same level of information about road characteristics across diverse regions. Therefore, in this project, the objective is to gather public information and predict missing road features based on previous research efforts. By building upon existing work, we aim to improve the accuracy and reliability of these predictions, thus bridging the information gap and advancing my understanding of road network characteristics.

By addressing this crucial challenge, this research aims to contribute to the development of more informed decision-making processes in the tire industry and enable tire companies to optimize their services for specific regions. Ultimately, this can lead to enhanced customer satisfaction and improved tire longevity, bringing significant benefits to both individuals and the broader transportation ecosystem.

1.2 Context

The research problem for this thesis revolves around the limited accessibility of comprehensive road network characteristics, specifically regarding road surface, road type, speed limits, and road signalling. While these factors significantly impact vehicle performance, the most detailed databases containing this information are primarily designed for GPS applications, which do not offer the same level of granularity for different regions. Consequently, there is a need to bridge this information gap and develop a solution that enables the prediction and completion of missing road features based on publicly available data.

1.3 Objectives

The main objectives of this work can be enumerated in the following paragraphs.

- Evaluate the suitability of neural network-based network embedding methods in prediction tasks: This study aims to assess the effectiveness of neural network-based network embedding techniques in predicting various road characteristics. Building upon the findings of a previous article published in 2018 (updated in 2019), which demonstrated the potential of regular network embedding methods[1], we explore more advanced algorithms that were not considered in the previous study.
- Compare the performance of advanced network embedding algorithms: Specifically, we compare the overall results of two more complex algorithms: GraphSage[2] and GATv2[3]. GraphSage[2], released in the same year as the initial study, represents a significant advancement in network embedding techniques. GATv2[3], on the other hand, is a recently released version of an older algorithm, known for its remarkable improvements, and both algorithms are officially supported by the Deep Graph Library (DGL)[4].

By pursuing these objectives, we seek to determine whether recent GraphSage[2] and GATv2[3] algorithms outperform the previously explored methods, thereby shedding light on the advancements in network embedding techniques.

1.4 Dissertation structure

This first chapter describes the context in which this dissertation is inserted, the motivation that led to its development, and identifies the main objectives to be achieved.

The next chapter, State of the Art, includes a description of some concepts that are relevant for the work presented, technologies used to study the different aspects and problems in the area on which the context fits in, and ends with an exploration of other solutions proposed for the problem.

The third chapter, Problem and Proposed Solution, describes the problem this project studies, what data is used to reach to a potential solution and the approach that was taken for the development of this project.

The fourth chapter, Result Analysis, presents the results of the two approaches, a comparison between them, and the article from which this project drew inspiration from [1].

The fifth and final chapter, Conclusions, expresses the development of the project and its results, along with describing potential future work and improvements that can be made in order to reach better results.

2. State of the Art

This chapter provides an overview of the research topics addressed in this dissertation and aims to briefly review some key concepts relevant to the work developed. It also presents relevant research conducted in the areas of interest, namely network embedding and machine learning in road networks.

2.1 Important concepts

- GRAPH

A graph is a fundamental data structure that comprises nodes (or vertices) interconnected by edges (or links). It is widely used to represent and analyze complex systems, where the nodes represent entities, and the edges capture the relationships or connections between these entities, as illustrated in Figure 2.1. In the context of this dissertation, the entities being represented are roads, making the graph an effective tool for studying the road network.

In this project, several characteristics of graphs play a crucial role, being the most relevant one the graph homophily, which refers to the tendency of similar nodes to have more connections with each other than with dissimilar nodes. In the context of a road network, a higher homophily implies that roads with similar properties, such as road type, traffic patterns, or other attributes, tend to be interconnected or clustered together. An example in a road graph of roads of high homophily would be ‘residential’ roads while low homophily would be ‘link’ type roads (such as entrance and exit of a primary road).

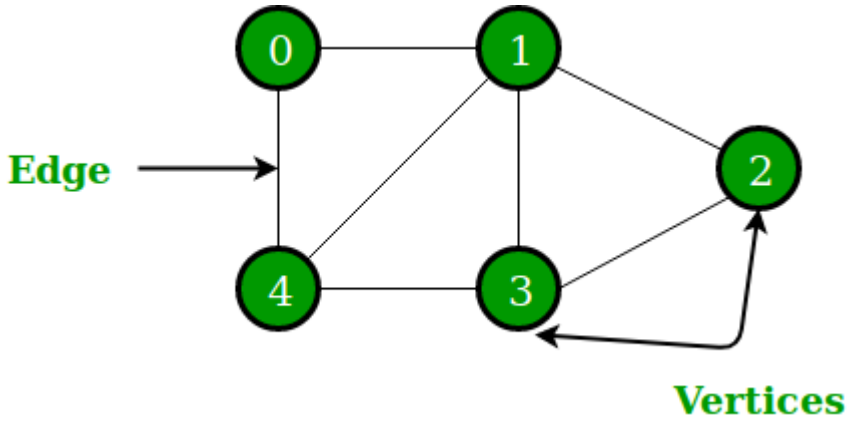


Figure 2.1: Graph structure [5]

- NETWORK EMBEDDINGS

Network embeddings are a fundamental concept in graph representation learning, which is a rapidly evolving field at the intersection of graph theory and machine learning. They provide a way to represent nodes in a network as compact and meaningful vectors in a lower-dimensional space.

The idea behind network embeddings is to encode the structural and relational information of nodes in a network, allowing us to capture important features and patterns of the network's topology. These embeddings enable us to leverage powerful machine learning techniques on graph-structured data, which were traditionally challenging due to the irregular and interconnected nature of networks.

The process of generating network embeddings involves learning a mapping from the original high-dimensional node space to a lower-dimensional embedding space. This mapping aims to preserve the essential structural characteristics and neighborhood relationships of nodes in the network. Figure 2.2 shows a brief explanation on how network embeddings work.

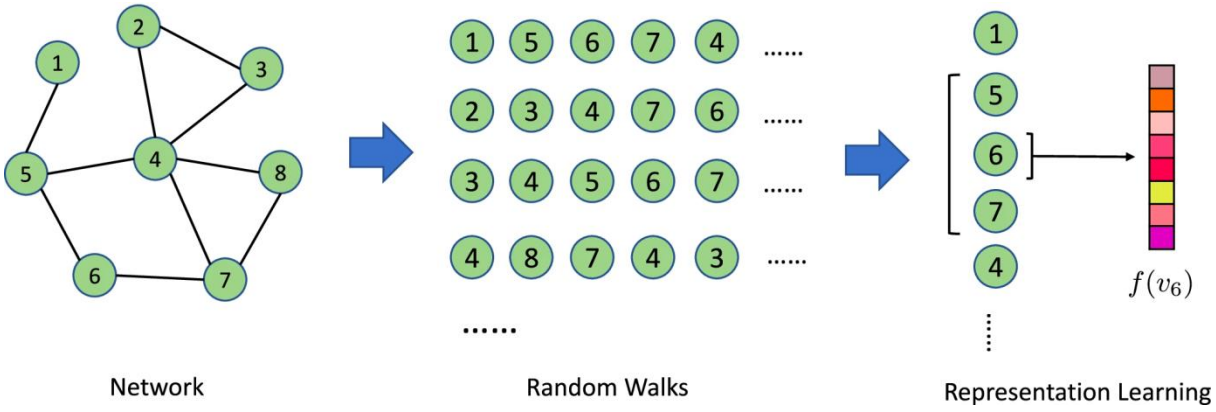


Figure 2.2: Representation of a network embedding algorithm process [6]

- NEURAL NETWORKS

Neural networks (NNs), also known as artificial neural networks, are a class of machine learning models that aim to simulate the behavior of biological neural networks. They are composed of layers of interconnected artificial neurons, which perform computations on the input data.

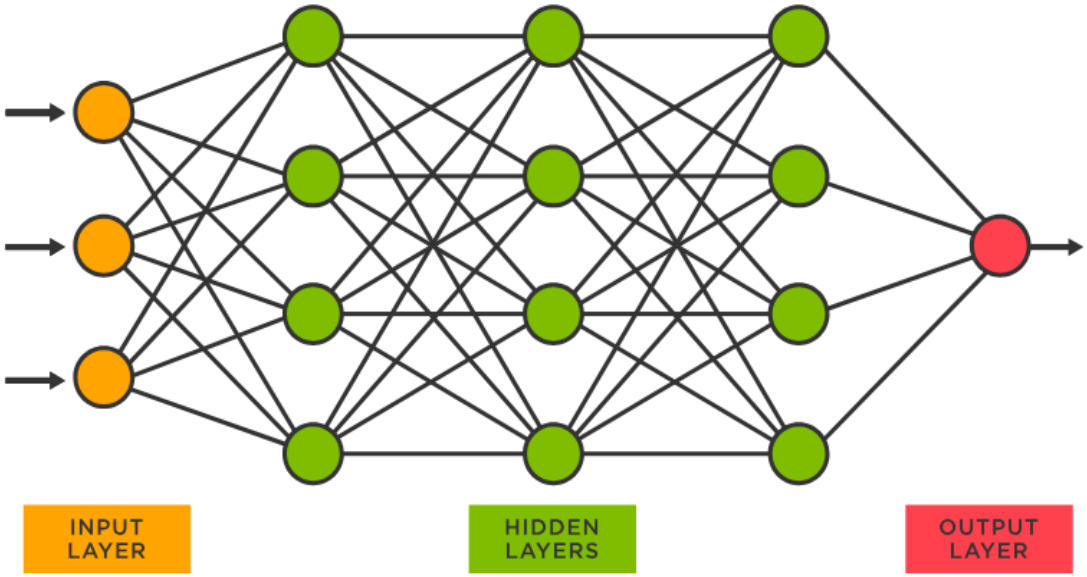
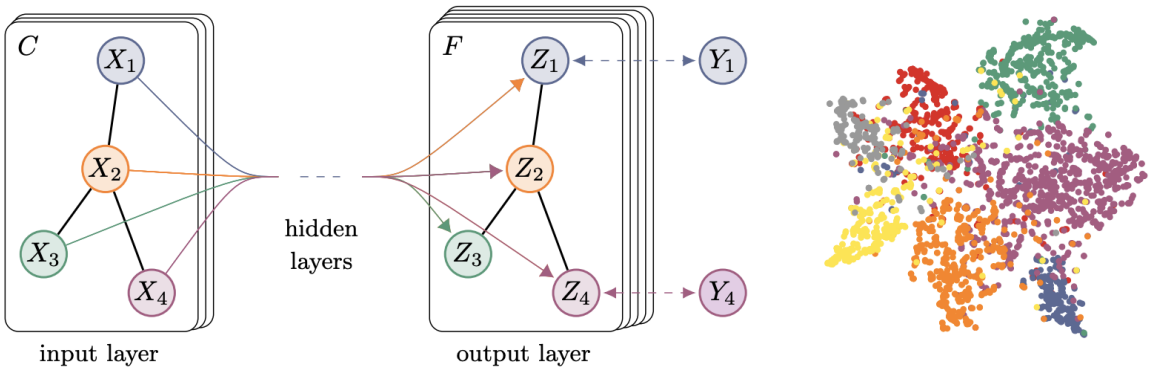


Figure 2.3: Representation of a neural network [7]

In a neural network, the input layer receives the raw data, which, depending on the problem in question, can be images, text, or network representations. The hidden layers, also called intermediate or deep layers, process and transform the input data through a series of mathematical operations. Finally, the output layer produces the desired result, generally predictions or classifications. Figure 2.3 illustrates a neural network composed of 5 layers: an input layer, 3 hidden layers with more hidden features

(explained further in chapter 3.4) than inputs and 1 output layer that processes the information obtained through the hidden layers.

Network embeddings and neural networks are interconnected in the context of graph-based machine learning. Network embeddings can serve as input features for neural networks, allowing them to learn from the structural information encoded in the embeddings or neural networks can be used to develop Neural network architectures, such as Graph Convolutional Networks (GCNs) [8]. GCNs operate directly on graph-structured data, effectively learning node embeddings as part of the model itself. This can be partially seen in the diagrams of Figure 2.4.



(a) Graph Convolutional Network (b) Hidden layer activations

Figure 2.4: GCN representation extracted from GCN article[8]

- NNs IN ARCHITECTURE OF NETWORK EMBEDDING

The architecture of a network embedding using neural networks typically involves two main components: the encoder and the decoder. The encoder network transforms the input network structure into low-dimensional vector representations, while the decoder network reconstructs the network structure from these embeddings. This process allows the model to learn meaningful representations that capture the underlying patterns and relationships within the network.

The encoder network is responsible for capturing the structural information of the network. It takes as input the adjacency matrix, node

features, or other relevant network attributes and processes them through layers of neural units, such as densely connected layers or convolutional layers. These layers learn hierarchical representations of the network by capturing local and global patterns, community structures, and other network properties.

The decoder network, on the other hand, aims to reconstruct the network structure from the learned embeddings. It takes the low-dimensional vector representations produced by the encoder and reconstructs the adjacency matrix or other network attributes. The reconstruction process is typically done using various decoding techniques, such as graph generative models or matrix factorization methods.

To train the network embedding architecture, a loss function is defined to quantify the difference between the original network structure and the reconstructed structure. This loss function guides the learning process, enabling the neural network to optimize the embeddings to capture the most informative and meaningful representations of the network.

One advantage of using neural networks for network embedding is their ability to capture complex nonlinear relationships and high-dimensional patterns in the network data. The deep learning architecture allows the model to learn hierarchical representations, enabling it to encode intricate structural information and capture the nuanced relationships between nodes.

By employing neural networks for the architecture of network embedding, researchers can leverage the power of deep learning to extract rich and expressive representations from complex networks. This approach enables the discovery of hidden patterns, facilitates downstream tasks, and ultimately enhances our understanding of network structures and dynamics.

2.2 Technologies used in the analysis of the road network

The analysis of the road network has been done a lot in recent years for diverse reasons. As such, different technologies have been used to explore the different aspects of the road network.

For instance, GPS applications (such as Google maps) may use a combination of satellite imagery, aerial imagery and a geographic information system (GIS) to create a visual representation of the road network. Additionally, these applications use machine learning with data from various sources (such as other devices locations) in order to find a better route while predicting traffic.

As for the road network analysis, most of them are done towards traffic related problems. For these studies network embeddings are used as they help to restructure the data into a format more easily processed and with more information accessible that the program would overlook such as node neighborhood or node relationships (such as can be seen in the road network when a link-motorway is always connected either to another of the same type or the motorway type), sometimes along with neural networks [9].

2.3 Approaches towards the same problem

As of now, most studies on the road network have focused more on traffic analysis [10], improving routes for the drivers[11] or finding parking lots[12]. As such, the most recent approach found to this problem is described below.

This work foundation is provided by an article on the use of network embeddings on the road networks[1], which verifies the suitability of network embedding for machine learning on road networks by providing promising results using node2vec[13], as well as identifying a major problem in classifying roads of types with low homophily scores. These problems were mainly in correctly identifying nodes of types of little representation (in comparison to others) and of low homophily

(meaning they have more connections to nodes of other types than to the same type).

As such, this project started by attempting a similar approach but using algorithms did not exist at the time. This however showed many challenges on the replication of the experiment, causing the process to diverge a lot while still retaining the same objective. The reasons for these changes were both not having enough practical explanation on certain steps of the article (such as how to deal with the class imbalance) and not having the required resources for a closer reproduction (such as the more specific dataset the authors of the article were provided with).

3. Problem and Proposed Solution

This chapter provides an overview on the problem to be solved along with the resources used (namely, the data) and the approaches that were explored in obtaining the solution. It aims to better orient the reader for the purpose of the project and the steps taken for either a reproduction of the experiments or for an external analysis of the approaches, leading to finding better alternatives that might have not been considered before.

3.1 Problem description

The problem this project approaches is the lack of information about some road features on a road network, and how to predict such information.

The information about road networks is not easily accessible as standard tools such as OpenStreetMap[14], that contains geospatial data, but might not give enough information about roads' variables such as speed limit, surface type and traffic signaling.

In other words, most of the information regarding this network is not documented in open source in great detail, as the sources that are open to the public don't have the corresponding data applied over the general road network. One such source is the well known OpenStreetMap[14] (see Figure 3.1).

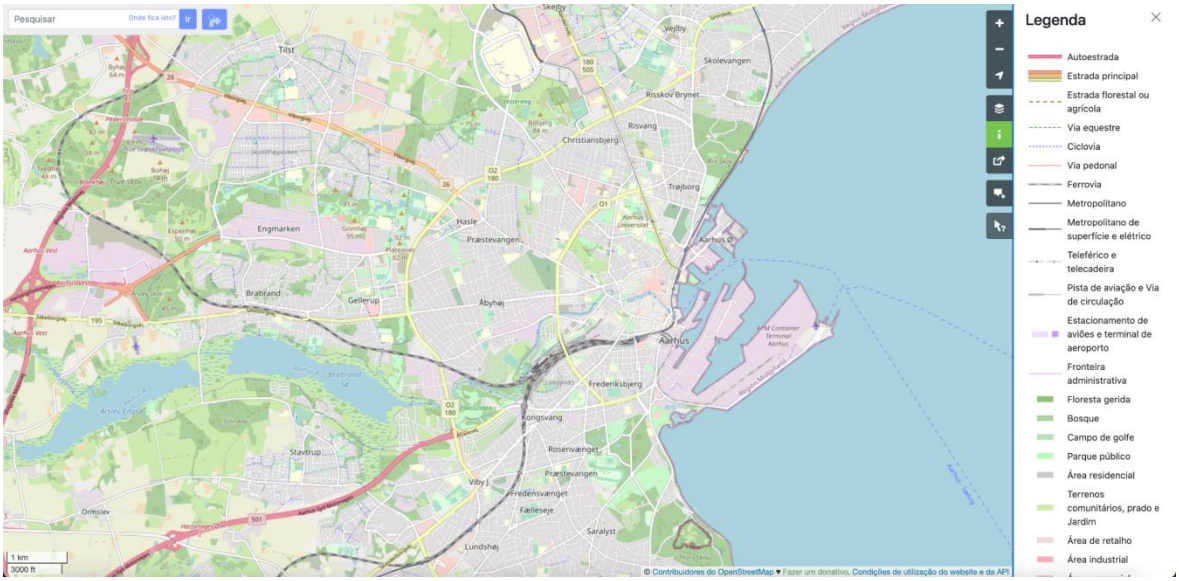


Figure 3.1: Image in OpenStreetMap. Only part of the roads have their type designated (highway, residential, secondary,...) and some road signs are present in the downloadable file

In this work, the problem to be treated will be restricted to the lack of road classification (type of road), since it is the only road feature that was found documented well enough.

Since the problem to be solved involves the prediction of missing information in the features road features that we gain access to, different network embedding methods are used to classify the roads with 20% of them being used for testing, and a classification of the road types (residential, primary, service, etc) with statistical metrics independent of the class imbalance is computed.

3.2 Data extraction

In an attempt to gather information on the road network of a country, an attempt to contact “Ascendi”, a company responsible for operating and maintaining road infrastructures in Portugal. However, despite their detailed data, the information they had at their disposal was mostly regarding the highways. Using this data would turn into a problem as it would create bias in the algorithms as most of the road attributes would be registered just in the roads of type ‘motorway’, causing the missing information on the other road types to be filled according to the only available ‘motorway’.

As such, from OpenStreetMap [14], on 17th of May, was extracted a ".osm.pbf" file containing a geospatial data of Denmark, including information regarding road connections and road types, which are used in this work. The choice of country to study on was based on the previous paper [1] that this project attempts to explore in a deeper and more practical manner since other road characteristics were not obtained.

3.3 Preprocessing and initial data analysis

The data used for this work was obtained from the file mentioned above and contains road network information on Denmark overall. The file extracted, in format ".osm.pbf", contains a lot of geospatial information, of which will be used the road network and, more specifically, road connections and road categories (residential, primary, motorway, etc).

The ".osm.pbf" file format was incompatible with the python libraries used for this work and had to be converted using "osmium" [15] to xml. Additionally, during the conversion, only the road network data was converted into the new file and many of the roads removed due to having types that could be used by the common vehicle, these being:

- raceway
- footway
- bridleway
- steps
- corridor
- cycleway
- proposed
- construction
- path
- stairs
- traffic calming
- bus_stop
- elevator
- disused
- services (different from ‘service’ as this one refers to the stopping places found in the middle of the highways)
- bus_guideway
- rest area
- platform
- planned
- busway
- pedestrian

Afterwards, the data was stored in a NetworkX [16] undirected graph, since the direction of the roads is only present by interpreting the road signs (which can’t be done solely with OpenStreetMap as there is neither the sign orientation nor other specific traffic sign characteristics).

In this format, roads are represented as edges with nodes either representing road connections, such as a crossway, or segments in the road with some traffic sign (such as a crosswalk or a stop sign). A function was used to identify isolated nodes and remove them, resulting in a graph of 4.336.566 nodes and 4.525.797 edges as illustrated in Figure 3.2.

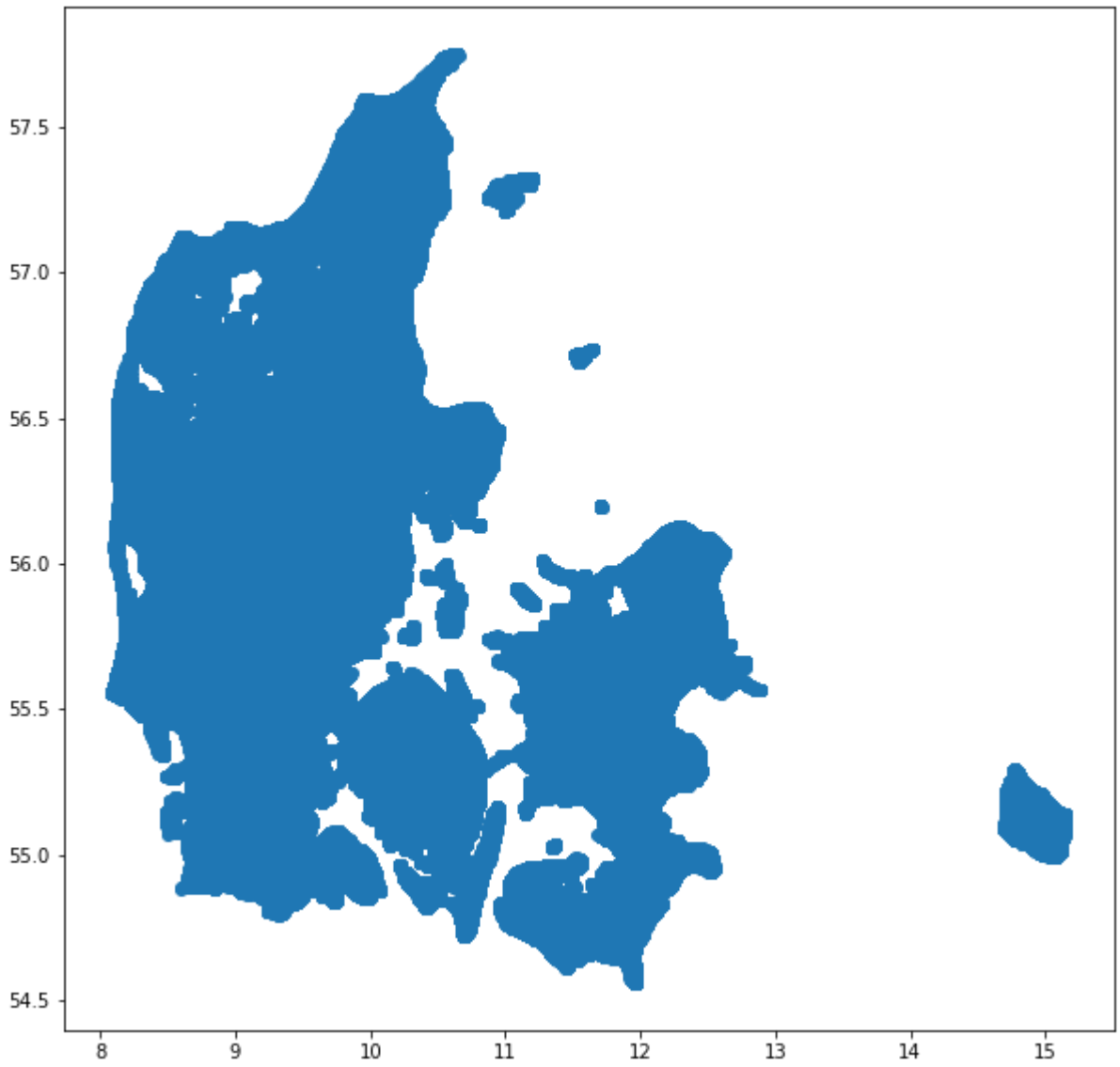


Figure 3.2: Visualization of the graph of entire road network of Denmark

As further research was made on the algorithms and existing graph related methods implemented in python, it was revealed that most functions were developed to study the nodes' characteristics. A function that converts the extracted graph into a line graph [17] was developed to make use of already implemented functions and libraries. A line graph is a type of graph obtained by converting the edges of a pre-existing one in the nodes of a new one. For each node of the original graph, all combinations of edges to enter and leave the node are represented as nodes in the line graph. This conversion changed the graph to contain 4.525.797 nodes and 5.633.593 edges.

A comparison between the distribution of road types in the original graph edges and the line graph was made to assure that the road numbers for each class had not changed, as presented in Table 3.1.

Road Type	**Number of roads (total = 4 525 797)**	**Percentage in the network**
service	1613032	35.64
residential	875367	19.34
track	719258	15.89
unclassified	580822	12.83
tertiary	500087	11.05
secondary	115459	2.55
primary	47151	1.04
motorway	23260	0.51
motorway_link	17950	0.40
living_street	11890	0.26
trunk	6157	0.14
secondary_link	5034	0.11
tertiary_link	4123	0.09
primary_link	3364	0.07
trunk_link	2843	0.06

Table 3.1: Distribution of road types/node classes in the entire network of Denmark

This database that will be called DataAll, contains the entire road network in Denmark,which proved to be too heavy for the GATv2 algorithm using standard workspace hardware, the algorithm used in the second approach to the network embedding task (this algorithm will be described later in section 3.4.2). This graph will be used to assess the potential of this approach with the GraphSage algorithm only, the first algorithm tested for the network embedding task in this project.

In order to build another database to be used with the two developed approaches, three spatial blocks with data were extracted regarding the roads,

containing both cities used in paper [1] (Aalborg and Copenhagen). It is illustrated in Figure 3.6 with the geographic distances. However, despite from first glance the dataset thus obtained containing the cities of Aalborg (illustrated in Figure 3.3) and the Capital Region of Denmark (illustrated in Figures 3.4 and 3.5), it proved to be smaller than the one used in [1] as can be seen in the total number of roads in Table 3.2 (for reference, in paper [1], the number of roads used was close to 1.3 million).

Using this new dataset that will be called DataAC, the results of both approaches are compared, GraphSage and GATv2, and we assess if, with computing power remaining the same, the GATv2 model could perform better.

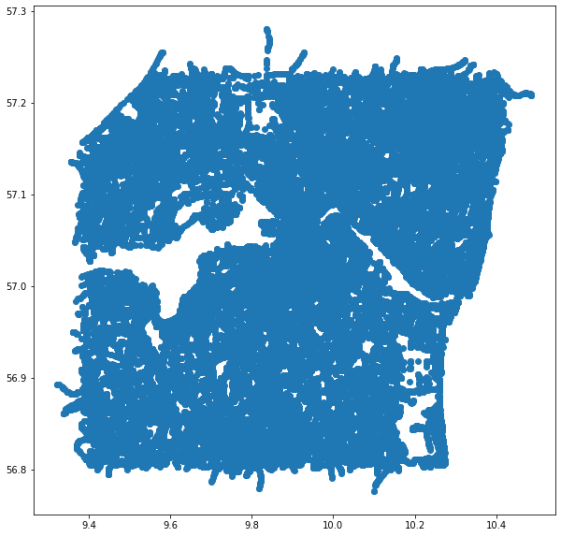


Figure 3.3: Graph of Aalborg

Figure 3.4: Graph of Capital Region of Denmark containing Copenhagen

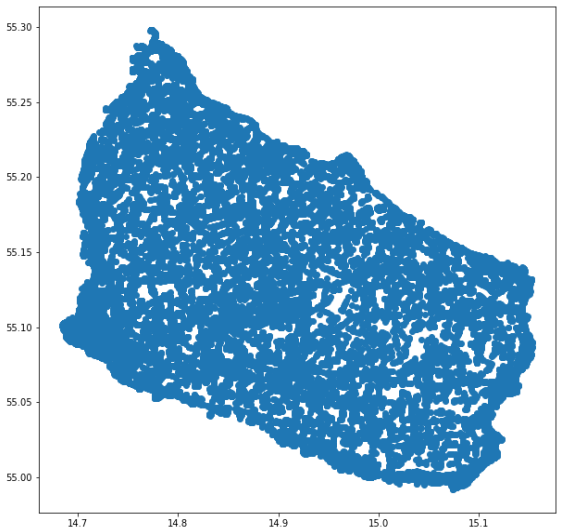


Figure 3.5: Graph of the capital Region of Denmark containing Bornholm

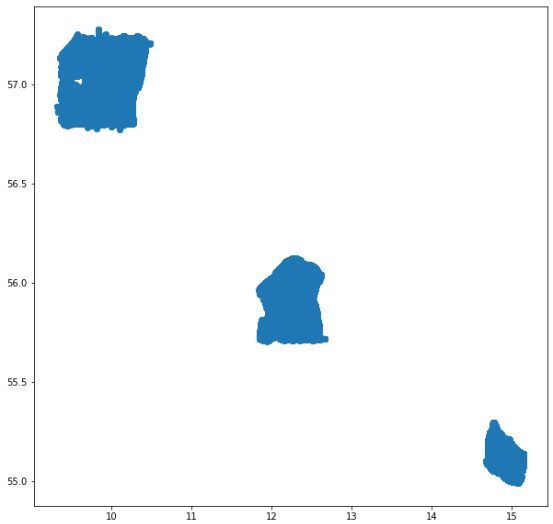


Figure 3.6: Representation of entire sub-graph in scale

Road Type	Number of roads (total = 663 084)	Percentage in the network
service	235506	35.51
residential	164291	24.78
track	91010	13.72
unclassified	58282	8.79
tertiary	76403	11.52
secondary	19657	2.96
primary	5785	0.87
motorway	3179	0.48
motorway_link	2978	0.45
living_street	1908	0.29
trunk	1092	0.16
secondary_link	869	0.13
tertiary_link	896	0.14
primary_link	506	0.08
trunk_link	722	0.11

Table 3.2: Distribution of roads by types in DataAC, using same type order

3.4 Data processing and node classification

The graph was transferred to a DGL [4] graph as this library has consistent updates, good support for graph analysis methods, and backed by a fair amount of stars in Github (11,9 k) at the time of this writing. Since the library does not have a direct representation of undirected graphs, this conversion doubled the graph edges to represent both directions. Additionally, the graph does not support string type attribute values, so to add the road type, a conversion had to be made, as illustrated in Algorithm 3.1.

```python
# Get the 'highway' attribute values from the networkx graph
attributes = nx.get_node_attributes(L, 'highway')

# Convert the attribute values to categorical numerical labels
label_encoder = LabelEncoder()
encoded_values = label_encoder.fit_transform(list(attributes.values()))

# Assign the encoded attribute values to the nodes in the DGL graph
dgl_graph.ndata['highway'] = torch.tensor(encoded_values, dtype=torch.long)
```

Listing 3.1: Python code to convert the values of the attribute 'highway' to a numerical value

In the paper on which the work was initially based, to deal with the unbalanced dataset (Table 3.2), it is mentioned that over-sampling was used. However since its implementation is not explained in full detail, we opted to use both stratified sampling (which is illustrated in [Figure 3.7]) and adding weight to classes inversely proportional to their representation. The following procedure was adopted:

- Stratified sampling allows a representative percentage of all road types to be selected, both to train the algorithm and to test it (with the corresponding percentages being 80% and 20% respectively).

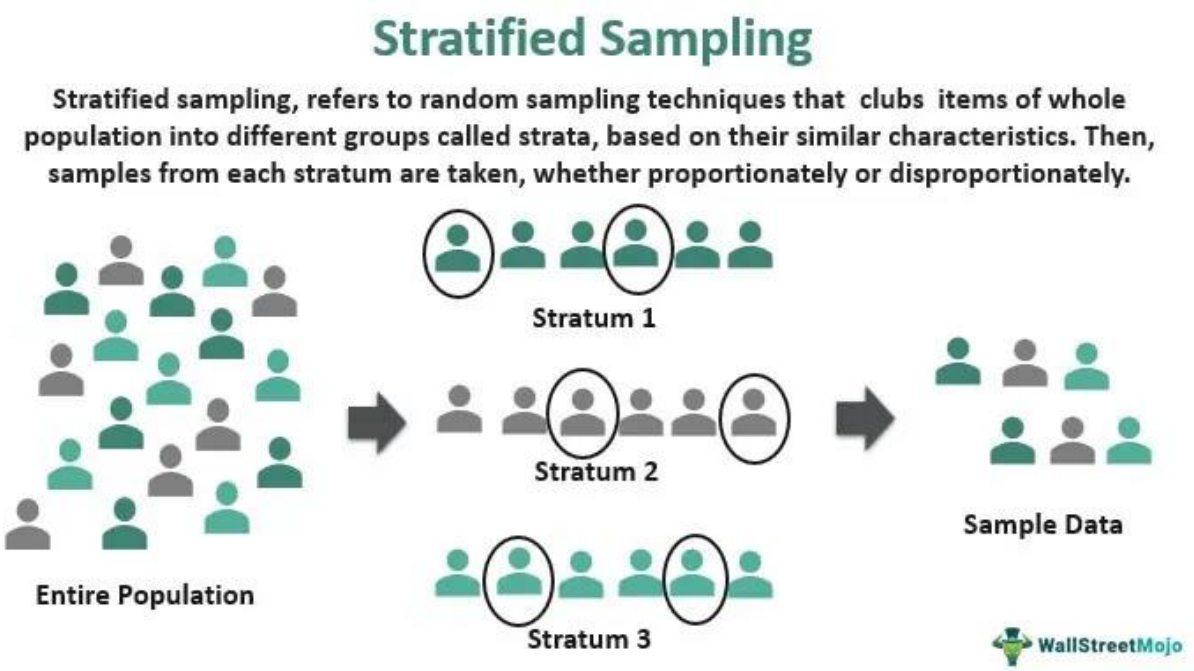


Figure 3.7: Brief explanation with visual representation of stratified sampling [18]
In this work the stratums are made using the road type attribute values

- Weighting classes inversely to their representation affects the loss calculating function that is optimized at each training epoch, by giving weight to the classes as described, incorrectly classifying a node results in a greater loss value, forcing the embedding and classification algorithms into a path that results in a more balanced classification without leading towards bias to

neither overly represented classes, such as ‘service’ and ‘residential’, nor under-represented, such as the links.

Afterwards, two network embedding models had to be implemented. The first one chosen is GraphSage [2] that can be applied to large datasets with its scalability. The second is GATv2 [3], which in theory is more computationally heavy in exchange for a more detailed observation of different patterns that can be found in a graph. These two methods will be presented in detail later on.

With both approaches implemented, the algorithms are run in a loop of at least 180 epochs. This loop consists of calling train (see Algorithm 3.2) and test (see Algorithm 3.3) functions implemented, while recording the results every 20 epochs.

In the training function, described in Algorithm 3.2, the arguments sent have the following purposes: *‘model’* refers to the model doing the embedding and classification sent to the function, *‘g’* is the graph, *‘features’* a tensor with the features of the graph, *’labels*’ is another tensor (which is similar to features as both label and feature are the same in this case), *‘train_mask’* is a tensor that for each node contains a boolean value to determine if it is computed, optimizer contains the algorithm used to optimize parameters (different from the hyperparameters), *‘loss_fn’* contains the function used for the calculation of the loss in each step.

```
def train(model, g, features, labels, train_mask, optimizer, loss_fn):
#sets the model in training mode.
    model.train()
#clears the gradients of all optimized parameters so the optimization is according
to new gradients calculated
    optimizer.zero_grad()
#does classification in the train dataset according to graph g and the features
    logits = model(g, features)
#calculates loss using the weights and the difference in the results when compared
with original values
    loss = loss_fn(logits[train_mask], labels[train_mask])
#calculates the new gradients from which the algorithm is optimized
    loss.backward()
#optimizes parameters in the algorithm
    optimizer.step()
```

Listing 3.2: Python implementation of train function

```
def test(model, g, features, labels, test_mask):
#sets the model in evaluation mode.
  model.eval()
#ensures that no gradients are computed during the evaluation. Since in this
step there is no change in the optimizer, it saves memory and speeds up the
evaluation process
   with torch.no_grad():
#does classification in the test dataset according to graph g and the features
    logits = model(g, features)
    _, predicted = torch.max(logits, 1)
   predicted = predicted[test_mask].cpu().numpy()
   labels = labels[test_mask].cpu().numpy()
#the rest of the function calculates the f1_scores for each class/road type
#     …
```

Listing 3.3: Python implementation of test function

3.4.1 GraphSage

The GraphSage model used in this work, and illustrated in Figure 3.8, is composed of two layers: a convolutional layer, implemented using the DGL and responsible for creating the network embeddings; and a classification layer, implemented using *torch.nn* library and responsible for attributing the classes to the nodes received. The idea is that the convolutional layer creates groups/hidden features based on patterns of the nodes, assigning the nodes to their corresponding features, and then the classification layer, based on the hidden features found in the nodes, classifies each node to the corresponding road type.

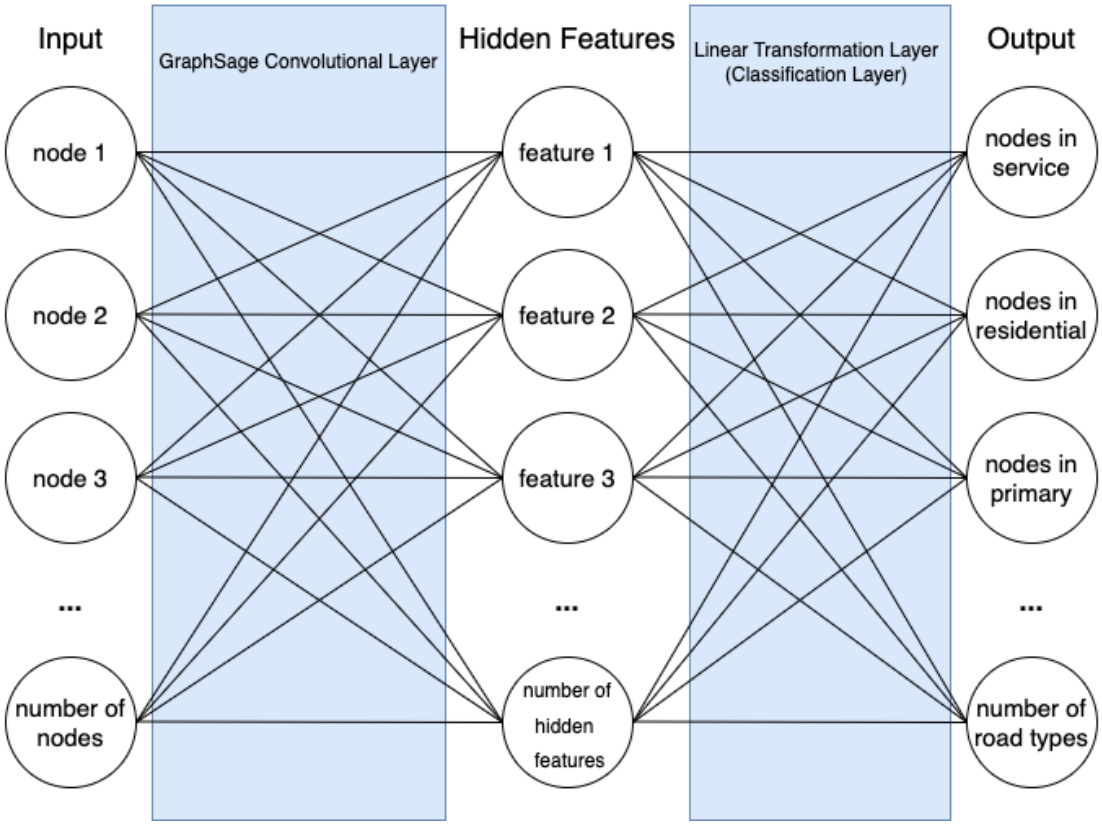


Figure 3.8: Diagram representing the GraphSage approach model used in this work

The convolutional layer serves the purpose of aggregating information from a node's local neighborhood and updating the node representations. It enables the model to learn meaningful and informative node representations that capture both the node's features and the structure of the graph by creating the appropriate network embeddings for the algorithm used. The name of the approach comes from the fact that the class created to classify the road types uses the GraphSage as its convolutional layer.

The classification layer is made using the *torch.nn.Linear* algorithm. As the name suggests, it takes as input the embeddings of the convolutional layer and, through a linear transformation, assigns values to the nodes, corresponding the possible road types. These two layers can be observed in the class implemented in Algorithm 3.4.

```
class GraphSageClassifier(nn.Module):
    def __init__(self, in_feats, hidden_feats, num_classes):
        super(GraphSageClassifier, self).__init__()
        self.sage = SAGEConv(in_feats, hidden_feats, 'mean')
        self.fc = nn.Linear(hidden_feats, num_classes)

    def forward(self, g, features):
        x = self.sage(g, features)
        x = self.fc(x)
        return F.log_softmax(x, dim=1)
```

Listing 3.4: Python implementation of GraphSage model

Other hyperparameters are represented in Algorithm 3.5, with emphasis on the fact that the parameter *hidden_features* was increased to 250 for the smaller graph.

```
#number of nodes in the graph
num_nodes = dgl_graph.number_of_nodes()
# number of features. 'highway' is both the label and the only feature in the graph
num_features = 1
# size of output from the convolutional layer. This means that from the many nodes that are inserted, the embeddings separate them in 200 different groups, which are analyzed by the classification layer and re-assigned to the different classes
hidden_features = 200
# number of unique values for the attribute 'highway'
num_classes = len(np.unique(dgl_graph.ndata['highway']))
#number of cycles for training and testing
num_epochs = 180
#learning rate of the algorithm
lr = 0.001
```

Listing 3.5:Python declaration of some hyperparameters used in GraphSage model

The model parameters are optimized at each iteration using *torch.optim.Adam()* which is an implementation of Adam optimization algorithm. The choice was made considering the versatility and how many situations the algorithm can be effectively applied. This means that although it does the job correctly, there may be other more specific algorithms that can do better.

3.4.2 GATv2

The second approach shares a similar structure with the previous one but has some small adjustments necessary for the network embedding algorithm used.

The convolutional layer uses GATv2 as the algorithm. GATv2 came in 2022 as an upgrade on many areas from GAT [19]. The major problem on the computations of this algorithm is the computation power required to run it. When attempting to use it in the entire network of Denmark, the result was a process crash as it took at least 50GB of memory RAM in the computer before failing. After some tests we found out it was due to how it processes the node connections, as processing a network with many fewer edges leads to actual results. This implementation can be seen in Algorithm 3.6.

```
class GATv2Classifier(nn.Module):
    def __init__(self, in_feats, hidden_feats, num_classes,num_heads):
        super(GATv2Classifier, self).__init__()
        self.conv1 = GATv2Conv(in_feats, hidden_feats, num_heads)
        self.fc = nn.Linear(hidden_feats*num_heads, num_classes)

    def forward(self, g, features):
        x = self.conv1(g, features)
        x = torch.reshape(x, (x.size(0), -1))
        x = self.fc(x)

        return F.log_softmax(x, dim=1)
```

Listing 3.6: Python implementation of Gatv2 model

Another change is in the hyperparameters. One of the big changes from the original version that GATv2 brings is the use of multiple attention heads. Attention heads refer to parallel attention mechanisms that operate on the input graph data simultaneously. In this work 5 heads were used and the number of hidden_features reduced to 100.The reason for the numbers was due to tests made using just one of the 3 graphs at the time showed both better results and run time than by using more hidden_features (200) and less heads (3) as can be seen in Table 3.3 and Figures 3.9 and 3.11, which show

F-scores using 200 hidden features and 3 heads,and Table 3.4 and Figures 3.10 and 3.12 which show using 100 hidden features and 5 heads. Both of these results were obtained by testing the model on just Aalborg (Figure 3.3).

Figures 3.9 and 3.10 use the macro average for the F-Scores presented. This is the regular calculation of the average, summing the values of each column and dividing by the number of classes, being that the calculation is done following the formula in Listing 3.8. As this disregards the class imbalance, it allows us to better perceive the growth of performance over all classes every 20 iterations of training and testing.

F-Scores through the project were calculated using the code in Listing 3.7, which would be the continuation of the test function previously defined in Listing 3.3.

```
unique_values = np.unique(g.ndata['highway'])

f1_scores = {}

for value in unique_values:

    tp = np.sum(np.logical_and(predicted == value, labels == value))
    fp = np.sum(np.logical_and(predicted == value, labels != value))
    fn = np.sum(np.logical_and(predicted != value, labels == value))
    tn = np.sum(np.logical_and(predicted != value, labels != value))

    if(tp!=0):
        precision = tp / (tp + fp)
        recall = tp / (tp + fn)
        f1_score = 2 * (precision * recall) / (precision + recall)

        f1_scores[value] = f1_score
    else:
        f1_scores[value] = 0

return f1_scores
```

Listing 3.7: Python Calculation of F-Scores for each road type

$$\text{Macro F1 Score} = \frac{\sum_{i=1}^{n} \text{F1 Score}_i}{n}$$

Listing 3.8: macro average f-score formula [20]

1 283 × 217

Figures 3.11 and 3.12 use the weighted average for the F-Scores displayed. This calculation is done by multiplying the F-Scores of each class/road type with the corresponding frequency in the graph as the weight (for instance, using Table 3.2, multiplying service F-scores by 0.3551) and afterwards sum the values of each column, following the formulas seen in Listing 3.9. As such, this average allows us to observe the growth of performance by nodes correctly classified every 20 iterations, disregarding the different classes.

$$\text{Weighted F1 Score} = \sum_{i=1}^{N} w_i \times \text{F1 Score}_i$$

$$w_i = \frac{\text{No. of samples in class } i}{\text{Total number of samples}}$$

Listing 3.9: weighted average f-score formula and weight calculation [21]

road type	number of iterations								
	20	40	60	80	100	120	140	160	180
living_street	0.000	0.000	0.000	0.099	0.149	0.340	0.358	0.445	0.456
motorway	0.048	0.170	0.426	0.506	0.497	0.434	0.372	0.311	0.162
motorway_link	0.000	0.104	0.019	0.027	0.045	0.301	0.546	0.611	0.623
primary	0.000	0.000	0.000	0.022	0.051	0.147	0.146	0.198	0.263
primary_link	0.000	0.006	0.043	0.005	0.005	0.006	0.005	0.005	0.006
residential	0.000	0.000	0.000	0.000	0.006	0.513	0.131	0.180	0.298
secondary	0.000	0.000	0.002	0.042	0.237	0.345	0.200	0.101	0.102
secondary_link	0.000	0.000	0.001	0.022	0.025	0.077	0.068	0.071	0.060
service	0.000	0.000	0.000	0.000	0.584	0.696	0.682	0.660	0.679
tertiary	0.000	0.000	0.000	0.000	0.000	0.514	0.247	0.243	0.223
tertiary_link	0.013	0.000	0.000	0.000	0.001	0.002	0.007	0.012	0.010
track	0.000	0.274	0.270	0.000	0.079	0.152	0.614	0.615	0.733
trunk	0.001	0.001	0.001	0.001	0.000	0.012	0.007	0.008	0.015
trunk_link	0.002	0.003	0.003	0.008	0.013	0.021	0.026	0.027	0.021
unclassified	0.334	0.308	0.282	0.270	0.578	0.530	0.540	0.538	0.545

Table 3.3: F-Scores recorded every 20 iterations with 200 hidden-features and 3 heads

road type	number of iterations								
	20	40	60	80	100	120	140	160	180
living_street	0.651	0.337	0.374	0.454	0.544	0.478	0.545	0.638	0.670
motorway	0.000	0.425	0.316	0.000	0.325	0.252	0.324	0.429	0.483
motorway_link	0.000	0.060	0.126	0.170	0.148	0.241	0.378	0.508	0.633
primary	0.000	0.019	0.174	0.084	0.064	0.077	0.074	0.401	0.448
primary_link	0.000	0.000	0.000	0.000	0.000	0.180	0.305	0.033	0.034
residential	0.000	0.000	0.006	0.689	0.690	0.756	0.773	0.783	0.778
secondary	0.000	0.000	0.005	0.101	0.005	0.503	0.525	0.566	0.504
secondary_link	0.000	0.000	0.000	0.002	0.036	0.037	0.030	0.012	0.014
service	0.000	0.000	0.066	0.002	0.615	0.614	0.600	0.592	0.616
tertiary	0.000	0.005	0.000	0.000	0.078	0.078	0.090	0.102	0.115
tertiary_link	0.000	0.000	0.000	0.000	0.000	0.006	0.005	0.011	0.011
track	0.000	0.226	0.256	0.264	0.536	0.571	0.569	0.631	0.706
trunk	0.000	0.000	0.000	0.000	0.000	0.000	0.034	0.048	0.081
trunk_link	0.003	0.003	0.002	0.001	0.002	0.004	0.006	0.010	0.041
unclassified	0.178	0.207	0.262	0.388	0.460	0.510	0.557	0.689	0.584

Table 3.4: F-Scores recorded every 20 iterations with 100 hidden-features and 5 heads

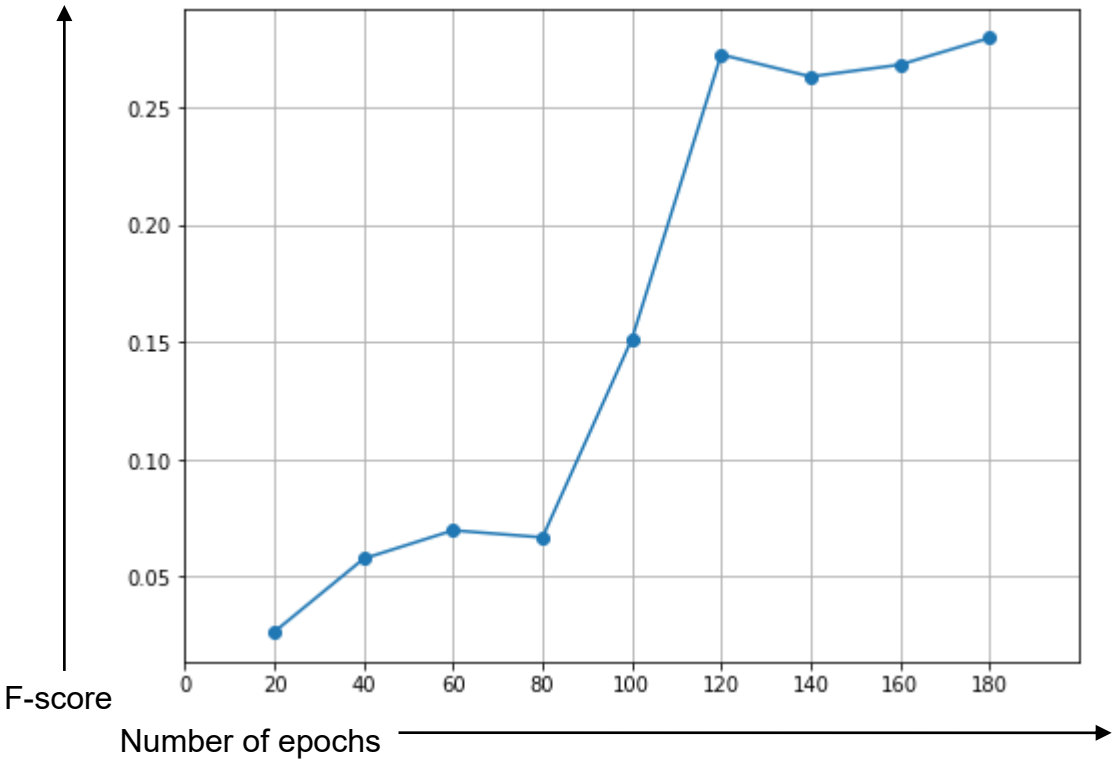


Figure 3.9: macro average f-scores using model with 200 hidden-features and 3 heads

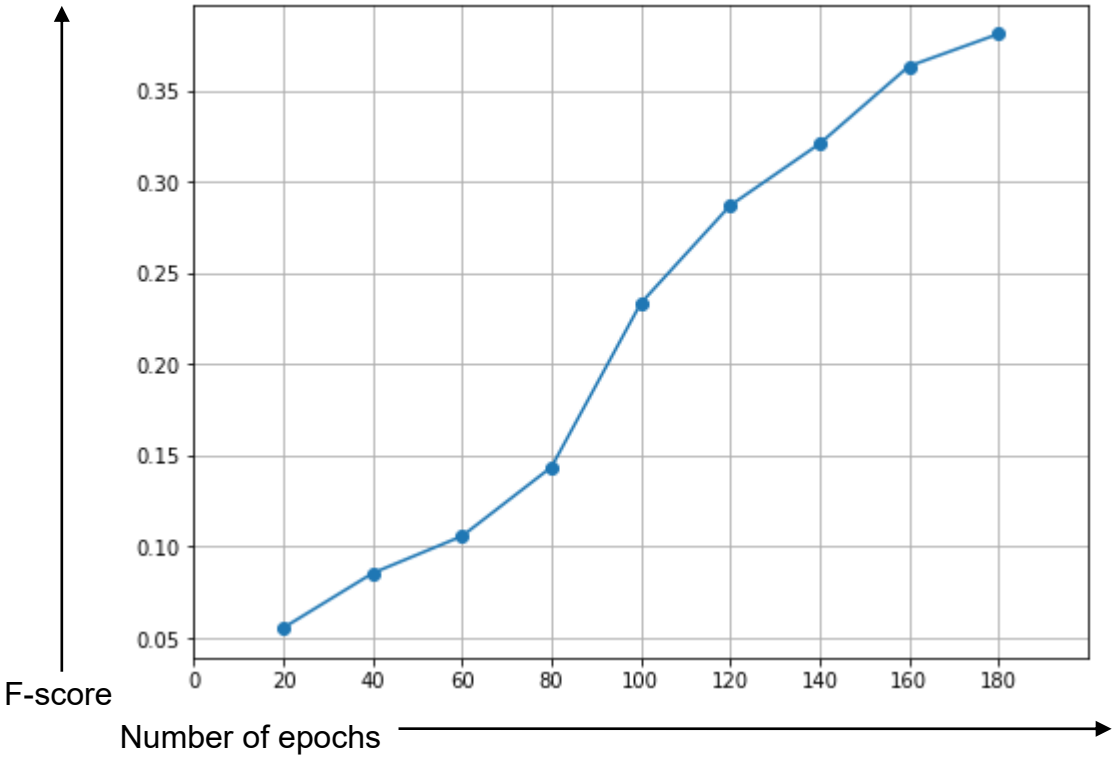


Figure 3.10: macro average f-scores using model with 100 hidden-features and 5 heads

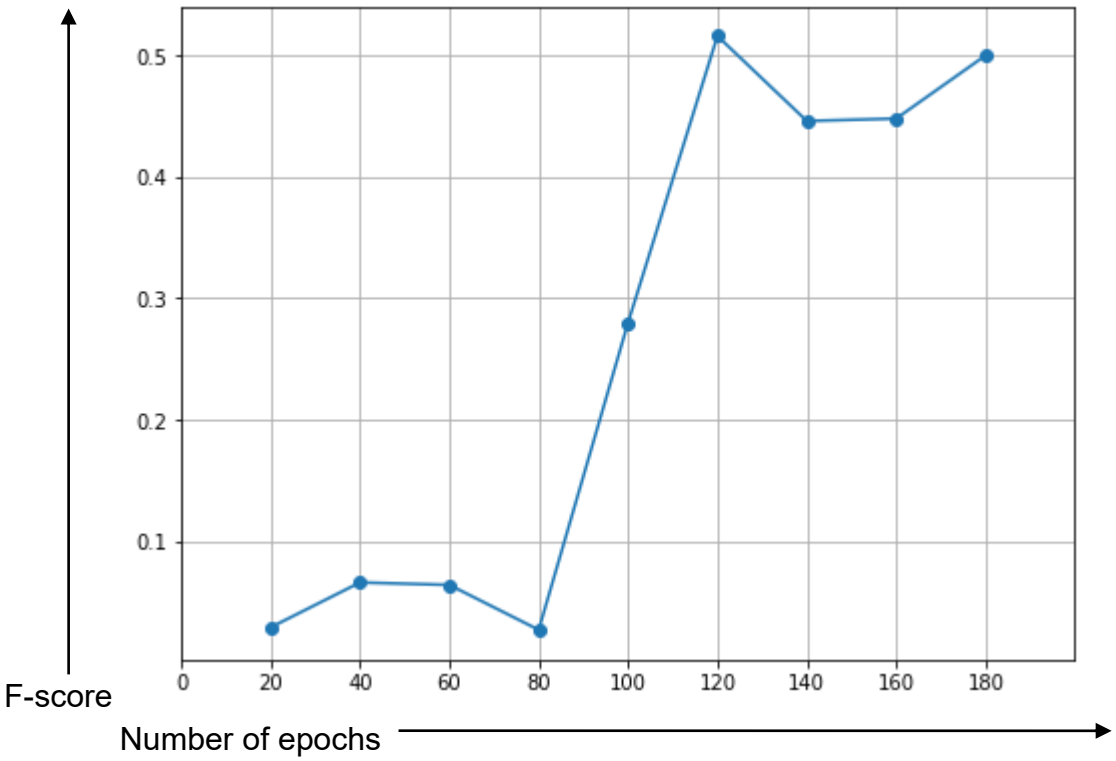


Figure 3.11: weighted average f-scores using model with 200 hidden-features and 3 heads

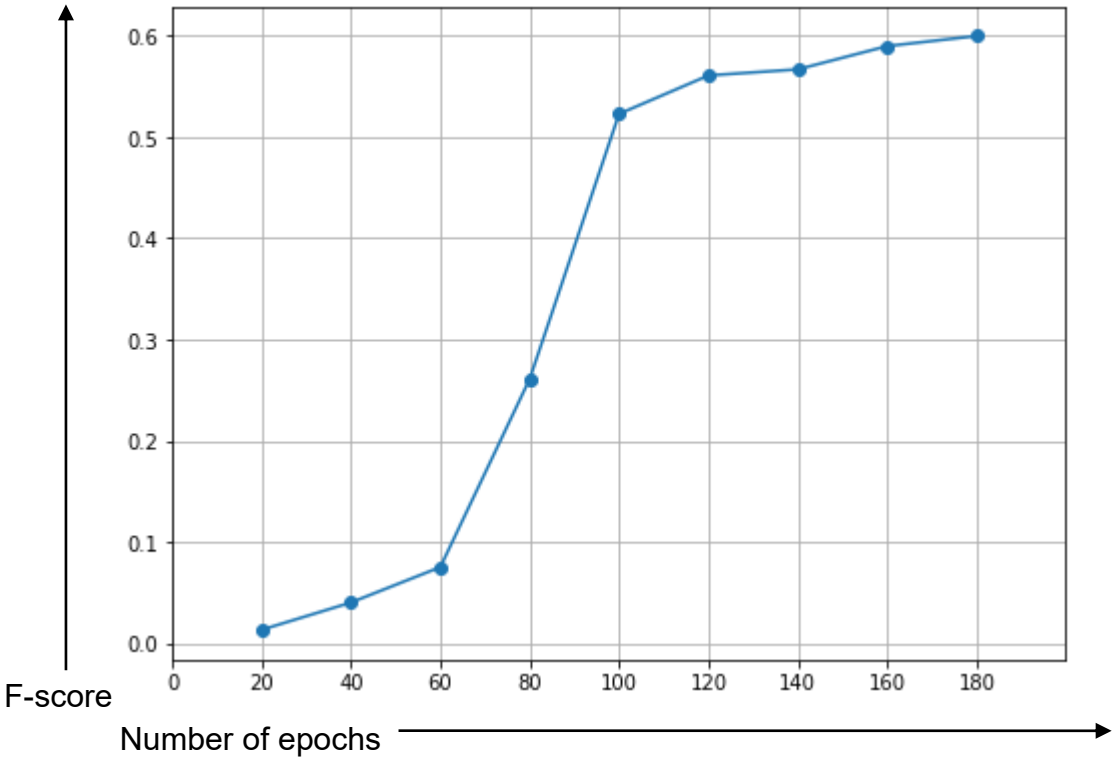


Figure 3.12: weighted average f-scores using model with 100 hidden-features and 5 heads

By using multiple attention heads, GAT can capture different patterns and relationships within the graph, as each head attends to a different set of features and dependencies. The outputs from the multiple attention heads are typically concatenated or averaged to create a final node representation. As

such, due to the output being 3-dimensional (input, heads, output), a transformation has to occur between the neural network layers. This transformation is done using *torch.reshape()* to join the 2 last dimensions of the tensor into a single one (which can then be processed by the linear layer as the output of the convolutional layer) as can be seen in Algorithm 3.6.

4. Result Analysis

This chapter provides some results obtained through running the previous tests, along with a brief overview on what each test was and an explanation on what observations can be taken from them.

As such, this section displays the F-Scores taken after doing the classification of the roads in both the complete road network of Denmark (DataAll) and a subsection of it (DataAC). For the dataset DataAll, the complete road network, it only shows and analyzes the results obtained by the GraphSage model as the GATv2 model showed problems (which will be further discussed ahead) when being run in such a vast network. For the dataset DataAC, this chapter presents and analyzes results that allows us to compare both models implemented (GraphSage and GATv2) and further observe their corresponding performances.

4.1 Results from DataAll

In this subsection, an analysis is made on the performance of two different node classification algorithms, GraphSage and GATv2, applied the entire network of Denmark, which can be visualized in Figure 3.2. Notably, GraphSage demonstrated successful results, efficiently producing outcomes without any significant computational issues. However, the GATv2 algorithm encountered substantial challenges during testing.

Despite some hyperparameter tuning attempts, GATv2 consistently crashed on the hardware used for this project, hindering its successful application. Reducing the number of processed values by selecting fewer nodes through the mask yielded no improvement. Thus, GATv2 does not present Tables or Figures illustrating results as this was not feasible with my current equipment. Further exploration, such as dividing the network into multiple graphs, was considered, but this approach would result in the exclusion of crucial road connections, making it impractical for this specific test. To enhance GATv2's potential for node classification in vast areas like Denmark, future research may focus on refining the architecture, exploring

alternative hyperparameter configurations, or acquiring additional computing resources to address its limitations effectively.

As for GraphSage, the algorithm showed its scalability and proved capable of categorizing very well classes of high representation (such as ‘residential’ with 19.34% frequency and ‘service’ with 35.64%, values taken from Table 3.1) and some under-represented classes (such as ‘living_street’ with 0.26% frequency in and ‘motorway’ with 0.51%, values taken from Table 3.1) as can be seen by the F-Scores in the last iteration/epoch (180) of the Table 4.1. Still, some classes that should have better results for their frequency (such as ‘tertiary’ and ‘track’) showed lower than expected values, this either suggests that the nodes selected by the seeds used for separation of train and test sets were not very adequate for the algorithm or the architecture of the model needs further tweaking to find more patterns (especially considering that one class had a abysmal F-score through all iterations).

Even then, when analyzing the line chart in Figure 4.1, we can see that the algorithm still was growing in performance considerably even in its last iteration. Through Figure 4.2 we can conclude that, as the curvature was lower, the classes that were improving in the last 20 iterations were in fact the less represented ones. Still, more iterations and further tests have to be done to take further conclusions.

road type	number of iterations								
	20	40	60	80	100	120	140	160	180
living_street	0.403	0.518	0.583	0.584	0.661	0.705	0.724	0.735	0.839
motorway	0.605	0.599	0.613	0.613	0.651	0.706	0.734	0.815	0.891
motorway_link	0.000	0.000	0.000	0.401	0.554	0.648	0.707	0.915	0.949
primary	0.437	0.392	0.403	0.471	0.486	0.595	0.606	0.665	0.695
primary_link	0.008	0.015	0.215	0.235	0.236	0.241	0.255	0.271	0.284
residential	0.385	0.430	0.731	0.754	0.760	0.781	0.975	0.979	0.981
secondary	0.000	0.262	0.324	0.384	0.376	0.383	0.546	0.550	0.552
secondary_link	0.000	0.015	0.017	0.020	0.022	0.022	0.110	0.111	0.113
service	0.000	0.268	0.378	0.782	0.787	0.784	0.828	0.828	0.829
tertiary	0.000	0.000	0.107	0.044	0.477	0.503	0.451	0.451	0.448
tertiary_link	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.007	0.012
track	0.086	0.320	0.587	0.593	0.617	0.694	0.693	0.694	0.694
trunk	0.000	0.000	0.063	0.000	0.000	0.000	0.081	0.102	0.226
trunk_link	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
unclassified	0.311	0.463	0.654	0.731	0.823	0.986	0.995	0.995	0.995

Table 4.1: F-Score measures for every road type of GraphSage model approach on DataAll

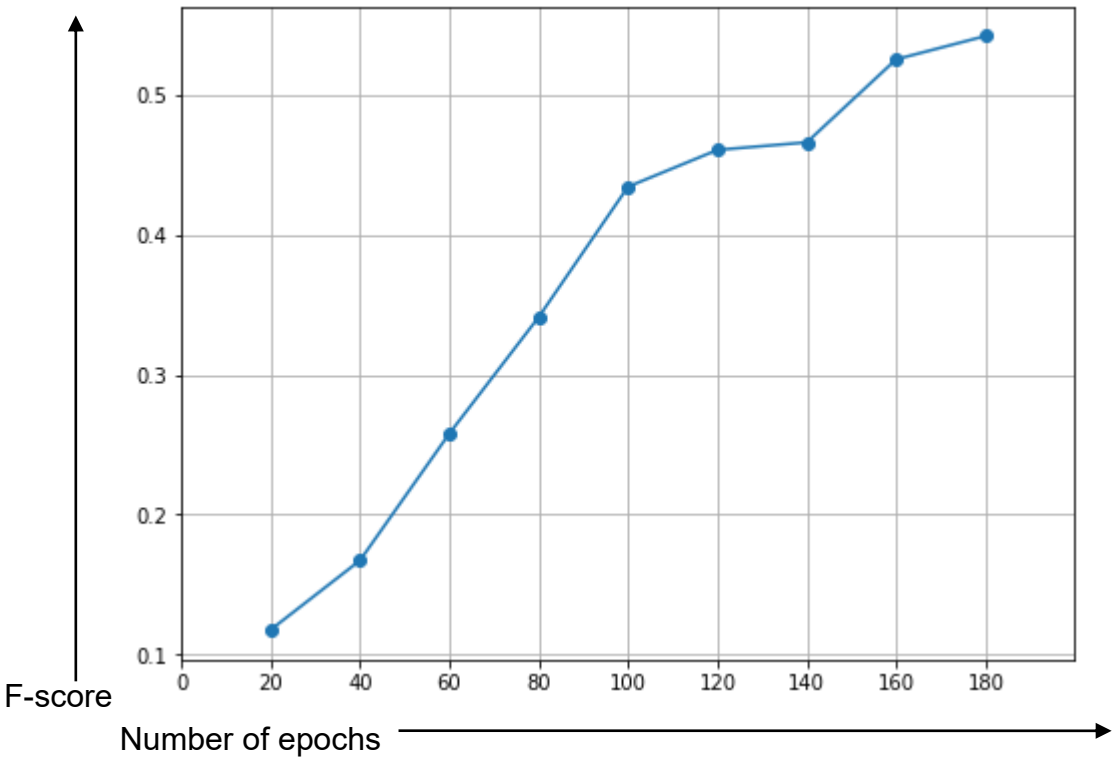


Figure 4.1: macro average F-score of GraphSage model approach on DataAll

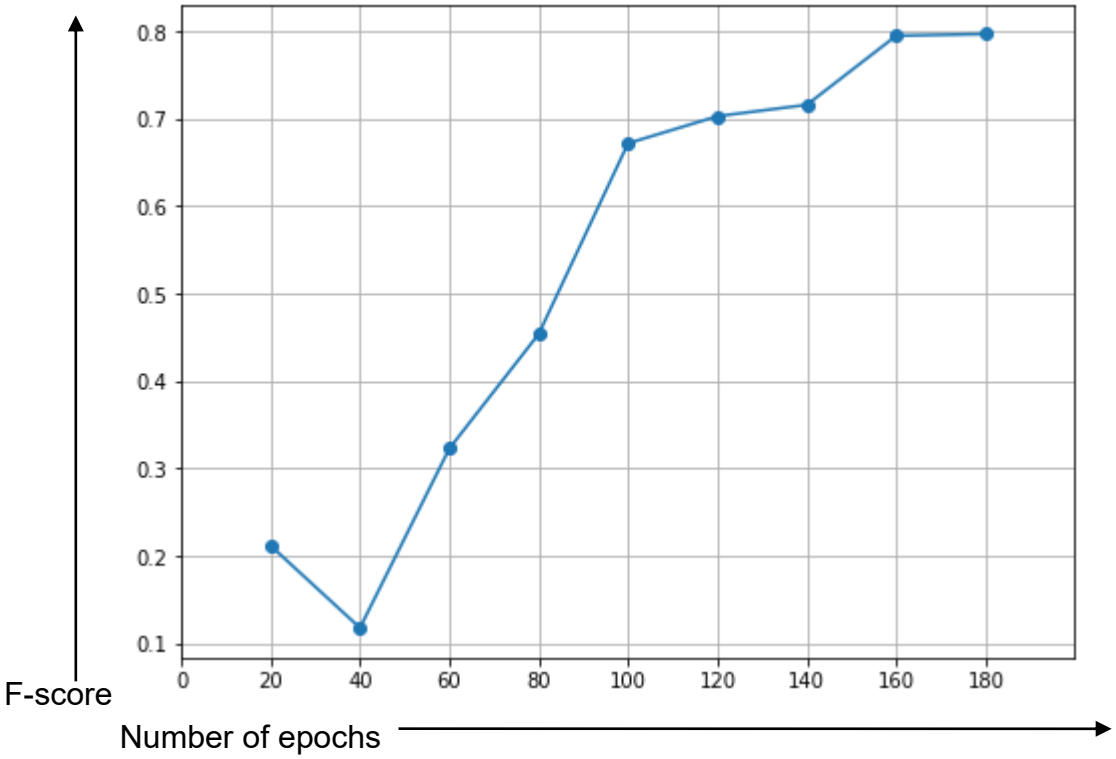


Figure 4.2: weighted average F-score of GraphSage model approach on DataAll

4.2 Results from DataAC

In this subsection, both approaches, GraphSage model and GATv2 model, yielded results and as such, both are presented and analyzed individually, followed by a comparison in growth speed, efficiency and results in their last iteration in order to assert their performances in a smaller graph, composed of 663 051 nodes (for better clarity in this dataset, observe Figures 3.3 to 3.6 displaying the graph used and Table 3.2 displaying the road types distribution).

4.2.1 Results of GraphSage

In the smaller dataset, the GraphSage model showed similar results to the previous dataset at the 180th iteration. However, as this dataset was much smaller, more iterations were done and the statistics showed a considerable growth in the classification of the least representative classes without sacrificing the others. This conclusion comes from the fact that in Figure 4.3 we can see the line chart still growing considerably (meaning that some classes scores were improving without lowering others) and in Figure 4.4 despite not having a considerable growth, in the last 40 epochs, it is still visible that graph growth, being either due to drastic improvements of the values in the under-represented classes or the improvement of the more representative classes (which we can see not being the case as the most representative classes from Table 3.2 had at the most a growth of 0.036).

In fact, in this test no class presented an F-Score value of 0, proving that the model still has the capabilities to classify the classes least favorable due to their under-representation and expected low homophily values (as can be seen in Table 4.2 under iteration 220 where even ‘Trunk’ road type has, despite still very low, an F-score of 0.013), despite its more basic architecture and hyperparameter tuning.

road type	*number of iterations*										
	20	**40**	**60**	**80**	**100**	**120**	**140**	**160**	**180**	**200**	**220**
living_street	0.000	0.612	0.555	0.650	0.673	0.685	0.716	0.829	0.857	0.891	0.923
motorway	0.424	0.528	0.511	0.564	0.576	0.596	0.657	0.745	0.788	0.890	0.947
motorway_link	0.000	0.000	0.132	0.326	0.482	0.569	0.716	0.795	0.842	0.960	0.999
primary	0.449	0.563	0.442	0.436	0.473	0.586	0.644	0.671	0.694	0.742	0.754
primary_link	0.059	0.094	0.010	0.011	0.011	0.250	0.283	0.297	0.306	0.323	0.344
residential	0.733	0.939	0.724	0.733	0.745	0.764	0.768	0.980	0.981	0.983	0.984
secondary	0.000	0.524	0.687	0.639	0.621	0.226	0.225	0.575	0.575	0.578	0.744
secondary_link	0.000	0.000	0.217	0.125	0.124	0.117	0.124	0.123	0.127	0.128	0.217
service	0.000	0.338	0.823	0.810	0.808	0.805	0.805	0.810	0.810	0.812	0.817
tertiary	0.000	0.000	0.000	0.398	0.457	0.457	0.443	0.449	0.452	0.474	0.480
tertiary_link	0.000	0.000	0.000	0.000	0.011	0.021	0.027	0.027	0.028	0.030	0.032
track	0.076	0.482	0.523	0.612	0.672	0.673	0.731	0.736	0.736	0.739	0.740
trunk	0.000	0.000	0.003	0.000	0.000	0.000	0.010	0.012	0.014	0.013	0.013
trunk_link	0.006	0.000	0.019	0.014	0.039	0.059	0.009	0.124	0.126	0.105	0.201
unclassified	0.000	0.391	0.495	0.586	0.685	0.724	0.749	0.934	0.941	0.959	0.977

Table 4.2: F-Score measures for every road type of GraphSage model approach on DataAC (represented in Table 3.2)

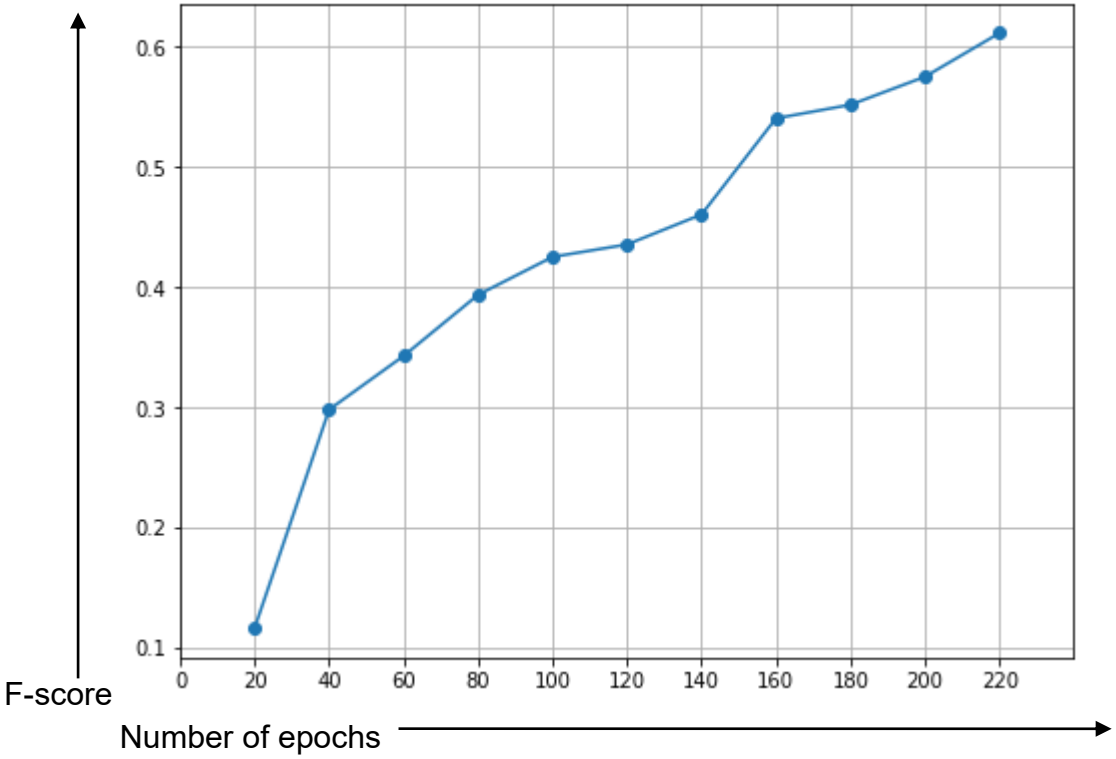


Figure 4.3: macro average F-score of GraphSage model approach on DataAC (represented in Table 3.2)

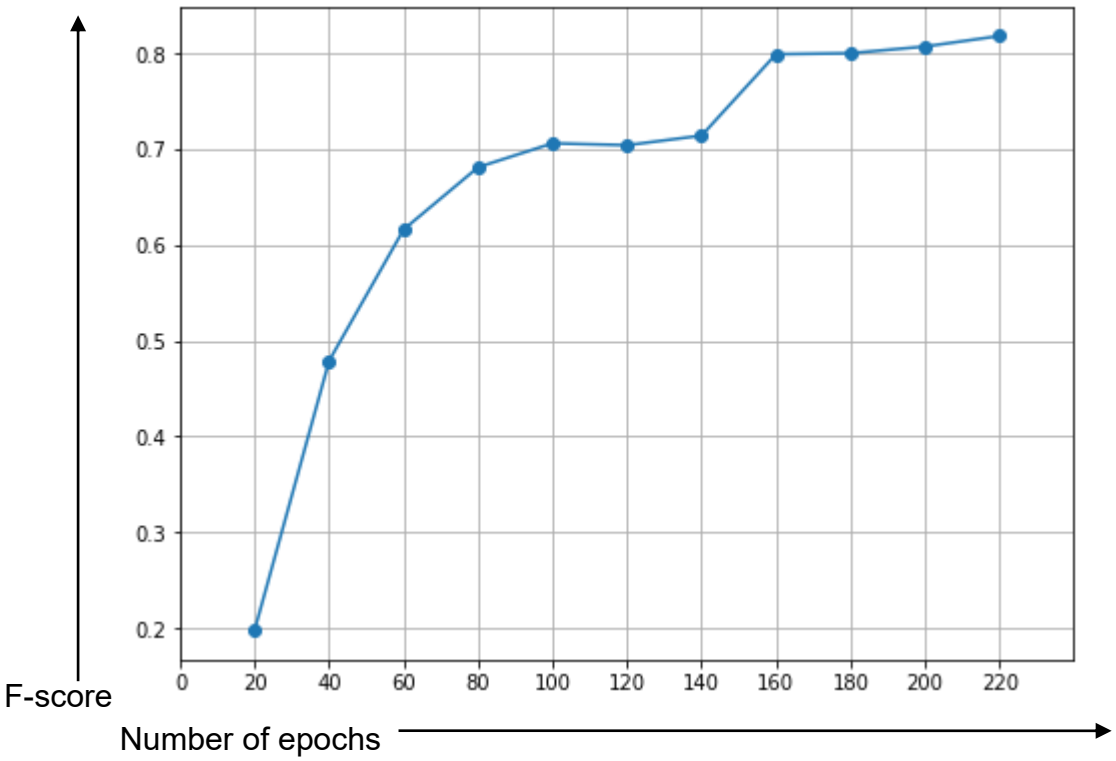


Figure 4.4: weighted average F-score of GraphSage model approach on DataAC (represented in Table 3.2)

4.2.2 Results of GATv2

In the smaller dataset, the GATv2 model managed to run full iterations thus being able to produce results. Although it managed to not have any F-Score at 0 as can be seen in the iteration 180 of Table 4.3, overall the scores were very low for the time taken to classify all the nodes. Still, it shows a great increase in the classification of all classes (as can be seen in Figure 4.5), while attempting to improve the results of the least representative at all times, as it shows in iteration 140 of Figure 4.6 where it sacrifices the score of a representative class (‘residential’ as can be seen from Table 4.3), where the curve lowers, but the macro average increases as seen in Figure 4.5.

Overall, this algorithm shows a lot of potential, maybe even more so with more iterations being done, but it requires a lot more resources and careful planning in its different architecture and hyperparameters since otherwise it takes far too much time to produce meaningful results.

road type	number of iterations								
	20	40	60	80	100	120	140	160	180
living_street	0.312	0.163	0.386	0.348	0.414	0.630	0.702	0.733	0.717
motorway	0.000	0.000	0.000	0.000	0.003	0.405	0.688	0.617	0.622
motorway_link	0.033	0.058	0.095	0.226	0.375	0.335	0.357	0.539	0.620
primary	0.000	0.000	0.032	0.040	0.051	0.057	0.083	0.157	0.301
primary_link	0.000	0.000	0.000	0.000	0.005	0.218	0.047	0.026	0.022
residential	0.000	0.000	0.010	0.682	0.716	0.726	0.517	0.774	0.774
secondary	0.000	0.000	0.000	0.000	0.022	0.048	0.187	0.315	0.130
secondary_link	0.000	0.000	0.000	0.048	0.034	0.031	0.017	0.031	0.041
service	0.000	0.000	0.000	0.000	0.619	0.625	0.633	0.651	0.630
tertiary	0.000	0.000	0.000	0.000	0.000	0.054	0.115	0.159	0.149
tertiary_link	0.000	0.000	0.000	0.000	0.000	0.000	0.015	0.010	0.011
track	0.000	0.000	0.291	0.348	0.646	0.594	0.573	0.577	0.639
trunk	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.015	0.009
trunk_link	0.000	0.002	0.000	0.001	0.001	0.001	0.002	0.003	0.004
unclassified	0.163	0.287	0.257	0.399	0.464	0.546	0.586	0.608	0.770

Table 4.3: F-Score measures for every road type of GATv2 model approach on DataAC (represented in Table 3.2)

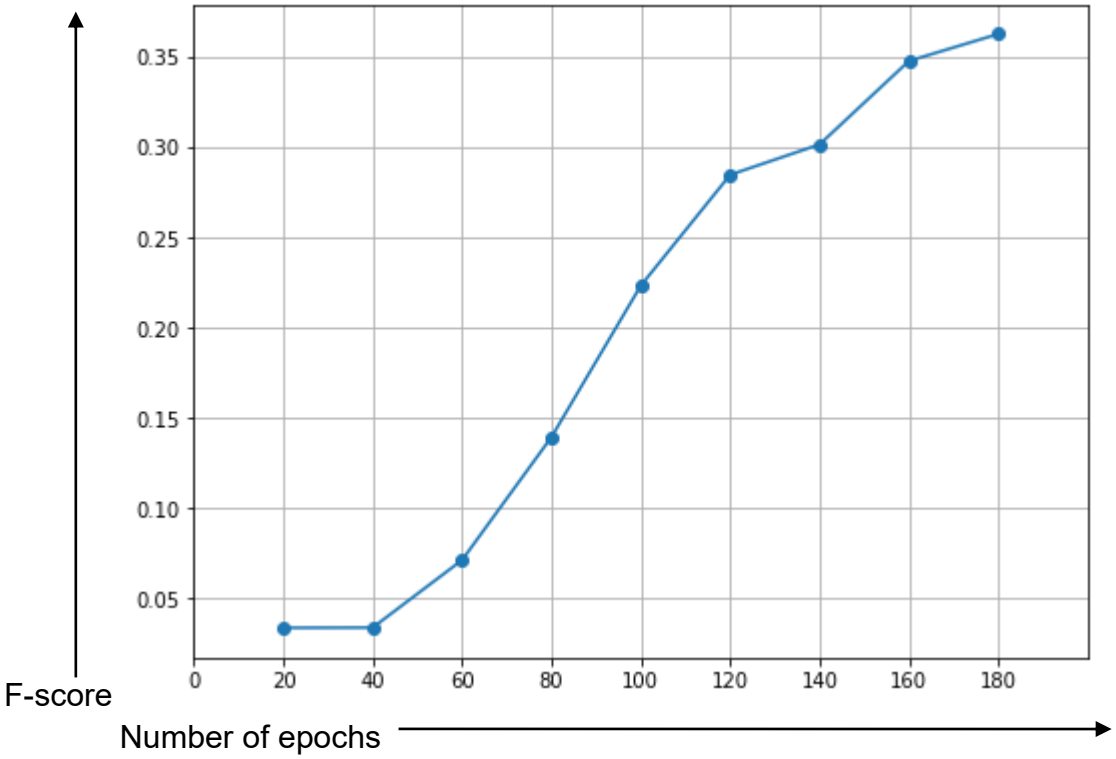


Figure 4.5: macro average F-score of GATv2 model approach on DataAC (represented in Table 3.2)

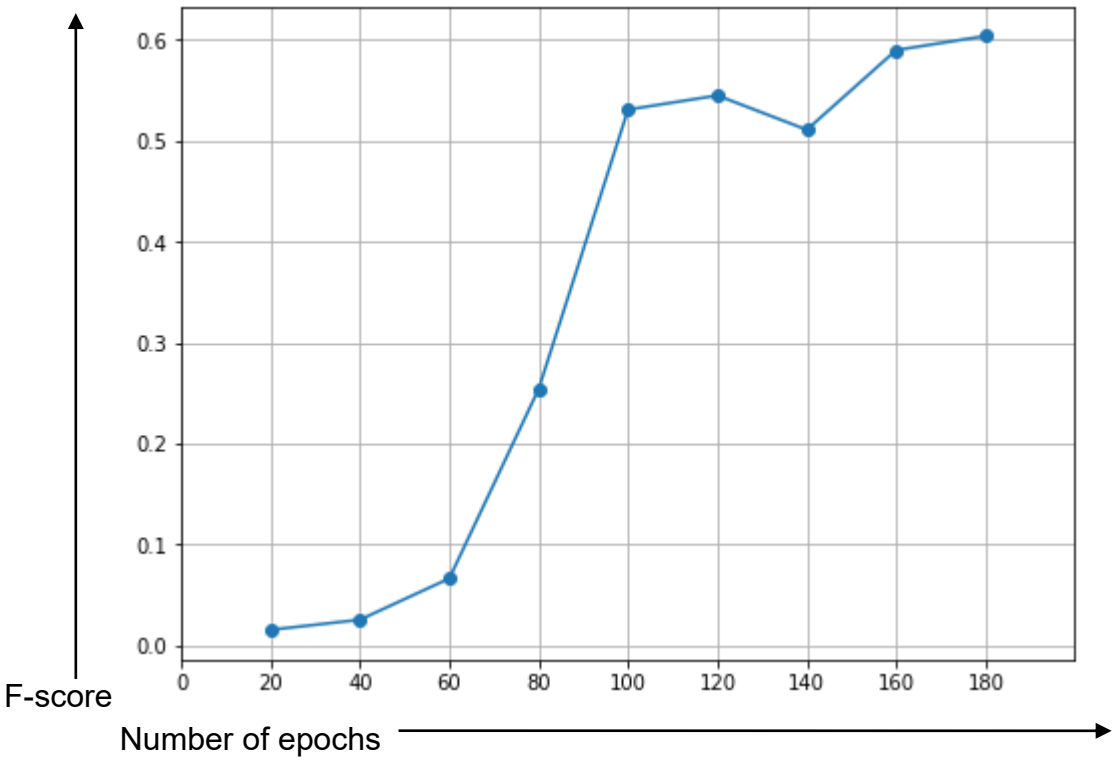


Figure 4.6: weighted average F-score of GATv2 model approach on DataAC (represented in Table 3.2)

4.2.3 Comparison of results

Overall, the results in all metrics point to GraphSage being better. As a simpler model, with little hyperparameter tuning achieved results greater than those of the article from which the work was initially based on. Still, seeing how much GATv2 improved with just a few adjustments (as discussed in chapter 3.4.2), there is a chance GATv2, with a more well planned strategy of tuning its structure and hyperparameters, can also achieve great results.

Nonetheless, GATv2 has problems regarding the consumption of the computer resources and requires a different strategy to be deployed whereas GraphSage can be used efficiently on a large scale as demonstrated by the results in chapter 4.1.

As for their growth by iterations, it seems that GATv2 takes more iterations to pick up the patterns of the road network with just 5 attention heads (as seen by the extremely low values in the first 40 iterations). Perhaps in the future, using more drastic values for the number of heads and hidden features, while taking in consideration that for every head the nodes will be processed into the same number

of hidden features (meaning that a balance is required to be found in order not to overload the computer).

5. Conclusions

In this final chapter, it is provided a comprehensive overview of the work conducted throughout this thesis and assess the extent to which the defined objectives were achieved. Following it, some observations are done on the future work that can be made in order to deepen the analysis done in this research, whether it is improving the results obtained or deploying different approaches.

5.1 Goal analysis

The objective of this work was to deepen the research presented in a previous article[1] and verify if neural network based network embedding algorithms (which were not considered there) could do a better work when it comes to node classification in a graph representation of the road network.

In regards to doing a more detailed analysis of this problem and its potential solutions this project explains some approaches with their flaws under certain design choices and the advantages that they bring, as could be seen at least by the results obtained by GraphSage.

However, when it comes to assessing the performance of these methods and their potential more can still be made. The computing limitations provided a big challenge when using one of the approaches, not being able to finely tune GATv2 and explore how well it could perform under better circumstances.

Nevertheless this work achieved purpose was to test new ways of solving this quite unexplored problem, as well as showing that NN based network embedding methods have potential to be further experimented with.

5.2 Future work

This work opens a lot of room to deepen the analysis of this problem and some future work should be done in order to truly solve it as, despite the overall great values achieved by one of the approaches used, certain road types could not be predicted correctly enough at the end. Not only that, but there may be regions in

which the road network structure does not share similar patterns, providing more unforeseen challenges.

Regarding data used, it would be better to find more road features for future proposed solutions. This would allow us to verify the algorithm's performances with multiple labels and help in identifying relationships between the different characteristics of the road network, relations such as most (if not all) ‘motorway’ roads in Portugal having a surface of Tarmacadam. In order to find unbiased data for these characteristics, attempting to contact companies responsible for the maintenance and registering of segments of the road network or contacting companies with visual data of the road network (via either satellite view or street view) and making as another project an algorithm to identify road surfacing materials.

Regarding the algorithms for the network embedding, the ones used in this project should have futurely more architectures tested such as adding a second convolutional layer before classifying, and more hyperparameter tuning before being ruled out as possible solutions for the problem explored in this project. This considering that GraphSage already had good results and GATv2 performance changed drastically with some tuning.

Additionally, more approaches can be explored, not only there are more network embedding algorithms that might have not been considered but also more are made over time with previous ones having some quality of life improvements. For instance, GraphSage has a hidden feature where if a node has no connections from which to go, the following walks are performed as it learns the neighborhood patterns and relationships are done considering a connection to itself. This feature is not currently present in GATv2 and if for some reason a node is isolated in an undirected graph, the algorithm will fail until the user removes that node.

References

[1] - Jepsen, Tobias Skovgaard, et al. "On network embedding for machine learning on road networks: A case study on the danish road network." 2018 IEEE International Conference on Big Data (Big Data). IEEE, 2018.
(https://arxiv.org/pdf/1911.06217.pdf)

[2] - HAMILTON, Will; YING, Zhitao; LESKOVEC, Jure. Inductive representation learning on large graphs. *Advances in neural information processing systems*, 2017, 30.
(https://arxiv.org/pdf/1706.02216.pdf)

[3] - BRODY, Shaked; ALON, Uri; YAHAV, Eran. How attentive are graph attention networks?. *arXiv preprint arXiv:2105.14491*, 2021.
(https://arxiv.org/pdf/2105.14491.pdf)

[4] - WANG, Minjie, et al. Deep graph library: A graph-centric, highly-performant package for graph neural networks. *arXiv preprint arXiv:1909.01315*, 2019.
(https://arxiv.org/pdf/1909.01315.pdf)

[5] Graphs. Available at https://www.geeksforgeeks.org/difference-between-graph-and-tree/, last accessed in 20/6/2023

[6] D. Zhang, J. Yin, X. Zhu and C. Zhang, "Network Representation Learning: A Survey" in IEEE Transactions on Big Data, vol. 6, no. 01, pp. 3-28, 2020.
doi: 10.1109/TBDATA.2018.2850013

[7] Neural Network. Available at https://www.tibco.com/reference-center/what-is-a-neural-network last accessed in 26/06/2023

[8] - KIPF, Thomas N.; WELLING, Max. Semi-supervised classification with graph convolutional networks. *arXiv preprint arXiv: 1609.02907*, 2016.
(https://arxiv.org/pdf/1609.02907.pdf)

[9] GUO, Kan, et al. Optimized graph convolution recurrent neural network for traffic prediction. IEEE Transactions on Intelligent Transportation Systems, 2020, 22.2: 1138-1149.
(https://ieeexplore.ieee.org/abstract/document/8959420)

[10] - BUNN, Frances, et al. Traffic calming for the prevention of road traffic injuries: systematic review and meta-analysis. *Injury prevention*, 2003, 9.3: 200-204.
(https://injuryprevention.bmj.com/content/injuryprev/9/3/200.full.pdf)

[11] - HONG, Huiting, et al. HetETA: Heterogeneous information network embedding for estimating time of arrival. In: *Proceedings of the 26th ACM SIGKDD international conference on knowledge discovery & data mining*. 2020. p. 2444-2454.
(https://dl.acm.org/doi/abs/10.1145/3394486.3403294)

[12] - YANG, Jihoon; PORTILLA, Jorge; RIESGO, Teresa. Smart parking service based on wireless sensor networks. In: *IECON 2012-38th Annual Conference on IEEE Industrial Electronics Society*. IEEE, 2012. p. 6029-6034. (https://ieeexplore.ieee.org/abstract/document/6389096)

[13] GROVER, Aditya; LESKOVEC, Jure. node2vec: Scalable feature learning for networks. In: *Proceedings of the 22nd ACM SIGKDD international conference on Knowledge discovery and data mining*. 2016. p. 855-864. (https://arxiv.org/pdf/1607.00653.pdf)

[14] OpenStreetMap. About OpenStreetMap. Available online: (https://wiki.openstreetmap.org/wiki/About_OpenStreetMap) last accessed in 29/06/2023

[15]- Official site of the OpenStreetMap osmium-tool used for the pre-processing (https://osmcode.org/osmium-tool/)

[16] HAGBERG, Aric; SWART, Pieter; S CHULT, Daniel. *Exploring network structure, dynamics, and function using NetworkX*. Los Alamos National Lab. (LANL), Los Alamos, NM (United States), 2008. (https://www.osti.gov/biblio/960616)

[17]GROSS, Jonathan L.; YELLEN, Jay. *Graph theory and its applications*. CRC press, 2005. p. 20

[18] WallStreetMojo. Stratified Sampling. Available online: www.wallstreetmojo.com/stratified-sampling/

[19] - VELICKOVIC, Petar, et al. Graph attention networks. stat, 2017, 1050.20: 10.48550. (https://personal.utdallas.edu/~fxc190007/courses/20S-7301/GAT-questions.pdf)

[20] - Macro average score formula. Available at https://www.v7labs.com/blog/f1-score-guide last accessed 2/7/2023

[21] - weighted average f-score formula and weight calculation. Available at https://www.v7labs.com/blog/f1-score-guide last accessed 2/7/2023