Individual Project
Final Report

ID Number: F027704

Programme: Electronic and Electrical Engineering

Module Code: 23WSC357

Project Title: Application of graph neural networks for simulating airway flow in the lungs

Abstract:

Background

Lungs are vital organs in the respiratory system responsible for oxygen exchange in the body. Simulating airflow within them using graph neural networks (GNNs) is a novel approach. This study aimed to introduce a GNN model to assess how accurately lung ventilation can be predicted.

Methods

Lungs' tree-like structure translates well to a graph representation. A large dataset generated from a validated full-scale airway network model provided training and testing data. This data included pressure, radius, and spatial coordinates, used to construct the graph. A 6-layer convolutional GNN with a rectified linear unit (ReLU) activation function was implemented to predict the pressure and flow rate at each graph node.

Results

The initial model produced high errors (1000%) for both pressure and flow rate prediction. Significant improvements were achieved through a combination of a learning rate scheduler, a new loss function (MAPE), and early stopping, reducing the error to 50% for both tasks. While the specific contribution of the learning rate scheduler needs further investigation, the results indicate potential for GNNs in lung ventilation modelling.

Conclusion

GNNs have shown great promise as a potential technique to model lung ventilation however further hyper-parameter tuning as well as the development of a physics-informed graph neural network is required to improve the accuracy.

Contents

1 Introduction

Motivation

According to the Lung cancer statistics study [1] conducted in the USA, approximately 350 people die of lung cancer each year and ranks as the 8th leading cause of cancer deaths among both sexes. The diagnosis of lung cancer is challenging due to its complex geometry. In addition to this, the lungs cannot be seen or touched which leads the medical world to rely on simulations of the lungs. Accurate simulations take a long time to compute, and, on the contrary, time-efficient simulations lack the necessary detail to provide an accurate image of the lungs. The study by Spencer R. and others [2] is a great example of this where the model requires the lungs to be modelled as symmetric which compromises anatomical realism.

The motivation of this report and underlying research is an attempt to solve this dilemma by introducing the field of deep learning via graph neural networks to assess lung ventilation. This particular investigation has not been previously undertaken, thus there is no assurance that a deep learning model can be effectively trained for this specific application and subsequently enhance the simulation of the human lungs. Nevertheless, considering the progression of graph neural networks across various domains, particularly in healthcare, the likelihood of achieving success is very high. For that reason, any advancement in this research has the opportunity to save lives.

1.1 Background

The lungs

The human lungs are an essential organ responsible for gaseous exchange between the air and the bloodstream. During inhalation, oxygen from the air enters the lungs and diffuses into the bloodstream. Conversely, carbon dioxide, a waste product of cellular respiration, diffuses from the blood into the lungs and is exhaled. The lungs are composed of a complex network of branching airways, terminating in air sacs called alveoli, which provide a large surface area for efficient gas exchange. Additionally, the lungs help regulate blood pH and contribute to immune function. The morphometry of the lungs was first well documented by Ewald R Weibel in 1963 [3] where it was identified that the lung bifurcates down 23 generations from the trachea all the way down to the alveoli as illustrated in Figure 1. This unique tree-like geometry makes it suitable for abstraction into a graph.

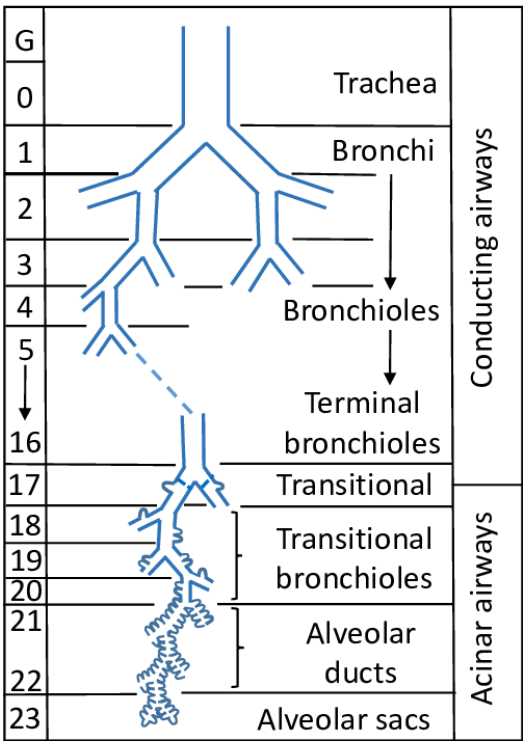

Figure 1 - Model of human airway system assigned to generations of symmetric branching from trachea (generation 0) to acinar airways (generations 15–23), ending in alveolar sacs [3].

Graph Theory

In discrete mathematics, a graph is a structure which represents relationships between a set of objects which are abstracted into nodes and edges. This can be represented mathematically via equation 1.

$$G = (N, E) \tag{1}$$

Where G is the graph, N is the set of nodes, and E is the set of edges where each edge can be represented as a pair of nodes (N_1, N_2). There are many types of graphs but for simplicity, consider this visual representation of a homogenous undirected graph G with 7 nodes and 6 edges which is shown in Figure 2. Another important concept in graph theory for this project is the degree of a node. The degree of a node N denoted as degree(N) is the number of edges connected to N. An example of this from Figure 2 is:

- Degree(1) = 2
- Degree(4) = 1
- Degree(3) = 2

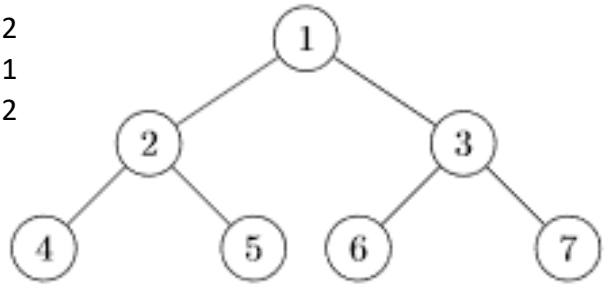

Figure 2 – homogenous undirected graph G

For a graph with a tree structure like in Figure 2 the degree of a node is a simple but effective technique in identifying the location of the node. From inspection, node 4,5,6,7 are terminal nodes (nodes located at the bottom of the tree) and therefore will all have a degree of 1. For large graphs, the average node degree is a useful way of assessing the shape without needing to generate a plot of the graph and therefore helps save computational power.

The next key idea in graph theory is graph representation. There are many ways to do this however the easiest way to visualise this is via an adjacency matrix. For a graph without any edge weights like Figure 2, the graph can be represented in the following way:

$$A_{ij} = \begin{cases} 1 \; for \; edge(i,j) \\ 0 \; otherwise \end{cases}$$

$$A_{ij} = \begin{bmatrix} 0 & 1 & 1 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 1 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 1 & 1 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 \end{bmatrix}$$

For any adjacency matrix, the row index corresponds to that node's connectivity. If a graph has an edge weight, the value stored in the element is the weight of the edge. For undirected graphs the adjacency matrix is always symmetrical along the diagonal as seen by A_{ij} however for directed graphs, where an arrow dictates the direction between two nodes, this does not apply. The last characteristic a graph has is whether it is heterogenous or homogenous. A graph where each node and edge are of the same type is called homogenous. A graph with many types of

nodes and edges is called heterogenous. An example of a heterogeneous graph is a navigation systems network where edges may be roads, pavements, tunnels, and bridges. Using this theory, the lung airways can be modelled as a homogeneous undirected graph where edges represent the length and diameter of the airway, and the nodes represent pressure and flow rate values.

Graph Neural Networks

Graph Neural Networks (GNN) are geometric deep learning algorithms that operate on graphs. The most common type of GNN is a graph convolutional neural network (GCN) which will be used as an example to provide intuition. GCNs are most commonly applied to images which are 2D arrays. Instead of inputting an image, a graph can be represented in terms of an adjacency matrix. Basic neural networks apply linear transformations to their input data. Mathematically, this can be expressed using equation 2:

$$h = Wx \tag{2}$$

Where x represents the input data, W denotes the weight matrix, and h signifies the hidden vector. However, the same mathematical operations cannot be straightforwardly applied to graphs due to the presence of connections between nodes. These connections are important as they hold important information about the abstracted structure. In graph theory the concept of network homophily exists which states that nodes which are connected are more likely to be similar than ones that are disconnected, emphasizing the relevance of understanding the relationships between nodes within the graph.

A neural network layer for a graph has two necessary input elements. One of which is the feature description x_i for each node i. This is the multiplication of the number of nodes N and the number of node input features F which provides the feature matrix X as shown in equation 3.

$$X = N \times F \tag{3}$$

The second element is a suitable representation of a graph, usually in the form of an adjacency matrix A. This allows for a definition of a neural network layer for graph G like in Figure 2, formulated in equation 4.

$$H^{(l+1)} = f\left(H^{(l)}, A\right) \tag{4}$$

Where L is the number of layers. When L = 0 and therefore being the input layer, H^0=X. Now consider this layer-wise propagation rule expressed in equation 5 :

$$f\left(H^{(l)}, A\right) = \sigma\left(AH^{(l)}W^{(l)}\right) \tag{5}$$

$W^{(l)}$ is the weight matrix of the l-th layer and σ is the nonlinear activation function. The most common σ is a ReLU function. The purpose of the activation function is to avoid linearity. This is because for complex problems data cannot be modelled well with linear equations, so it is common to multiply the matrix multiplication by σ. Other activation functions include $tanh$ which is more commonly used for recurrent neural networks. There are however two limitations to this layer which are addressed by Thomas Kipf and Max Welling [4]. firstly, the multiplication with A doesn't include the node itself (unless there are self-loops), which is mitigated by adding the identity matrix to A; second, the scale of feature vectors changes due to the non-normalized A, which is resolved by normalizing A such that all rows sum to one using the diagonal node degree matrix D. By combining these adjustments, the propagation rule introduced in Kipf and Welling was formulated as shown in equation 6:

$$f(H^{(l)}, A) = \sigma(D^{-\frac{1}{2}}\hat{\mathrm{A}}\,\hat{\mathrm{D}}^{-\frac{1}{2}}H^{(l)}W^{(l)}) \tag{6}$$

With $\hat{A} = A + I$ where I is the identity matrix and $\widehat{D}$ is the diagonal node degree matrix of $\hat{A}$. The convolutional layer will be using this mathematics with the purpose of recognising patterns in the graph data.

1.2 Aims and objectives

The aim of this project is to introduce a graph neural network model to assess how accurately lung ventilation can be predicted. There is only one main objective that needs to be met to make this project a success. This is to develop a machine learning model that can predict nodal pressure and flow rate values based on an unseen graph (which represents a patient's lungs) with a percentage error value of 30% or less. The scope of work for this project is as follows:

1. Discover if the nodal pressure and flow rate values are trainable outputs. The approach to this is an assessment of the learning curve for a small batch of data and identifying a learning curve with a clearly decreasing loss.
2. Training the machine learning model with a large dataset. Convolutional neural networks are known for requiring a large amount of data to predict accurately. To ensure this, patient data with multiple timestamps was gathered via the FAN model [5]. These were CSV files with a size of over 500GB. The goal here is to predict pressure and flow rate values for the large dataset.
3. Testing and validating the machine learning model. During this part of the project, detection of overfitting or underfitting will be necessary and important in understanding how well the deep learning model is predicting. Due to the complexity of the lungs underfitting may be an issue as the model could be too simple to capture and understand underlying patterns.
4. Improving the model. To improve the model hyperparameters will be tuned based on preliminary results.

2 Literature Review

2.1 Overview of Lung physiology and anatomy

[3] ”Morphometry of the Human Lungs”

This book was written in 1963 by Ewald R. Weibel and was one of the first to provide a comprehensive overview of lung morphometry, everything from how the lung works all the way to its geometry, dimension of airways and alveolar capillary networks. The first idealised airway model as shown in Figure 1 was an important step for the scientific community in modelling the lungs. The author identified that the lungs are comprised of 3 sections including a complex network of repeated airways that bifurcate down 23 generations. The geometry of the lungs resembles an upside-down tree which provides the opportunity to model the lungs as a graph. The author also points out that the thickness and diameter of the branches get thinner as you traverse down the tree. This is important information that the neural network will hopefully learn from processing a large dataset. The author also points out the acinar region of the lungs which is located between the 20-23rd generation and is where gas exchange occurs. This provided the fundamental understanding behind the dataset, especially the AcinarID which is a feature of each node of the network. The AcinarID value is binary. 1 represents a terminal airway and 0 is a non-terminal airway and will help in generating a more realistic prediction of the lung airways. This is an excellent book however due to its age it lacks the necessary complexity to model the lungs accurately. As seen in Figure 1, a symmetrical structure is presented however from research it is clear that the lungs are asymmetrical. This is the main limitation of this book.

2.2 Existing simulations of airflow

Since Weibel's model A with a regular dichotomy, a lot of research was conducted to model the lung airways more accurately.

[6] "Tracheobronchial geometry: Human, dog, rat, hamster–a compilation of selected data from the project respiratory tract deposition models"

This paper was the first to seriously tackle the problem of asymmetry in the conducting airways. In particular, it focused on the change in bifurcation angles and what parameters caused this. It was found that parameters like length, diameter, and the angle to the direction of gravity all have an impact on the bifurcation angles and therefore the shape of the lungs. The length and diameter were considered in the model which provided confidence in the inclusion of both the length and diameter as useful edge features for a graph. Unfortunately, this research did not provide any correlation of graph theory to lung airway modelling and so the techniques in this research did not apply to this project.

[7] "Development of human respiratory airway models: A review"

To address the randomness of lung geometry, stochastic lung models were theorised. Monte Carlo methods were added to the model described in the previous paper. The author describes the implementation in the following way :

"A statistical lung structure with a random selection of airway branches as well as particle transport pathways were developed using this technique which allowed for variation in dimensions (diameters, lengths, branching and gravity angles). Using this approach, statistical density functions were used to describe the airway dimensions which led to a geometry that was able to characterize the inter-subject variations of the human lung."

Simulating anatomically accurate models became much better after this research was published and what was impressive about this paper was the consistency of the model throughout all generations. It was able to calculate airway parameters for length, diameter, branching angle and volume to $3x10^8$ nodes. This however came with a massive computational cost which is expected when randomness was embedded at each stage of the simulation. Due to this it will not be feasible to apply this using university machines and is beyond the scope of this project however this may be very useful for future research on this project when developing a physics-informed graph neural network.

[5] " Lobar Ventilation in Patients with COPD Assessed with the Full-Scale Airway Network Flow Model and Xenon-enhanced Dual-Energy CT ".

The research conducted in this paper tested the FAN model and compared it to the xenon-enhanced dual-energy (DE) CT. The xenon-enhanced dual-energy (DE) CT is a large and

powerful machine capable of detecting regional lung ventilation abnormalities in patients with COPD. It was found that the FAN model correlated with the results of the xenon-enhanced dual-energy (DE) CT and provided similar results. The main advantage of the FAN model is that it's a simulation unlike the counterpart whilst providing similar results. This allows for a much simpler generation of data. The only strong positive of the Xenon-enhanced Dual-Energy CT is that it is used clinically which therefore provides more trustworthy data however for the purpose of the introduction of GNNs to lung ventilation this is not necessary. Therefore, the FAN model was chosen to generate the dataset.

2.3 Graph theory

[8] "Using graph theory to analyse biological networks".

Graph theory is a well-studied branch of mathematics and has received much more attention from systems biologists to compute models of complex biological systems. This paper summarised the fundamentals of graph theory and gave insight as to how it is used in protein-protein interaction networks, regulatory networks, signal transduction networks and metabolic and biochemical networks. It provided the necessary intuition to understand how graph theory is suited to certain biological phenomena. This paper provided a great overview however did not go into the necessary detail for tree topologies within graph theory which is the primary focus for this project as the lungs bifurcate down in a tree-like manner. It also illustrated a general lack of research done by systems biologists in modelling tree structures within biology. Not only the lungs but biological hierarchies may also benefit from this. The paper did however provide confidence that graph theory will work well in modelling the lungs due to the extensive applications of graph theory in other biological domains.

[9] " Spectral graph theory efficiently characterizes ventilation heterogeneity in lung airway networks"

This is an exciting paper which provided a linear operator for quantifying properties in the airflow of the lungs. It addressed most criticisms from the paper above and considered properties like asymmetry of the lungs in its mathematics which was very promising. It also introduced the Maurey Matrix as an alternative representation of airway tree networks. The Maurey operator is a more efficient tool for the reduction of complexity of airway flow calculations and works by removing the asymmetry of smaller airways sacrificing complexity and reducing dimensionality. This obviously does not reflect the dynamics of ventilation completely which is the main disadvantage of this paper. The authors provided clear mathematics for modelling the lungs accurately as a graph. This method was directly applied to this project and therefore had a great contribution to the method of transferring patient lung data into graphs. In addition to this, the Maurey Matrix will be useful if the graph neural

network becomes informed by physics in future work. This is beyond the scope of this project however important for anyone continuing this project for further research.

2.4 Deep learning

[10] “A holistic overview of deep learning approach in medical imaging”

The author of this paper provided the main frameworks and techniques used in deep learning for medical imaging purposes. Convolutional neural networks were discussed. The paper states that CNNs are generally used for image-based tasks and are flexible for hospital environments where both 2D images like X-rays and 3D images like MRI scans need to be processed. It also introduced the convolutional layer. The paper explains that convolutional layers operate via sliding a filter all over an image in a sliding window form until the whole image is covered. This was useful background information to build intuition on the convolutional neural network.

It also identified that a full-scale airway of the lungs is incredibly large and therefore a convolution may slide through the graph part by part just like for an image. The author states that this technique is used for supervised learning which was exactly the problem of this project. Supervised machine learning is when an algorithm is trained on a labelled dataset. Input data is paired with corresponding output labels, and the algorithm learns to map the input to the correct output. For this project, the labelled data is the pressure and flow rate values at each node of the graph. Based on this research a convolutional layer was chosen for the design of the GNN.

The paper also introduced the main activation functions used in deep learning which are the ReLU and Sigmoid functions. It stated that ReLU activation functions are most popular in medical imagining and based on this the ReLU function was chosen to be used as the activation function. On the whole, this is a fantastic paper. The only critique is that it focused on images so not all information translated exactly to this project.

[11] “On Empirical Comparisons of Optimizers for Deep Learning”

The research paper conducted an extensive comparison of Adam, RMSprop, SGD, and momentum optimizers. It was observed that popular adaptive gradient methods such as Adam or RMSprop consistently outperformed momentum or gradient descent. However, no consistent trends were found when comparing optimizers that could not approximate each other. Consequently, the paper concluded that utilizing Adam or RMSprop would result in optimal outcomes. Nonetheless, the analysis did not account for batch size as a parameter, which could significantly influence the performance of each optimizer. Previous studies referenced in this paper have indicated that increasing the batch size may widen the gaps between training times for various optimizers. The paper offered valuable insights into the

most widely used optimizers and emphasized that performance is heavily reliant on how hyperparameters are configured for a particular problem. Therefore, the primary takeaway from this paper is the necessity for experimentation with hyperparameters to achieve satisfactory performance. Although the paper aided with prioritizing optimizers, it lacked testing different batch sizes which undermined its certainty.

[12] “How To Choose Loss Functions When Training Deep Neural Networks”

The author, Jason Brownlee, investigated a wide range of loss functions, accompanied by Python examples. Among these, three were specifically highlighted for regression problems: Mean Squared Error (MSE), Mean Squared Logarithmic Error Loss (MSLE), and Mean Absolute Error Loss (MAE). The author advocated for MSE as the default choice of loss function, particularly for data following a Gaussian distribution. However, due to the uncertainty regarding the distribution of pressure and flow rate in the airways, reaching a definitive conclusion on the preferred loss function was challenging. While the journal provided valuable implementation examples, beneficial for both the design and implementation phases of the project, it lacked the necessary information to make a conclusive choice. Consequently, the MSE loss function was selected with the expectation that it may be subject to change in the future depending on initial results.

The literature review outlines the absence of research conducted in graph-neutral networks for lung airways. Existing simulations and models mainly provided confidence for implementing the radius and diameter of airways as valid edge weights to model the lung’s airways. From all models reviewed the FAN model was to be used due to relatively simple, quick, and accurate generation of a dataset. Research in graph theory for lung networks exists. The mathematics was used to write the code for modelling the lungs as a graph. Deep learning research for medical applications is heavily focused on images as inputs for predictions. Since graphs and images are both matrices, the two research areas can be merged to provide a path to explore lung ventilation via graph neural networks.

3 Methods

3.1 Choosing a deep learning framework

The code for this project was written in Python using the PyTorch framework. The choice of framework was based on two main reasons. Firstly, most research is being conducted in PyTorch and secondly, PyTorch has a specific library called PyTorch Geometric specifically to train GNNs which saved a lot of time during the project. From a software engineering perspective, it also aids readability and reusability due to the plethora of documentation and resources available.

3.2 Implementation of patient data into a graph

Graph convolutional neural networks require input data to be a graph represented as an adjacency matrix. To do so it was necessary to transform the data appropriately. A common way is to use NetworkX [13], a Python package for the creation and manipulation of networks. However, before this can be achieved, an intuition for the data is necessary [see Appendix 1. and 2. for examples of data].

Node data for each patient was stored in a CSV file which held all node information. First is the index of each row representing the node itself and all the information in each row represents the features of the node. Next are Points:0, Points:1 and Points:2 which represent the node location in Euclidean space. Next are Pressure_Pa and FlowRate_mL which are continuous values of the pressure and flow rate at that point in space. These are the features that the neural network will be predicting. The AcinarID is also included and is a binary value which signals whether a node is a terminal node. For example, the first node will always be a terminal node and therefore have a value of 1 as this represents the trachea. Using Figure 2 as an example of an oversimplified graph representation of a lung, nodes 4,5,6 and 7 would be terminal nodes and therefore have an AcinarID of 1. In lung physiology, these values are always fixed and are considered boundary conditions and therefore do not need to be trained. The AcinarID was used as a filter to identify which nodes are to be trained. The AwayRadius_mm is a continuous value also stored in the CSV file and was necessary for calculating the radius of an airway. However, this value alone was not enough to understand the airway radius. To do so, the connectivity of the graph was necessary. The provided dataset came with a `generated_airways.edge` file [see Appendix 2.] which describes the connectivity of all node timestamps for each patient. In essence, the first column in the .edge file indicates the edge index whereas the second column and third column hold the information about which two nodes are connected. The full .edge and .csv files provided all the necessary information to generate a graph. For this application the edges represent the airways of the lungs and since the airways exhibit physical properties the weight of an edge can be used to represent this. To calculate edge weight, the average between two connected nodal AwayRadius_mm values was implemented. This however was not the full representation of a lung airway. Another vital feature is its length. After each bifurcation, both the radius and length of the airway decrease. In determining the radius of an airway, a similar methodology was applied. Each node within the airway possesses coordinates (X, Y, Z) within Euclidean space. By identifying the nodes that are interconnected, the use of Pythagoras's theorem was used to calculate the length between these nodes as shown in equation 7.

$$l = \sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2 + (z_2 - z_1)^2} \tag{7}$$

Since the length and airway radius are interrelated, multiplying the two values together provided a more comprehensive representation of the edge weight. With the availability of edges, edge weights, nodes, and node features, all necessary information for generating a full-scale airway model of the lungs was accessible. This implementation was conducted using NetworkX, and a plot of a patient's lungs was generated and illustrated in Figure 3.

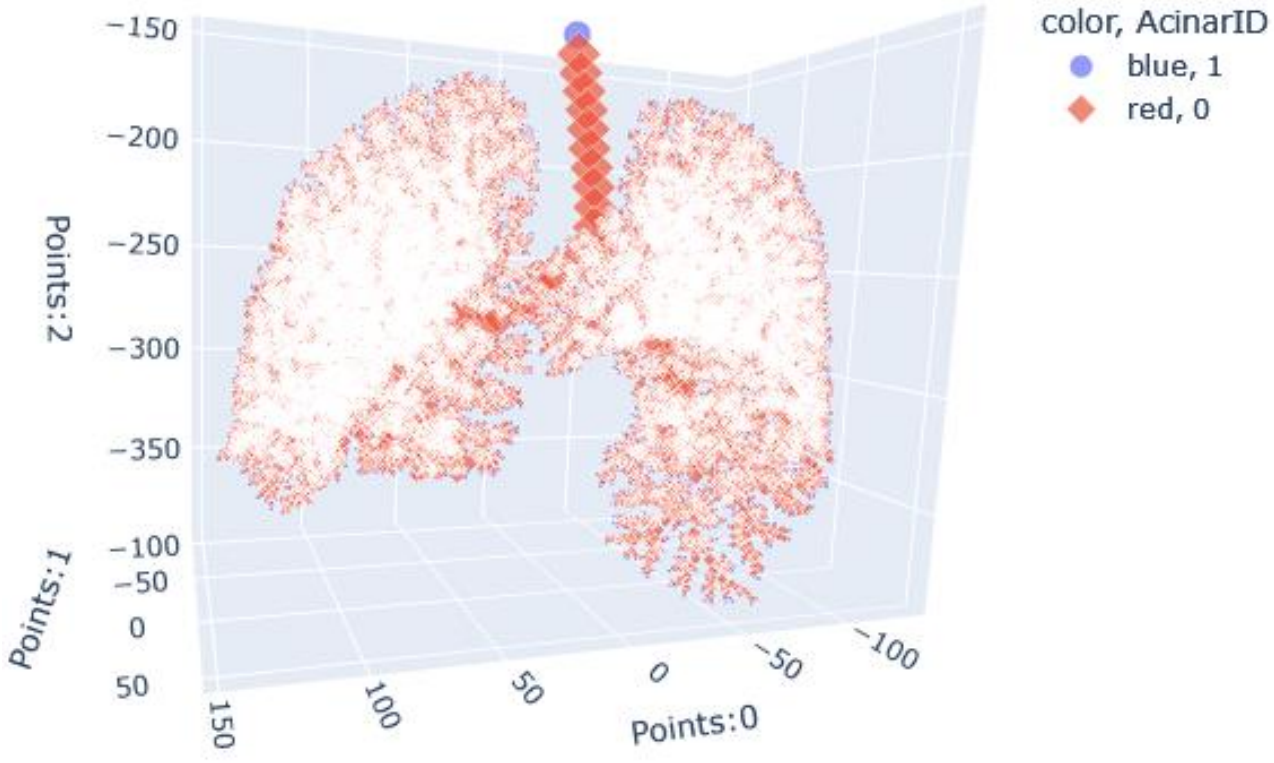

Figure 3 – Human Lungs of a patient modelled as a graph

3.3 Verification of trainability

Due to the complexity of the project, it was unknown whether the pressure and flow rate would be trainable. This is because the lungs, by nature, inherit complex patterns and it was unknown whether this can be recognised by a deep learning model. To do this, a simple program was made which followed the flow chart shown in Figure 4. Here the dataset held one element which is precisely Figure 3. The data was manipulated into relevant PyTorch tensors which were then fed into the training loop. Once the model training process was complete the program was to output a loss curve. The model was a graph neural network with an input layer

for a graph, a single hidden layer of 16 neurons and an output layer for the pressure and flow rate prediction.

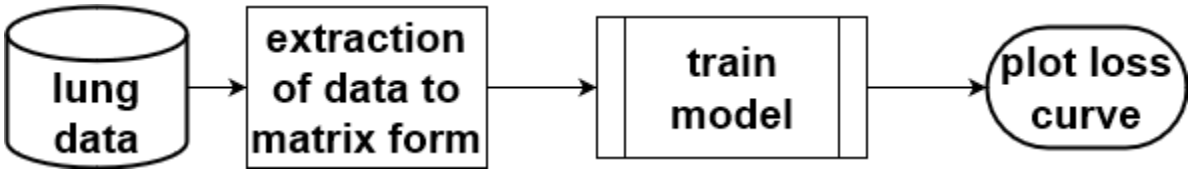

Figure 4 – Flowchart for training verification software

During the design of this simple model, an Adam optimiser and an MSE loss were used to compute the loss. The reasons for this are explained thoroughly in 3.6 Development of Training and validation loop but in essence, the Adam optimiser is a stochastic gradient descent method that adapts the learning rate for each parameter, allowing it to converge quickly and efficiently. MSE is the mean squared error and is a common criterion used for regression problems [12].

3.4 Design of Data Handler

An important part of any deep learning project is understanding how the data will be used. The data provided by the FAN computational model was already pre-processed. This means there were no missing values, duplicate records or any need to convert categorical variables by encoding them with techniques like one hot or target encoding. The data provided was split by patients. Each patient had a unique lung structure(therefore unique .edge file) and snapshots of the lung's respiratory parameters (.csv files) in 5ms intervals were captured where each patient had a different number of time stamps. This deep learning model needed to be trained, validated, and tested on large amounts of data and therefore a data handler was necessary. To do this a simple UML class diagram was created as shown in Figure 5. Since the input to the deep learning model is a graph, the dataset required the input to be a collection of graphs for each patient. The Airway_Graph class served as a comprehensive repository for all relevant data for each timestamp of every patient. As illustrated in Figure 3, an instance of the Airway_Graph encapsulated the entirety of patient-specific information. These instances were efficiently managed within the Graph_Container class, where each patient's data is organized. This design yielded several notable advantages. Firstly, scalability was addressed as the dataset expanded to accommodate additional patient data. With the Airway_Graph and Graph_Container classes, no adjustments were required, ensuring a streamlined integration process. Secondly, the storage of all patient data within a single class facilitates efficient storage and retrieval mechanisms. By saving the aggregated data as a .pkl file, significant time savings were achieved, as complex data handling processes were abstracted away. Lastly, this design promotes a c Gar separation of concerns, with data management operations being completely abstracted from the training, validation, and testing phases. This ensured a modular and

maintainable architecture, enhancing code readability and facilitating ease of future enhancements or modifications.

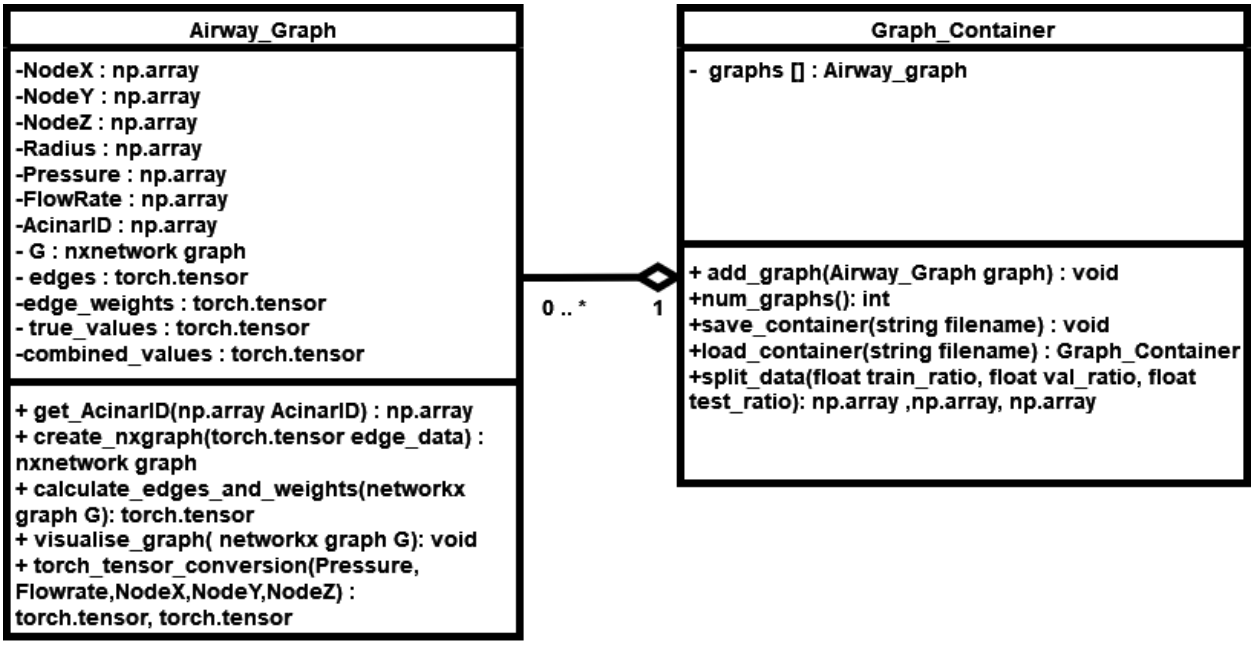

Figure 5 – UML Class Diagram for Data Handler

3.5 Design of GNN model

Designing a GNN model is a fundamental step of any machine learning project, unique to every problem and can be broken down into a few sections. The input layer, hidden layers, output layer and choice of activation function.

Input layer

For a GNN the input is a graph. This included the node feature matrix which stored the pressure and flow rate values for each node, the edge index which was a substitute for the adjacency matrix of the graph which also described the connectivity and lastly, the edge attributes matrix which was a weighted value that combined represented the radius and length of each airway of the lung.

Output layer

The output layer needed to provide true value predictions for the pressure and flow rate for each node. This is a regression problem which required an output of continuous values and

therefore no activation function was necessary. Therefore, two output neurons that connect to the last hidden layer were needed. One for the pressure and one for the flow rate prediction. For classification problems where the probability that the input is for example of type car, plane or boat a SoftMax function is commonly applied to the output neurons.

Hidden layers

Within the design of hidden layers of a GNN, there are two main choices an engineer must make. How many hidden layers to include and how many neurons each hidden layer should have? The number of neurons in each hidden layer dictates the capacity and expressiveness of the model. Increasing the number of neurons enhances the model's capacity to capture complex relationships in the input graph data, but it also escalates the risk of overfitting, especially with limited training data. Additionally, higher neuron counts increase computational requirements during training and inference. In addition to this, the number of hidden layers determines the depth and hierarchical representation learning capability of the GNN. Deeper architectures enable the model to learn increasingly abstract features and hierarchical representations of the graph structure, potentially enhancing its expressiveness and learning ability. However, deeper models are more prone to overfitting and may suffer from vanishing or exploding gradients during training, which can hinder convergence and degrade performance.

Since the lungs are a complex structure, one or two hidden layers would be insufficient. However, 20 hidden layers were likely to be expensive computationally on a university Linux machine therefore 6 hidden layers were chosen. In addition to this, a neuron count of 32 per layer was chosen. At this point, an exact number of hidden layers was not necessary as fine-tuning the model was expected.

Activation function

As discussed in 1.2.3, the activation function was necessary in forming a convolutional layer. In this case, a Rectified Linear Unit (ReLU) function (shown in Figure 6) was chosen based on research conducted in deep learning for medical imaging paper [10]. It addressed the vanishing gradient issue encountered in deep networks by providing non-linearity without saturation. ReLU ensures efficient backpropagation by maintaining a derivative of 1 for positive inputs and 0 for negatives, thus preventing gradient saturation and enabling effective pattern recognition. This is illustrated graphically in Figure 6 where a gradient exists for all input values more than 0. Its computational efficiency, compared to alternatives like Sigmoid or Tanh, coupled with success in many deep learning tasks, including medical imaging, led to its selection as the activation function. However, this remained a part of the model open to experimentation.

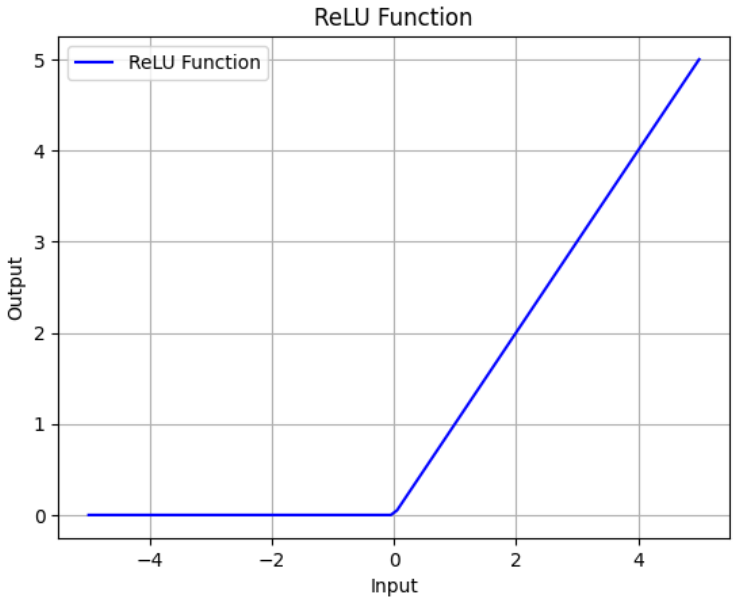

Figure 6 - ReLU function

GNN model in PyTorch

All design features above were implemented using PyTorch geometric. The model was contained within its own Python script and imported into the training and validation loop. [See Appendix 3. for GNN model in code.]

3.6 Development of Training and validation loop

Choice of Loss function and Optimizer

When developing the training procedure two essential features needed to be identified. What is the criterion (or loss function) and optimizer? The criterion is a measure of how well a model's predictions match the actual target data values and changes depending on whether the problem is a regression or classification problem. For classification problems it is common to use categorical cross-entropy however the goal of this project was to predict continuous values and therefore a mean squared error (MSE) loss function, as shown in equation 8, was chosen based on [12] where it states this is the default loss for regressions problems and should be evaluated first.

$$MSE = \frac{1}{N}\sum_{i=1}^{N}(y_i - \hat{y}_i)^2 \tag{8}$$

Where MSE = Mean Squared Error, N = number of data points, y_i = true value, $\hat{y}_i$ = predicted value

Note this equation is generalised. For the case of this algorithm y_i and $\hat{y}_i$ were arrays which store both the pressure and flow rate. Equations 9 and 10 are snippets that display the shape of the arrays.

$$y_i = [[true\_p\_node0, true\_q\_node0], [true\_p\_node1, true\_q\_node1], \dots] \quad (9)$$

$$\hat{y}_i = [[pred\_p\_node0, pred\_q\_node0], [pred\_p\_node1, pred\_q\_node1], \dots] \quad (10)$$

Where p and q represent the pressure and flow rate, respectively. The resulting equation for the MSE loss is expressed in equation 11.

$$MSE = \frac{1}{N}\sum_{i=1}^{N}\left((p_i - predictions_{i,0})^2 + (q_i - predictions_{i,1})^2\right) \quad (11)$$

Where :

p_i is the true pressure value for the i-th sample,

q_i is the true flow rate value for the i-th sample,

$predictions_{i,0}$ is the predicted pressure value for the i-th sample,

$predictions_{i,1}$ is the predicted flow rate value for the i-th sample,

N is the total number of samples.

The optimizer is an algorithm used to update the parameters of the model during training to minimise the loss function. It determines how the model learns from the data received by constantly adjusting the weights and biases. The criterion and optimizer work together to train the model and therefore the pair must be chosen well. The adaptive moment estimation (Adam) optimizer was chosen based on research conducted [11]. Adam is an extension of stochastic gradient descent which incorporates an adaptive learning rate and momentum and is well suited to MSE due to these features. MSE tends to have noisy gradients due to its sensitivity to outliers and the concept of momentum helps negate this. It accelerates the optimization process by accumulating past gradients and so helps smooth out the optimization trajectory, enabling faster convergence [14].

Development of training and validation loop

Once the criterion and optimizer were identified the training and validation loop for the project was designed and developed as illustrated in Figure 7. The function takes in the train and validation set as parameters. 70% of the total dataset was allocated for training, 10% for validation, and the remaining 20% for testing. This was based on convention in the deep learning community however was subject to experimentation.

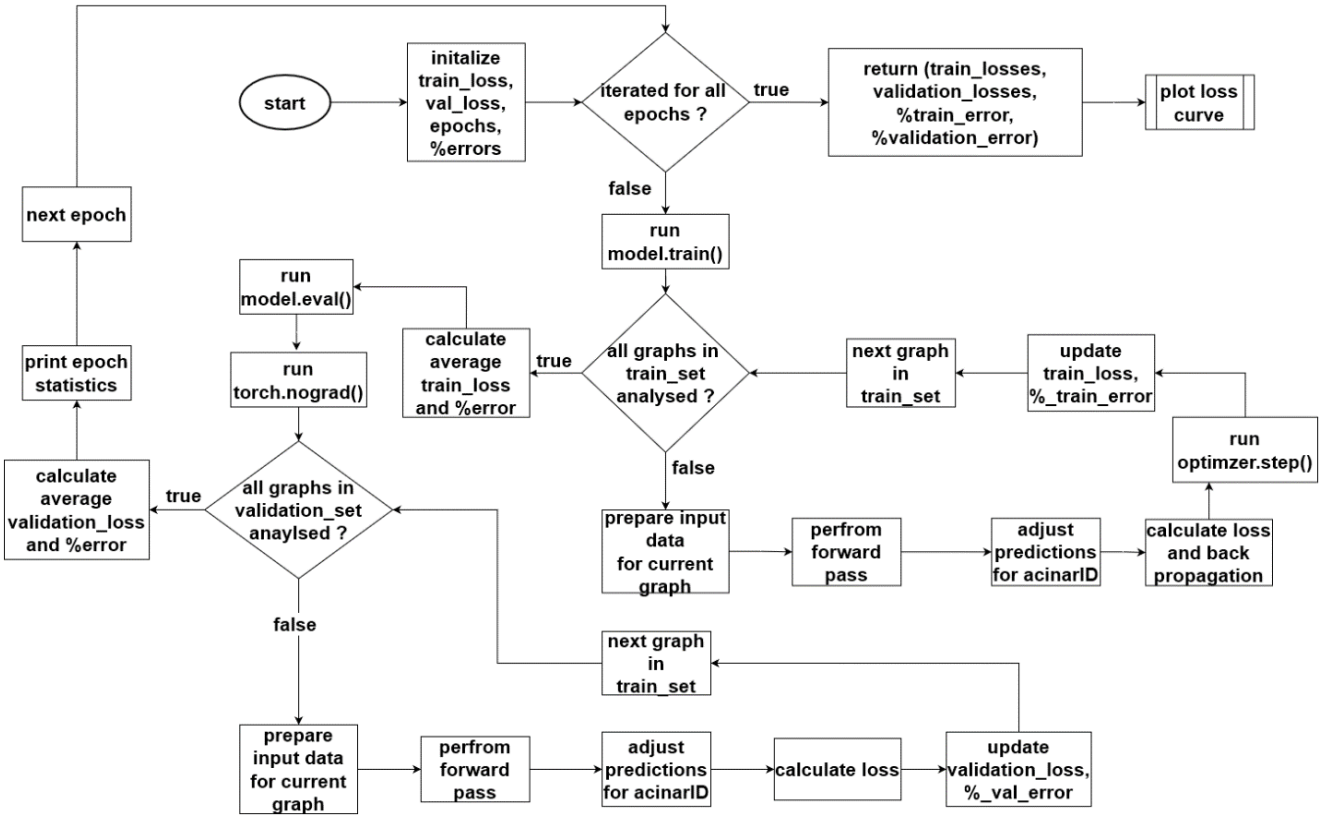

Figure 7 - Flowchart for test and validation loop

The validation dataset was used to fine-tune the model's hyperparameters and monitor its performance during training. It was used to prevent overfitting by providing an independent dataset to validate the model's performance. In addition to this, the percentage train and validation errors were also calculated as well as the MSE error. This provided a more intuitive idea as to how close the final predicted pressure and flow rate values were to the true value. The calculation for the mean absolute percentage error(MAPE) is shown in Equation 12.

$$MAPE = \frac{1}{N}\sum_{i=1}^{N}\left|\frac{true\_values_i - predictions_i}{true\_values_i}\right| \times 100 \quad (12)$$

The flowchart in Figure 7 was built from 3 loops. The epoch iterator, which is the outer loop and within it the training and validation loops. Training and validation are similar but have two important distinctions other than the fact that they use two different subsets of data. Firstly, during validation, a function called `torch.nograd()` was run. In the training loop, gradients were calculated and updated using backpropagation. This was necessary for training the model as it helped optimize the parameters to minimize the loss function. On the other hand, during the evaluation loop updating the model parameters was not necessary; all that was required was to evaluate the model's performance on unseen data. Running `torch.nograd()` ensured this was the case. Secondly, backpropagation and optimization occurred only during the training loop because they are essential processes for iteratively updating the model's parameters to minimize the loss function. Backpropagation involves computing the gradients of the loss with respect to the model's parameters, which is necessary for understanding how each parameter contributes to the overall error. Additionally, the optimization algorithm which in this case was Adam utilizes these gradients to adjust the model's parameters in the direction that minimizes the loss.

The forward pass is another important process that occurred during both training and validation and was run when graph data was in the correct type and shape. This built-in PyTorch function, commonly implemented within the `forward` method of a neural network model, propels the input data forward through the network's layers, ultimately producing predictions. During this process, each layer performs its operations on the input data, applying transformations based on the learned parameters of the model. This operation was crucial for generating model predictions during both training and validation phases, allowing for the evaluation of the model's performance and generalization ability on unseen data.

Both training and validation also had post-processing. The code checked for terminal nodes which were identified by nodes with an `acinarID == 1`. These nodes did not require training and were therefore filtered out in the prediction process. The reason for this post-processing was explained in section 3.2. Finally, after each epoch, all calculated errors are printed out and stored in arrays to be plotted.

3.7 Testing the model

Testing the model is crucial in evaluating the performance of a deep-learning model. It provides insight as to how well a model generalises to unseen data. The function designed for testing can be seen in Figure 8.

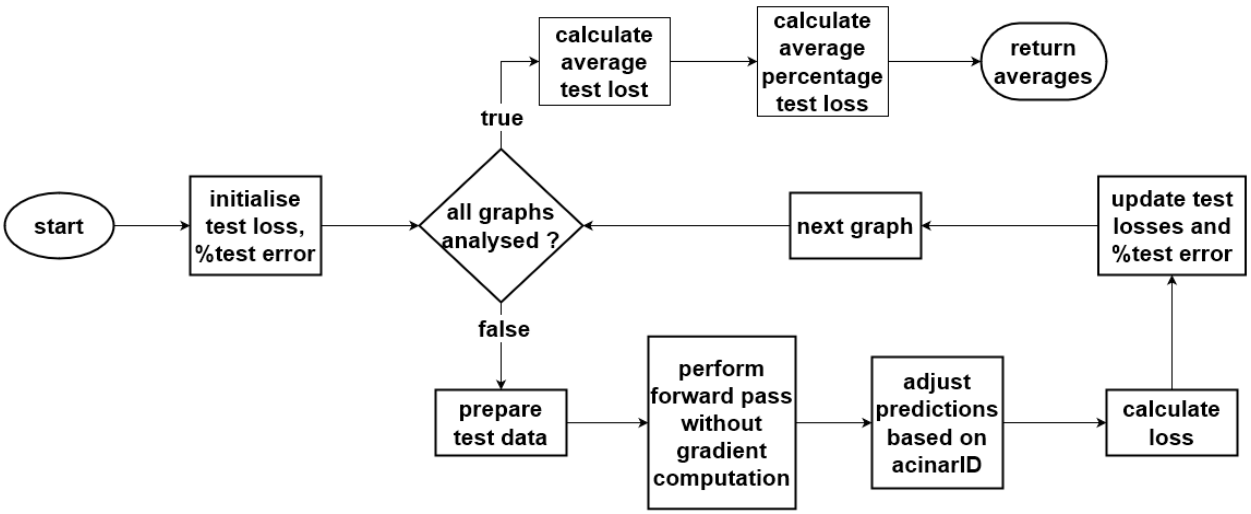

Figure 8 - Flowchart for model testing

The test function returns two averages: one for the MSE loss and the other for the percentage error. A well-performing model results in test averages similar to the best-performing epoch of the loss curve during the training stage. This was to indicate that the weights and biases of the model were successfully generalizing on unseen data.

4 Results and Analysis

The result was a working deep learning model which was able to provide substantial evidence for learning advanced patterns of how the airway flows in the humans' lungs.

4.1 Results for trainability verification

Figure 9 shows the output of the code from Figure 4 in section 3.3. It is a model run only on one full-scale graph. A clear reduction in the loss was observed until the 150th epoch when the model reached a plateau and stopped learning. This was a good sign that underfitting did not occur as the gradient of the curve after the 200th epoch was very close to 0 meaning no more learning has been achieved.

The model converged to an MSE error of 20 which was still considered high. From this learning curve, it was clear that a validation set was necessary to verify model generalization and was to be implemented into the next stage of design and implementation. The objective for this part of the project was to observe a learning curve which provides evidence for learning and Figure 5 delivers on this. The evidence presented clearly supported the suitability of this problem for deep learning applications. By appropriately tuning hyperparameters, adjusting hidden layers and neurons and employing optimization techniques, a substantial reduction in loss can be achieved.

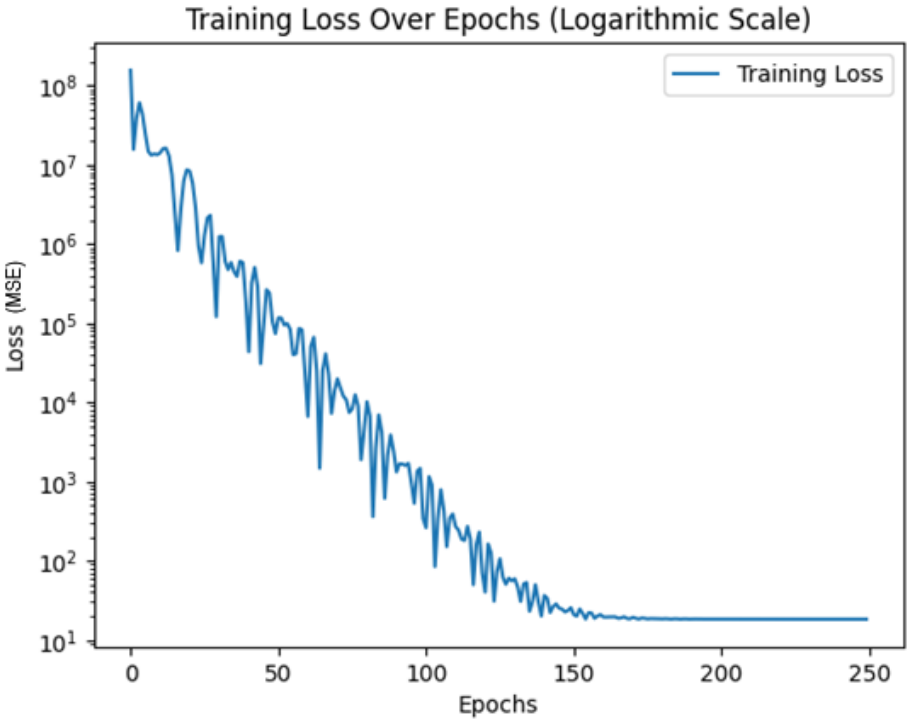

Figure 9 - Training loss Over Epochs for validation of trainability

4.2 Initial Results

The deep learning model outlined in the design phase was run with an input of 6 full-scale airway graphs and the initial results can be seen in Figure 10 and Table 1.

Epoch 50/50	
Training MSE loss	6.9201
Training Percentage Error (%)	1009.3411
Validation MSE Loss	22.9812
Validation Percentage Error (%)	677.1682

Average Test Loss	33.1212
Average test percentage error (%)	1260.5240

Table 1 – best-performing epoch metrics for Figure 10

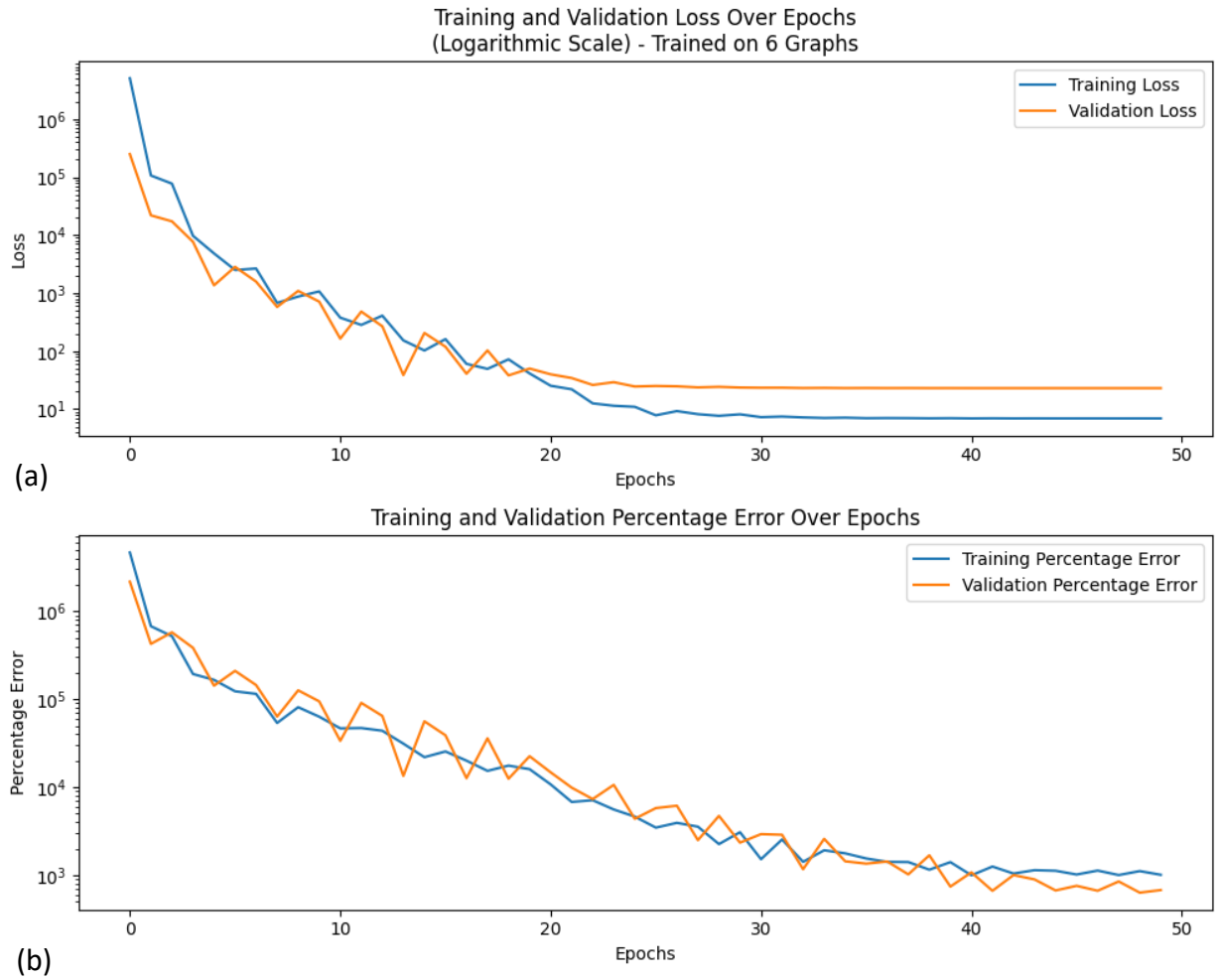

Figure 10 - Initial Results for a dataset of 6 graphs

The provided results from Table 1 indicate the performance of the deep learning model on the 50th epochs. The training Mean Squared Error (MSE) loss was reported as 6.9201, while the validation MSE loss was considerably higher at 22.9812. Additionally, the training and validation percentage errors were remarkably high, with the training percentage error at 1009.3411% and the validation percentage error at 677.1682%. These high errors suggest that the model struggled to accurately predict the target variable based on the input features, both during training and when evaluated on unseen validation data. Moreover, the average test loss and percentage error were further elevated, with an average test loss of 33.1212 and an average test percentage error of 1260.5240%. These results indicated that the model's performance extends beyond the training and validation datasets, highlighting its limited ability to generalize to unseen data. Overall, these findings suggested that the model needed further refinement and experimentation was necessary.

The shape of the curves from Figure 10 is also worthy of analysis. The deep learning model was steadily learning until the 25th epoch in Figure 10 (a). Afterwards, a clear plateau was witnessed. A useful piece of information that was extracted from Figure 10 (a) was the

similarity of the training and validation curve up to the 20th epoch. This was a good indicator that the model generalized well for unseen data. The plateau experienced shows a loss curve with a gradient close to 0 after the 25th epoch indicating that the model had stopped learning and highly likely that the model was not underfitted. On the other hand, slight overfitting could be inferred from Figure 10 (a). This is because the validation curve intersected the training curve at the 18th epoch and from that point experienced no further improvement, unlike the training curve. This is a sign that the model was too specialized for the training set. Experimentation with overfitting prevention techniques like early stopping was necessary.

Another thing to point out was that the MAPE error from Figure 10 (b) decreased with a similar trend to the MSE loss. This suggested that improvements made to the model which decreases the MSE loss could further decrease the percentage. However, despite the consistent decrease in percentage error, it remained notably large across all three datasets - training, validation, and test - hovering around 1000%, indicating that this was not the correct loss function to use for this problem and therefore an alternative was necessary. To add to this, unlike Figure 10 (a), a clear plateau was not seen showing that there was a possibility that the model did not finish learning and further indicated that a change in loss function was necessary.

4.3 Model experimentation and tuning

Following the analysis of preliminary findings, it became evident that conducting experiments was essential to minimize the percentage error to an acceptable range, preferably below 20%.

In order to tackle the large percentage error, it was decided to eliminate the Mean Squared Error (MSE) error and instead utilize the mean absolute percentage error (MAPE) as the loss function. The main reason for MSE's poor performance was due to the combination of errors with two different physical quantities: pressure in (Pa) and flow rate in (L/s), which is strictly incorrect. The new approach was to calculate two separate MAPE errors, similar to equation 4, but one for the pressure and the other for the flow rate. Since MAPE is unitless the errors can be summed and fed back into the optimizer, providing much better feedback on how well the model was learning. Due to this change, a small redesign was necessary which consisted of substituting the loss function and calculating MAPE separately in the following way:

$$MAPE_{pressure} = \frac{1}{N}\sum_{i=1}^{N}\left|\frac{true\_values\_p_i - predictions\_p_i}{true\_values\_p_i}\right| \times 100 \quad (7)$$

$$MAPE_{flowrate} = \frac{1}{N}\sum_{i=1}^{N}\left|\frac{true\_values\_q_i - predictions\_q_i}{true\_values\_q_i}\right| \times 100 \quad (8)$$

$$Loss_{Total} = MAPE_{pressure} + MAPE_{flowrate} \quad (9)$$

Equations 7 and 8 are separate calculations of MAPE for the flow rate and pressure which are then added together to give the total loss as seen in equation 9. $Loss_{Total}$ was then used for backpropagation.

Another issue observed in the initial results was the presence of a plateau that needed to be addressed. Both Figure 10 (a) and (b) did not converge to a low enough loss which could be the result of the learning rate. A specific module, known as a learning rate scheduler, was identified to help deep learning models in overcoming this challenge. PyTorch's ReduceLROnPlateau scheduler adjusts the learning rate throughout the training process based on the validation loss. Its primary function is to decrease the learning rate when the validation loss ceases to improve, typically indicating that the model is either plateauing or converging. This potential solution was implemented into the code.

Furthermore, a common practice in experimenting with deep learning models involves adjusting the initial learning rate. The Adam optimizer was initially configured with a default learning rate of 0.001 which changes using the adaptive learning rate mechanism. Nevertheless, it is beneficial to support the Adam optimizer by choosing a more suitable learning rate. Experimentation was conducted with learning rate (lr) values of 0.01, 0.005, and 0.1. A learning rate of 0.01 provided the best results.

Early stopping with a patience of 7 epochs was implemented to ensure that the model stops training once the validation loss fails to improve for a specified number of consecutive epochs. This was implemented to address the slight overfitting that occurred in the preliminary finding illustrated in Figure 10(a). This approach helped prevent overfitting and conserved time and computational resources, particularly when dealing with large inputs.

Experimentation was conducted with different numbers of neurons per layer. 8,16,32 and 64 neurons per hidden layer was chosen however this did not have any significant impact on the learning curve.

Experimentation with optimizers was also conducted. Prior research conducted [11] concluded that RMSprop was worth experimenting with instead of Adam. Unfortunately, RMSprop yielded identical performance.

4.4 Final Results

After many iterations and combinations of potential improvements stated above, a much better configuration was found. With the incorporation of a learning rate scheduler, utilization of the new MAPE loss function, and Adam optimizer with a learning rate of 0.01, the model experienced a significant enhancement in its learning capacity. As a result, the model achieved a percentage error of 50% for both the pressure and flow rate as illustrated in Figures 11 and 12. This progress represents a twenty-fold improvement from the initial results, where the percentage error stood at 1000%.

The training and validation curves for both the pressure and flow rate exhibited an identical trend with some expected volatility. This consistency suggests that the model was effectively learning from the training data and generalizing well to unseen validation data, which was subsequently confirmed through the test script, yielding a test percentage error of 51%. From a graph theory standpoint, this indicates that every node requiring a prediction for both pressure and rate will have a prediction possessing an average accuracy of 51%.

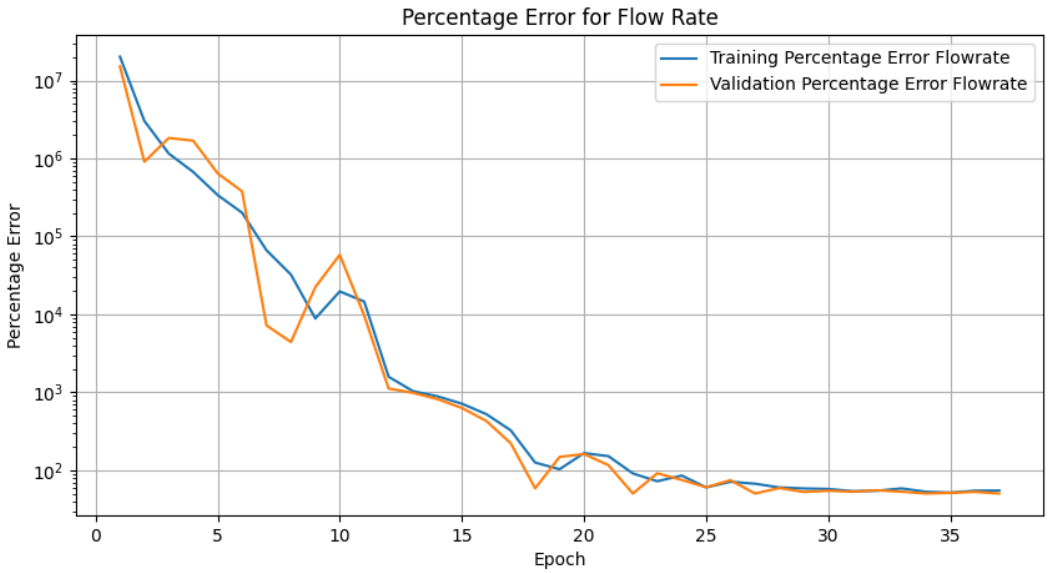

Figure 11 – Learning curve for the accuracy of flow rate predictions using 6 graphs

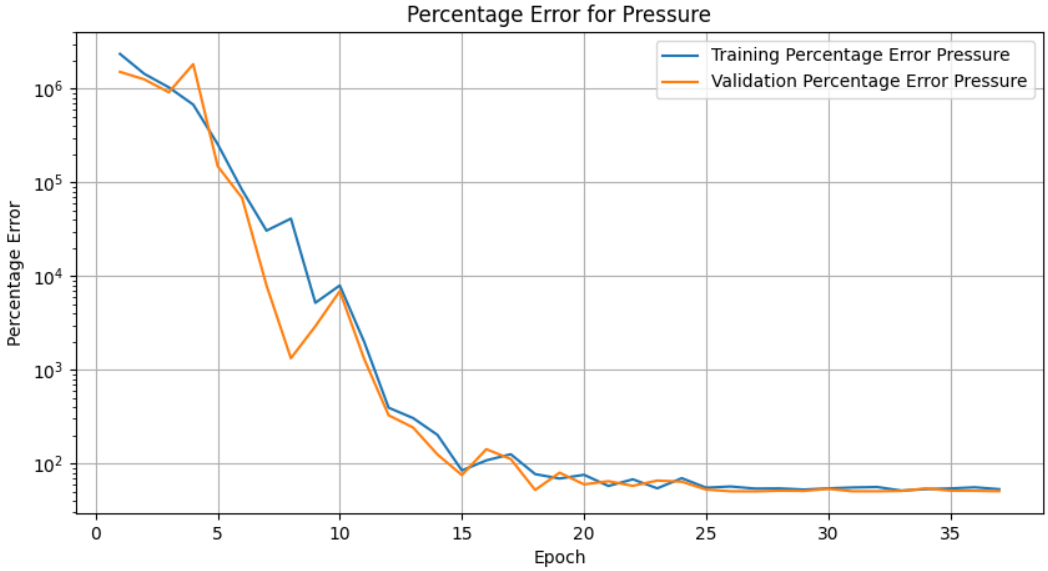

Figure 12 – Learning curve for the accuracy of pressure predictions using 6 graphs

An interesting difference to note between the final and preliminary results is the number of epochs on the x-axis. The final model incorporated early stopping with a patience of 7 which was triggered and resulted in 15 fewer epochs being run. As a result, it is evident that the problem with overfitting seen in Figure 10 (a) was minimised as both the validation and training curves converged at the same epoch, with the same final value. In addition to this, less computational power was used resulting in a more efficient algorithm.

From the final results, it is not clear exactly how much the learning rate scheduler contributed to the improvement of the model however interesting insights can be extracted. Both the final and preliminary results reached a plateau early, at around the 25^{th} epoch which suggests that the learning rate scheduler was not effective. This is important as it indicates that further experimentation with learning rates is not necessary and decreasing the loss further may require developing a new model which could potentially use a different type of GNN instead of a convolutional layer.

5 Discussion

Experimentation was successful, and the results represent a deep-learning model capable of predicting nodal pressure and flow rate values with an accuracy of 50%. Initial results were unsatisfactory due to an extremely high MAPE error of around 1000%. This was solved with a combination of techniques. The substitution of the criterion had the largest impact on reducing the loss. As mentioned in the results section, it was unclear whether the learning rate scheduler had a positive impact on the model which may suggest that more work on the learning rate is unnecessary as it was tuned sufficiently. There are three main reasons for this speculation. Firstly, Adam has a built-in variable learning rate with the task of optimization. Secondly, the ReduceLROnPlateau learning rate scheduler was implemented with the purpose of assisting the optimizer with plateau and thirdly, the initial learning rate for the optimizer was also experimented on with a range of values. Extensive experimentation was conducted on the learning rate from all aspects. The lack of improvement in this regard indicates that the model is too simple and additional time invested into learning rate tuning is likely inefficient.

Exploration of loss functions in this project is also worth discussing since its impact on the project's percentage error was substantial. MSE was initially chosen as it is the most widely used loss function for regression problems but was substituted due to poor performance. Results suggest that lung airways may not exhibit a Gaussian distribution, which is the primary assumption for the application of Mean Squared Error (MSE) analysis. Its poor performance was also caused by combining MSE losses of pressure in Pa and flow rate in L/s. This is useful but what is more interesting is why MAPE worked so well. Firstly, it allowed the errors to be added which was important for this model's success. MAPE's effectiveness also suggests that the scale of pressure and flow rate values in lung physiology varies significantly across patients. This

variability is due to the randomness associated with lung generation. MAPE, by considering the percentage difference between predicted and actual values, accommodates this variability in scale and provides a standardized measure of prediction accuracy. Based on this, further experimentation with loss functions is not necessary as there is a great degree of certainty that MAPE as the criterion is optimal for this problem.

Limitations and challenges

Deep learning models are known for their heavy computational requirements and require powerful hardware. Unfortunately, during the course of this project, memory issues arose when attempting to create a graph container to store the dataset, resulting in a limitation on the input size. Additionally, due to the time complexity associated with training and verifying a model's performance on large datasets, it was necessary to reduce the dataset size to 6 in order to conduct thorough experimentation within the project's timeframe. Varying the dataset sizes to 500 or 1000 graphs may yield very different model performance.

Furthermore, the code developed for this project is specific to CSV input data format and the FAN model, limiting its reusability for researchers employing different methodologies, and causing challenges associated with collaboration between research groups.

It is also important to acknowledge that the deep learning model's accuracy, particularly from a medical perspective, is still inadequate and requires further tuning. This is an obvious limitation as it cannot be used for any real medical application until its predictions are extremely accurate.

Further Development

Further research is required to conduct additional experiments on the model to potentially decrease the percentage error to 1% or lower. One recommendation is to explore different optimizers that are responsible for calculating the gradient descent. Although Adam was utilized throughout the entire project, it is uncertain whether it is the most effective optimizer. Another suggestion involves adjusting the number of layers and neurons per layer. While there was an exploration of neuron count with values of 16, 32, and 64, no significant impact was observed. However, due to limitations in computational resources, the number of layers was not increased to larger values such as 20, 50, or 100. Consequently, the optimal number of layers and neurons remains unknown. There may exist a unique correlation between neuron count and the number of hidden layers for this problem, which could enhance the model's performance.

Another interesting idea that could yield large improvements in the prediction of pressure and flow rate values is developing a physics-informed graph neural network(PIGNN). This would provide the deep learning model with insight into the aerodynamics and airway resistance in

the airways and how this changes at each generation. The equation that governs the airway flow is just Ohm's law where flow = pressure gradient/resistance. Embedding this simple equation could provide the deep learning model with the necessary information to provide an accurate prediction. Moreover, it is essential to investigate and test different neural network layers in conjunction with this technique. This is because the potential effects of alternative deep neural network architectures on performance are yet to be fully understood. A custom layer specific to this problem may also provide promising results.

To address the problem of computational resources, using a cloud computing service with powerful GPUs and TPUs is recommended. These services come at a premium however ensure great execution times and do not require anything to be run locally. This will provide much faster development and experimentation.

6 Conclusion

The project delivered on its goals and reached the following achievements:

- Successful identification of trainability for nodal pressure and flow rates is shown in Figure 5, a promising training loss curve converging to a low MSE loss.
- Developed a machine learning model, trained on a large dataset, capable of predicting the pressure and flow rate for an unseen graph with an average percentage error of 50% based on findings in Table 12.
- Preliminary and final models all underwent validation and testing with great proof of a good fit based on consistency between all training, validation, and testing losses.
- Successful experimentation was conducted and resulted in a 20-fold improvement from the preliminary to the final model. The main source of improvement was the substitution of criterion from MSE to percentage error.

This project successfully introduced the application of graph neural networks in assessing lung ventilation. Its main objective was to predict the nodal pressure and flow rate, which it achieved with an accuracy of 50%. This result was close to the target of 30% accuracy that was established at the start of the project.

Although the desired accuracy was not fully met, the primary purpose of this project was to pave the way for further research in airway flow imaging and establish a foundation for the future use of this technology in diagnosing lung-related illnesses. In light of this overarching goal, the project can still be considered a success, as it demonstrated the capability of GNNs to comprehend the complex patterns observed in the human lungs.

7 References

[1] “Cancer statistics, 2023,” *PupMed,* 2023.

[2] Spencer, “Computer simulations of lung airway structures using data-driven surface modelling techniques,” *Computers in Biology and Medicine,* 2001.

[3] Ewald R Weibel, Morphometry of the human lung, 1963.

[4] M. W. Thomas N. Kipf, SEMI-SUPERVISED CLASSIFICATION WITH GRAPH NEURAL NETWORKS, 2017.

[5] Minsuok Kim, Lobar Ventilation in Patients with COPD Assessed with the Full-Scale Airway Network Flow Model and Xenon-enhanced Dual-Energy CT, Radiology, 2020.

[6] O. Raabe, Tracheobronchial geometry: Human, dog, rat, hamster–a compilation of selected data from the project respiratory tract deposition models, US Energy Research and Development Administration, Division of Biomedical, 1979.

[7] Development of human respiratory airway models: A review, European Journal of Pharmaceutical Sciences, 2020.

[8] Pavlopoulos, Using graph theory to analyze biological networks, 2011.

[9] Whitfield CA, Spectral graph theory efficiently characterizes ventilation heterogeneity in lung airway networks, Journal of the Royal Society Interface, 2020.

[10] Yousef Rammah, A holistic overview of deep learning approach in medical imaging, Multimedia Systems, 2022.

[11] Dami Choi, “On Empirical Comparisons of Optimizers for Deep Learning,” 2019.

[12] J. Brownlee, “Loss and Loss Functions for Training Deep Learning Neural Networks,” *Machine Learning Mastery (2019),* 2019.

[13] “NetworkX,” [Online]. Available: https://networkx.org/documentation/stable/index.html.

[14] J. B. Diederik P. Kingma, “Adam: A Method for Stochastic Optimization,” 2015.

8 Appendix

Appendix 1.

Pressure_Pa	FlowRate_mL	AwayRadius_mm	AcinarID	Points:0	Points:1	Points:2
0	17.718	9.5	1	-6.5725	-35.166	-149.82
1.8248	17.718	9.5	0	-6.6046	-32.441	-158.54
3.6313	17.718	9.5	0	-6.829	-29.523	-167.11
5.4076	17.718	9.5	0	-7.326	-26.333	-175.44
7.1762	17.718	9.5	0	-7.7268	-23.239	-183.78
8.9607	17.718	9.5	0	-7.7268	-20.578	-192.32
10.735	17.718	9.5	0	-7.7268	-18.109	-200.86

First 7 rows of data from patient's lung airway

Appendix 2.

54308		
0	0	1
1	1	2
2	2	3
3	3	4
4	4	5
5	5	6

First 6 connections of graph corresponding to node data

Appendix 3.

```python
# Define input, hidden, and output dimensions
input_dim = 1
hidden_dim = 16  # You can adjust this value based on your requirements
output_dim = 2  # Output features for the second layer (a and b)

# Define the Graph GNN model class
class GraphGNN(nn.Module):
    def __init__(self):
        """
        Initialize the GraphGNN model.

        Parameters:
        - None

        Returns:
        - None
        """
        super(GraphGNN, self).__init__()

        # Define graph convolutional layers
        self.conv1 = GCNConv(input_dim, hidden_dim)
        self.conv2 = GCNConv(hidden_dim, hidden_dim)
        self.conv3 = GCNConv(hidden_dim, hidden_dim)
        self.conv4 = GCNConv(hidden_dim, hidden_dim)
        self.conv5 = GCNConv(hidden_dim, hidden_dim)
        self.conv6 = GCNConv(hidden_dim, output_dim) # Output layer

    def forward(self, x, edge_index, edge_attr):
        """
        Forward pass of the GraphGNN model.

        Parameters:
        - x (Tensor): Node feature matrix.
        - edge_index (LongTensor): Graph edge indices.
        - edge_attr (Tensor): Edge feature matrix.

        Returns:
        - x (Tensor): Output tensor after passing through the GNN layers.

        Note all matrices are torch tensors
        """
        # Forward pass through each layer with ReLU activation
        x = F.relu(self.conv1(x, edge_index, edge_attr))
        x = F.relu(self.conv2(x, edge_index, edge_attr))
        x = F.relu(self.conv3(x, edge_index, edge_attr))
        x = F.relu(self.conv4(x, edge_index, edge_attr))
        x = F.relu(self.conv5(x, edge_index, edge_attr))

        # Output layer without activation (linear transformation)
        x = self.conv6(x, edge_index, edge_attr)

        return x.float()  # Convert the output to float
```