Budapest University of Technology and Economics
Faculty of Electrical Engineering and Informatics
Department of Automation and Applied Informatics

Medical data processing with graph neural networks

Master's Thesis

Author
Anna Gergály

Advisor
dr. Luca Szegletes

May 7, 2025

Contents

HALLGATÓI NYILATKOZAT

Budapest, 2025. május 7.

Gergály Anna
hallgató

Kivonat

Jelen dokumentum egy diplomaterv sablon, amely formai keretet ad a BME Villamosmérnöki és Informatikai Karán végző hallgatók által elkészítendő szakdolgozatnak és diplomatervnek. A sablon használata opcionális. Ez a sablon LaTeX alapú, a *TeXLive* TeX-implementációval és a PDF-LaTeX fordítóval működőképes.

Abstract

This document is a LaTeX-based skeleton for BSc/MSc theses of students at the Electrical Engineering and Informatics Faculty, Budapest University of Technology and Economics. The usage of this skeleton is optional. It has been tested with the *TeXLive* TeX implementation, and it requires the PDF-LaTeX compiler.

Chapter 1

Introduction

1.1 Structure

In the second chapter, I will be introducing machine learning and neural network concepts important to my work, with a special focus on graph neural networks, introducing their key components and grouping them by structure and function. In the second half of the chapter I introduce medical data, legal and ethical questions surrounding it, the types of medical imaging data and give a bit of background on brain fMRI data.

In the third chapter I will introduce the technical background of my work: the programming language, deep learning frameworks and libraries used and document the used versions. In the second half of the chapter I go into the technical parts of working with fMRI data. This includes the used data formats, preprocessing measures and the libraries necessary for these tasks.

In the fourth chapter, I detail the various use-cases of graph networks in fMRI processing that I worked on. For each section I will introduce the corresponding dataset and detail the approach and methodology used in an attempt to solve the problem. I present the results of this approach and the conclusions that can be drawn from the experiment.

In the last chapter, I will be detailing possible ways to enhance the models' capabilities and directions this research could be taken in the future and drawing final conclusions about my work.

Chapter 2

Background

2.1 Graph Neural Networks

Many real life problems can be best modelled with graphs and these representations can code a lot of information if assessed correctly. By developing techniques that work to extract this information and extrapolate from it effectively, we can build powerful systems that can predict, classify and advise based on graph data.

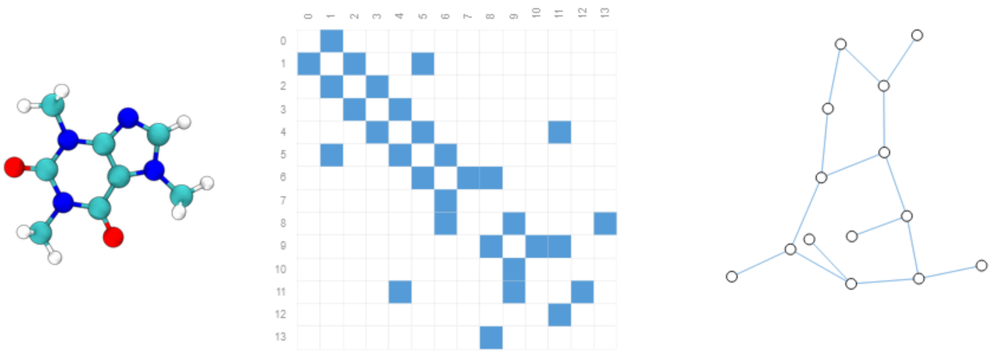


Figure 2.1: From left to right: 3D representation of a caffeine molecule, its adjacency matrix, its graph representation. [29]

Graph algorithms have a long history in mathematics and there is a rich variety of graph related problems that are best solved using these classical discrete mathematical methods: pathfinding algorithms, breadth or depth-first search algorithms are used every day both in everyday IT applications and infrastructure, and also in research. But in certain cases these methods may not be well-equipped to handle the task at hand. With the large amount of data collected every day and the specialized tasks that need to be fulfilled there is reason to bring machine learning into graphs and the discipline has exploded in popularity in the last five years or so.

2.1.1 Neural Networks

Neural networks are an especially interesting and useful area of machine learning and data analysis: as referenced in their name, they were created in resemblance of the human brain's structure, mimicking the interconnected neurons. In recent years they became the flagship of AI research because of their ability to work on incredibly large datasets effectively and produce never seen before results.

The basis of a neural network are neurons which sum up incoming "signals" multiplied by weights and apply an activation function to their output. The network learns by updating these weights and biases based on the training data, to match the desired output to each piece of input. The activation function adds non-linearity to the model, making it capable of learning more complicated relationships in the data.

Neurons are typically organized into layers in a neural network and the models usually benefit from having a large number of layers: modern models for more complex tasks can get very 'deep' which lead to the popularization of the term deep learning when talking about such models. But since having more layers makes a model more computationally intensive, leading to higher costs and slower inference, researchers are often looking to prune models or find architectures that deliver similar results while having a lower parameter count.

Parameters in a model are updated through a process called backpropagation. Here, for labeled training data, the error of the prediction of the model is calculated through the use of a loss function. This should be representative of how far off the model was from the ground truth and also easy to work with derivation-wise and numerically. This is then "propagated back" through the model by deriving what each of the model's parameters contributed to this loss2.1. This is what's known as the gradient and it is used to perform gradient descent: based on the resulting derivative we have a direction to move in to lower the loss.

$$\frac{\partial E}{\partial w_{i,j}} = \frac{\partial E}{\partial o_j}\frac{\partial o_j}{\partial net_j}\frac{\partial net_j}{\partial w_{i,j}}, \tag{2.1}$$

In case of larger datasets it is impossible to calculate the gradient for the entire dataset in one go, so a process called stochastic gradient descent is used, where randomly selected subsets (batches) of the dataset are used to calculate it during training. Having a gradient, the other part needed is the learning rate: how much to change weight in the given direction. This can be considered hyperparameter of the model, but modern optimizers (such as ADAM) use adaptive learning rate optimization techniques (such as momentum and RMSProp) to adjust learning rate during training to ensure a smooth and fast convergence.

From the most fundamental concepts of the original perceptron and the breakthrough idea of using backpropagation, neural networks became a widespread and varied phenomenon: there are architectures for different types of data optimized for different kinds of tasks; these models have proved to be applicable in almost any area.

A very active and widely used branch of neural networks are convolutional neural networks (CNNs). These are most commonly used on image data and work by convolving learnable kernels with the input to extract features. This is useful for capturing spacial relationships allowing for better pattern detection while also greatly decreasing the number of trainable parameters compared to a fully connected feed-forward network. Attention-based networks are another important model class that became very successful in recent times. Transformer architectures are used in many disciplines and are capable of state-of-the-art results. Both of these concepts turn up in graph neural networks as well.

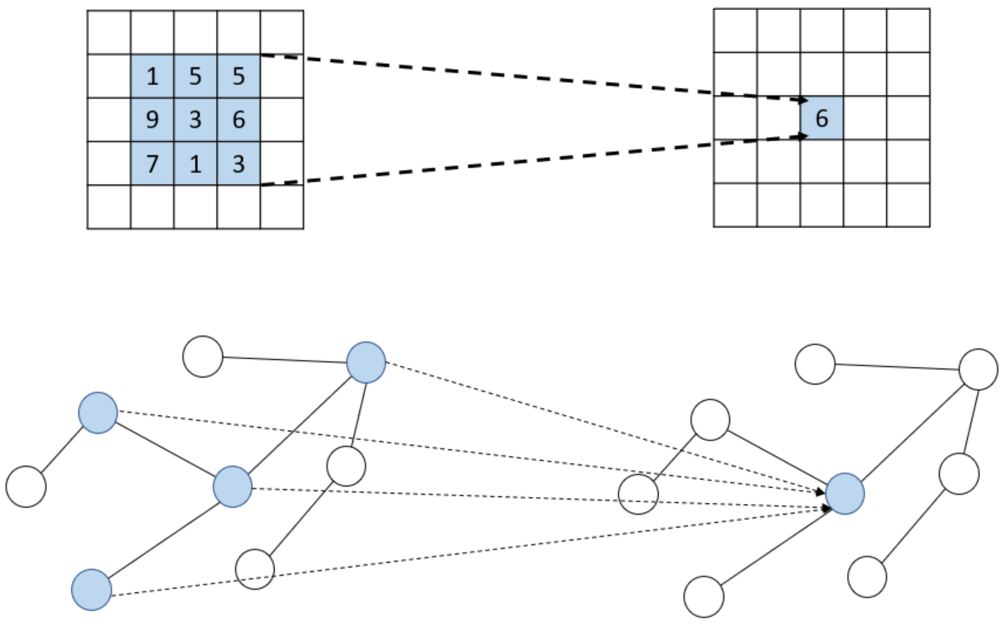


Figure 2.2: Convolution on images versus graphs.

2.1.2 Working on Graph Data

Graphs are a very versatile mathematical concept because of the way they lend themselves very neatly as a generalization. They have great expressive power when it comes to describing relations, groups and things with a rich inner structure. We can find graph-based datasets in a variety of domains: molecules in chemistry, interaction graphs in social sciences, knowledge graphs and computer networks. The level of abstraction they provide makes them suitable for use in many fields, as connections between items or molecules or people are vital in understanding complex natural systems.

One area that is particularly rich with such examples is biology and medicine. Interactions between drugs, relationships between species and contact tracing in epidemiology are all important facets that can benefit from the ability to analyse graph datasets. We can also find interesting networks to analyse inside of organisms: the focus of this thesis is the analysis of network formed by brain regions and the interactions that happen in the human brain.

Graphs are mathematically defined as a set of nodes (or vertices) and edges (or links) which run between two nodes. They can be categorized by their edge structure: directed graphs specifically code the information of a starting and end node, while undirected graphs do not. In terms of graph neural networks this is a very important distinction as it fundamentally alters how the network's meaning. Certain types of networks are designed with a specific type of graph in mind and might need special normalisation.

Certain types of graphs with special properties can be especially interesting in certain areas. Trees (which contain no cycles) can be useful for describing containment hierarchies or sentence structures, for example. Based on semantic meaning we can also talk about heterogeneous graphs were nodes may represent entirely different things, purchase or interaction graphs in a recommender system where a vertex could be a user or an item to be purchased. In this case it is important to make this information available to the network.

A graph dataset often codes much more information then just the mathematical structure itself. Ideally some information is available of each node; a feature descriptor vector for each node supplies more information for the network and also helps identify the node it is attached to: since graphs are order agnostic the model needs another way to identify which node is which.

The contents of such a descriptor vector must be domain specific. For example in case of a social network it would code information about a given person: age, gender, height. In case of a molecule prediction scenario it could be atomic weight or covalent radius. For the brain fMRI analysis case most relevant to this thesis, it could be information about a given brain region, average activation or even an activation timeseries.

In case there is no suitable information available for the node feature vectors it is common practice to use rows of the identity matrix: in absence of additional data this can be useful to serve as the identification tool the network needs. In some cases information might be available in other ways, for example as edge weights or edge feature vectors. These could be descriptive of the users' interactions in a social network example.

Node feature vectors can be concatenated together to form a matrix and together with the adjacency matrix (or edge weight matrix) of the graph they form a very neat representation of the graph. This makes the highly variable structure of the graph containable within a constant, well-structured manner that is crucial for machine learning applications.

But since graphs can be used to represent complex structures and hierarchies, where this inner construction is more pronounced and relevant then in other cases, they require specialized methods when it comes to machine learning and neural networks. A lot of these methods take after image recognition or object detection solutions.

The reason for this is that while images do not have explicitly structured data in the same way as graphs do, their contents can be in complex spatial relations with each other. A computer vision model needs to take these into account when performing complex tasks such as segmentation or creating a full description of what is happening in a photo. Even for simple object detection, more complex object have hierarchical structures that the model needs to 'understand' in some way.

2.1.2.1 Common tasks

The motivation behind working with graph datasets is of course to perform some sort of inference using the patterns extracted from the training data. Since these datasets are diverse both in semantics and in structure this could mean a lot of different types of tasks. In this section the most common of these will be described with examples and commonly used methods for solving them.

Most graph architectures work by assigning a vector to each node on their output. This is often called a node embedding: an n-dimensional vector is attached to the node that attempts to convey all available information about the node and its role in the graph structure, effectively embedding it in an n-dimensional space. Node embeddings are not strictly only generated in deep learning models, there are for example random-walk based methods for this purpose, such as node2vec.

In case of deep learning models the creation of node embedding is an iterative process as there is a new vector attached to the node after every layer. In this process the original feature vectors can be thought of as a sort of 0th iteration of the embedding, purely representing information about the node itself. With more and more layers aggregating information on a larger and larger neighbourhood of a node this slowly transforms into containing information of the structure of the network as well.

Node prediction. In this type of task the goal is to infer some information about individual nodes of the graph. This could be sorting into already existing categories, classification or estimating a value related to the node, regression. Both types of tasks can work with datasets that are either graphs made up of labeled nodes that have a certain

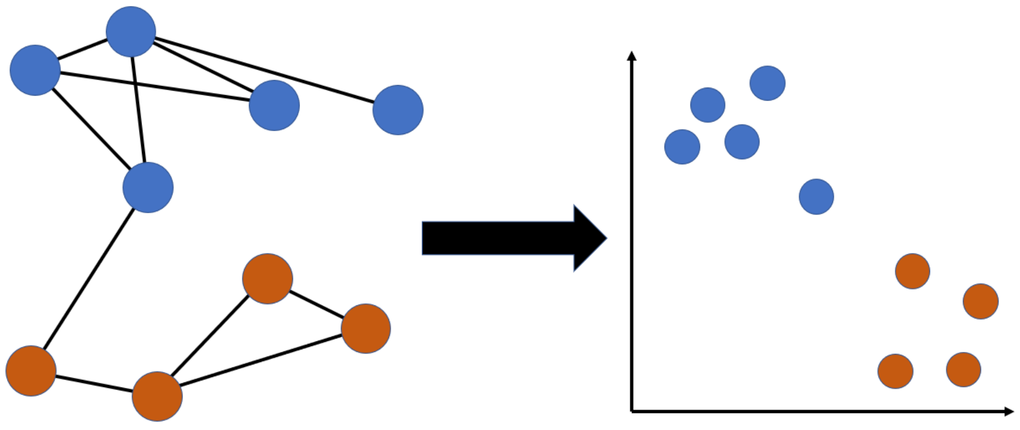


Figure 2.3: Visual representation of node embeddings in 2 dimensional space.

number of missing values or with training graphs that have fully labeled nodes, performing inference on graphs the model has not seen before. The second type of task is much more difficult, meaning models often performs significantly worse on such tasks, but oftentimes this reflects the real world scenarios and use-cases for models better.

Since the network output is representative of a node and its role in the structure in the first place, this is the most straightforward type of task to solve using a deep learning model. In case of node classification a commonly used technique is to match the dimensions of the final embedding to the number of possible classes and have each of the values represent probability of class membership: this can work for both multi-label and simple classification using different activation functions.

Another technique is to use a small fully connected MLP on the embedding for prediction. This is also useful in case of regression and gives a bit more control over the possible output formats.

Graph prediction. In case of this task the goal is to predict some sort of information about the entire graph. Node embeddings can be aggregated in a lot of ways to predict graph classes or labels. These aggregation methods are commonly referred to as a 'readout function': the values attached to nodes are 'read' together to reveal information about the entire graph.

More common, basic readout methods are similar to the simpler pooling methods used in CNNs: node features can be averaged, summed or there can be a minimum or maximum choosing operation.

Edge prediction. There are two main types of edge prediction: either, similarly to nodes there is a specific feature or classification of edges we want to predict or the goal is to find edges in the graph that are not present in the structure right now but are most likely to exist; the second type is a very common task in recommender systems.

There are multiple commonly used techniques for edge predictions using node embeddings generated by a GNN there are capable of providing either continuos or binary results. These methods usually rely on using the information about a pair of nodes to determine their 'compatibility' or other feature of their shared edge.

Clustering, influence prediction. Clustering in this domain usually refers to clustering the nodes of the graph to find groups more 'similar' or 'closely connected'. There are a lot of classical solutions for this problem as well such as the Girvan-Newman algorithm which optimises an edge-betweenness metric and the Louvain method with uses modularity.

In case a GNN is used in this task it is in an unsupervised or semi-supervised manner, where it used to generate embeddings and then the embedding are clustered using a common clustering ML solution such as K-means or DBSCAN. Hybrid solutions are also possible: a neural network could be used to create edge weights that a classical algorithm could incorporate.

Influence prediction is also an unsupervised task that seeks to find a node or multiple nodes that have the most influential role in the graph, finding hotspots of activity

2.1.2.2 Graph convolutions

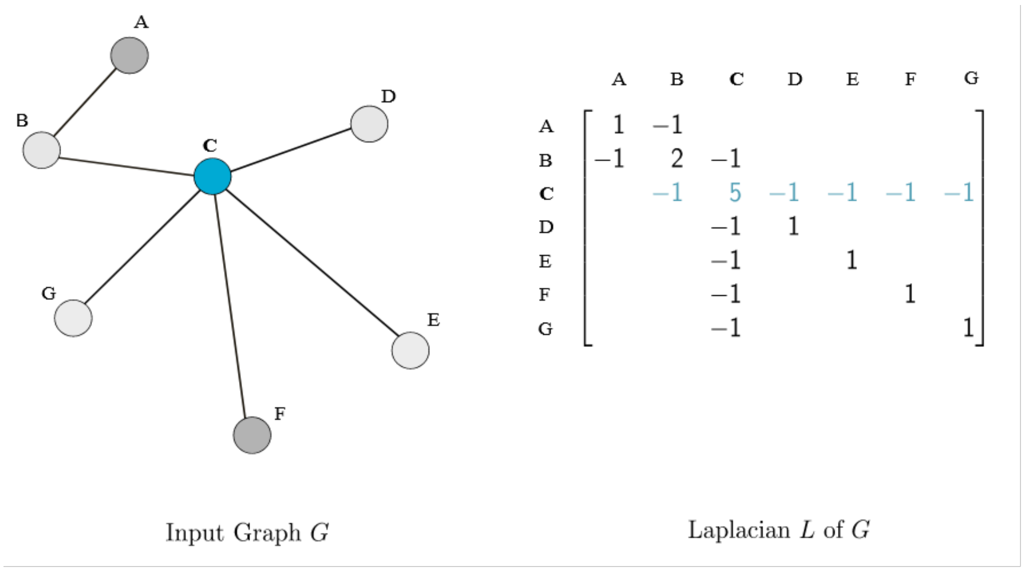
The Laplacian of a graph is a square matrix $n \times n$ in size (where n is the number of nodes in the graph). It can be calculated from the adjacency matrix and diagonal node degree matrix:

$$L = D - A,$$

where L is the Laplacian, D is the diagonal node degree matrix and A is the adjacency matrix. In the diagonal node degree matrix there is similar to an identity matrix but in each row the instead of one we have the degree of the node corresponding to said row. The Laplacian can be used to build polynomials that can be used on node features.

$$p_w = w_0 I_n + w_1 L + w_2 L^2 + \cdots + w_d L^d = \sum_{i=0}^{d} w_i L$$

These polynomials can be thought of as analogues of 'filters' in CNNs, and the coefficients w_i as the weights of the 'filters'. There exists a type of network that built directly on the concept of Laplacian polynomials: ChebNet used Chebyshev polynomials and normalized Laplacians to create a more numerically stable and 'stackable'.



$$\begin{array}{c|ccccccc} & A & B & C & D & E & F & G \\ \hline A & 1 & -1 & & & & & \\ B & -1 & 2 & -1 & & & & \\ C & & -1 & 5 & -1 & -1 & -1 & -1 \\ D & & & -1 & 1 & & & \\ E & & & -1 & & 1 & & \\ F & & & -1 & & & 1 & \\ G & & & -1 & & & & 1 \end{array}$$

Figure 2.4: The Laplacian L for an undirected graph G, with the row corresponding to node C highlighted. Zeros in L are not displayed above. The Laplacian L depends only on the structure of the graph G, not on any node features. [8]

ChebNet was a big step in ushering research into graph networks, as it motivated many to think about graph convolutions from a different perspective. Current graph neural networks utilize graph convolutions in ways that make use of 'computational shortcuts': bypassing costly operations such as eigenvalue calculations can make a model much more scalable, both in terms of data throughput and model size.

These shortcuts often mean calculating local approximations of the graph convolutions. Local graph operations are often so called 'message passing' operations: every node in the graph sends a 'message' to its neighbours based on its own data and then each node aggregates the received messages. In practice this means each node vector is updated in a layer based on its neighbours.

2.1.3 Graph Neural Network Types

There are many architectures that fall under the umbrella of graph neural networks and they can be grouped by many different criteria with interesting similarities between them. In [39] the authors break down the most common parts of graph architectures: a way of **propagating** information between nodes, a way of **sampling** relevant nodes when graphs and neighbourhoods get large, and, in case it is needed for the task at hand, a way of **pooling** information to represent subgraphs or graphs.

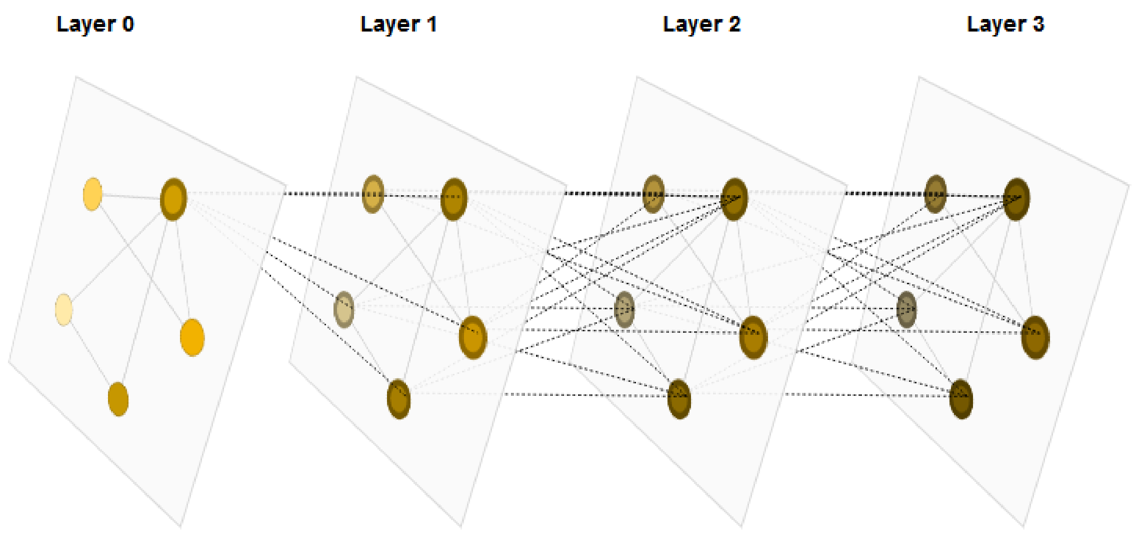


Figure 2.5: Visualization of a GNN accumulating information via message passing. [29]

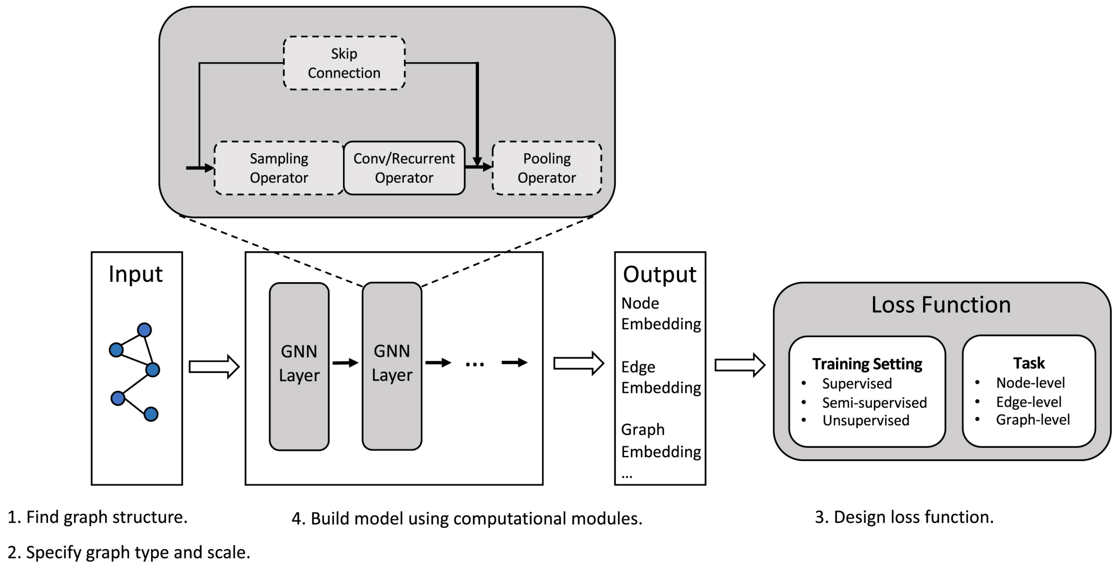


Figure 2.6: A generalized schematic of GNN architectures [39]

We can also differentiate architectures based on the type of graphs they operate on, the previously discussed task of the network2.1.2.1 or the used learning type: supervised (using labeled data), semi-supervised (using a small amount of labeled data to guide training) and unsupervised (using unlabeled data to find patterns).

The most commonly used network types/layer types include GCNs (Graph Convolutional Network), GATs (Graph Attention Network), GraphSAGE (Graph Sample and AGgregatE). These are all convolution-based solutions with different approaches to the implementation. GCNs try to stay true to spectral convolutions following ChebNet's footsteps, GATs leverage the very powerful concept of attention for convolutions and GraphSAGE focuses on building a good way to sample large neighbourhoods in large graphs.

In the realm of convolution based models MPNNs (Message Passing Neural Network) can be seen as a generalization of the concept, extracting common features from other methods, serving as a framework that can instantiate any of those models. Other than convolutions another type of architecture that has been used in graph networks with success has been recurrent networks.

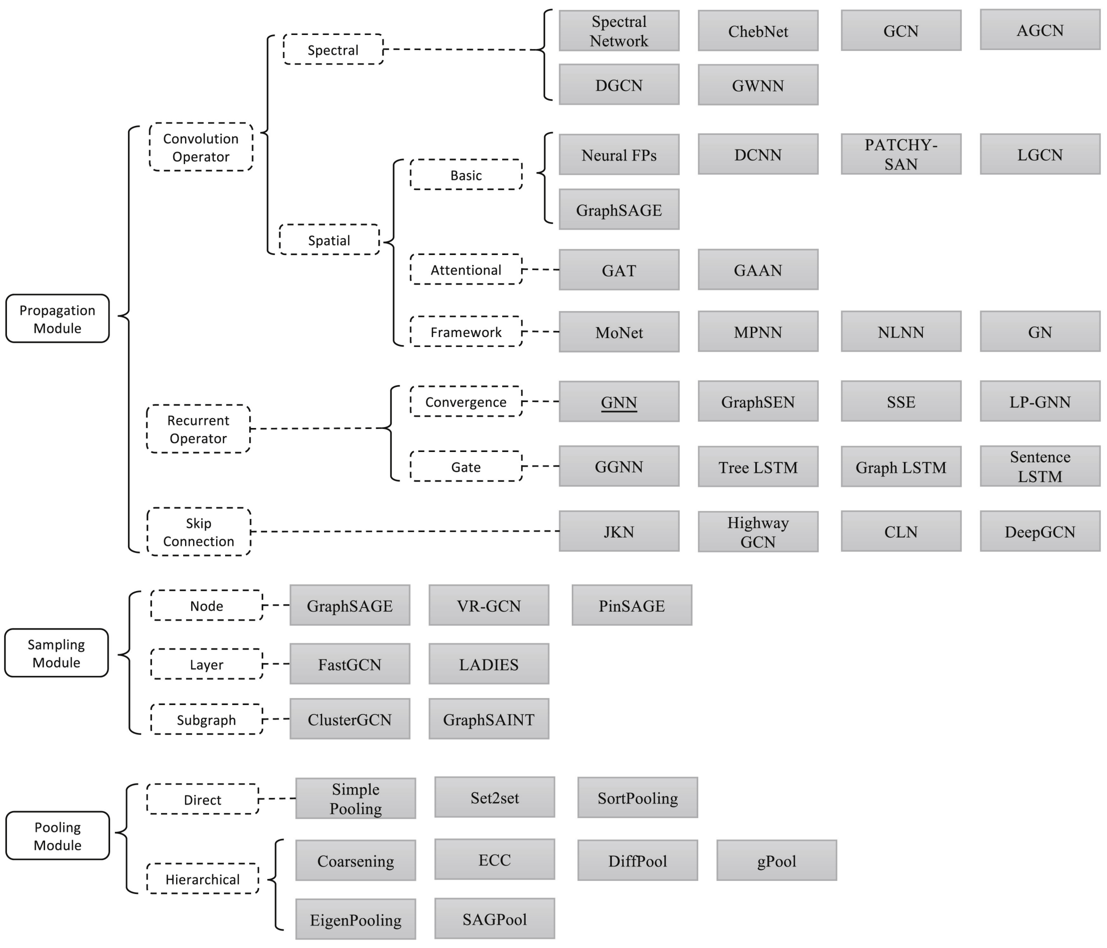


Figure 2.7: GNN architectures grouped by characteristics[39]

2.1.3.1 Graph Convolutional Networks

Graph Convolutional Networks (GCNs) expand on the idea of CNNs, which are commonly used in image processing. Observing from a graph perspective, an image can be considered a very special case of a graph, where pixels are nodes and their neighbours can be determined from the grid. This type of network generalizes the concept of local convolutions from the image domain to general graphs.

$$H^{(l+1)} = f\left(H^{(l)}, A\right) = g\left(\hat{D}^{-\frac{1}{2}}\hat{A}\hat{D}^{-\frac{1}{2}}H^{(l)}W^{(l)}\right) \tag{2.2}$$

Equation 2.2 shows how the next layer's node feature vectors are calculated from the previous ones: A is the adjacency matrix, $\hat{A}$ is the adjacency matrix with self-loops added in ($\hat{A} = A + I$), $\hat{D}$ is the diagonal node degree matrix with the purpose of normalisation, W is the learnable weight matrix and g is some type of non-linear activation function, most commonly ReLU or leaky ReLU in practice.

This computation model is a nice analogue for convolutions on images, and approximates spectral convolutions while remaining computationally stable and non-resource intensive. Layers can also be stacked nicely to allow information to travel further than the immediate neighbourhood of a node.

2.1.3.2 Graph Attention Networks

Graph Attention Networks (GATs) leverage masked self-attention layers to address problems present in earlier architectures of graph convolutional networks[35]. By using the attention to implicitly assign different weights to nodes in a neighbourhood, the model can gain a deeper understanding of the graph structure without the use of costly matrix operations.

In a layer of a GAT architecture input node features ($h \in \mathbb{R}^{N\times F}$, where N is the number of nodes and F is the number of features) are turned into output node features by first applying a linear transformation via a weight matrix ($W \in \mathbb{R}^{F'\times F}$) and then performing self-attention (a) on the nodes. The full equation is as follows:

$$h_i' = \sigma(\sum_{j\in\mathcal{N}_i} \text{softmax}(a(Wh_i, Wh_j))Wh_j)$$

Where σ is the non-linear activation function and h_i and h_j refer to the node features belonging to node i and j. Self-attention coefficients $e_{ij} = a(Wh_i, Wh_j)$ indicate the importance of node j's features to node i. If the model would allow every node to attend every node, that would mean dropping all structural information. To avoid this, attention is masked: e_{ij} is only computed for nodes j that are in the neighbourhood of node i ($j \in \mathcal{N}_i$). This is generally the first-order neighbourhood, but it can be parametrized.

The attention mechanism is a feedforward neural network, parametrized by the concatenated transformed node features of the two nodes. The calculated coefficients are then normalized across the choices of j using the softmax function. Once obtained, the normalized coefficients are used to compute the linear combination of the features corresponding to them, resulting in the final ouput of the layer (after applying an activation function). Multi-headed attention is commonly used to stabilize the learning process.

2.1.3.3 GraphSAGE

GraphSAGE (graph SAmple and AggregatE) was created as a general inductive framework to leverage node feature information and efficiently generate node embeddings for previously unseen data[15]. Each node of the graph is represented as some aggregation of its neighbors, therefore, even if a new unseen node is encountered, it can be represented as some aggregation of its neighbourhood.

GraphSAGE aggregates information from the neighbours of nodes similarly to other architectures, but also takes a step to sample this neighbourhood: a fixed-size set gets uniformly sampled from the available neighbourhood. This is resampled at each iteration when calculating embeddings. Weight matrices are learned using gradient descent based on random walks: the graph-based loss function encourages nearby nodes to have similar representations, while enforcing that the representations of disparate nodes are highly distinct.

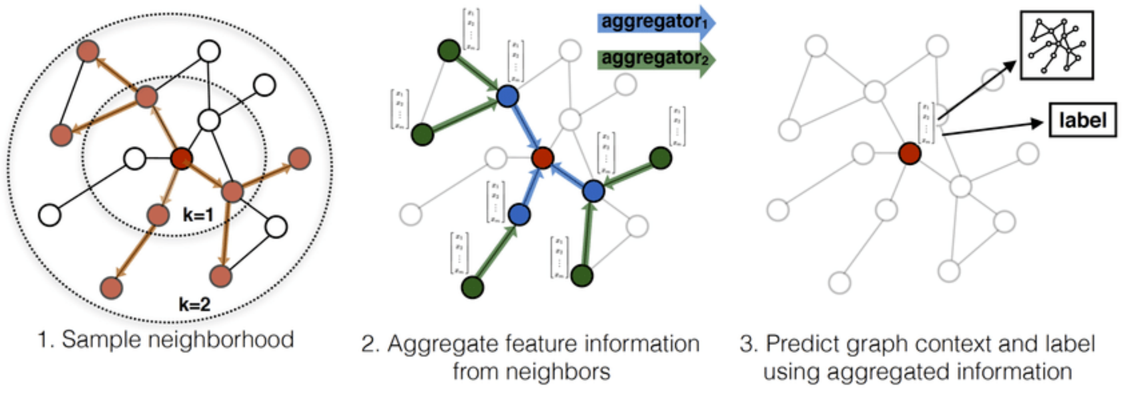

k=1
k=2
1. Sample neighborhood
aggregator₁
aggregator₂
2. Aggregate feature information from neighbors
label
3. Predict graph context and label using aggregated information

Figure 2.8: Visual illustration of the GraphSAGE sample and aggregate approach.[15]

The aggregation method from the neighbourhood is an important feature in a GraphSAGE model, a symmetrical function is most desired, since node neighbours do not have a set ordering. Based on the used aggregation function GraphSAGE can be very similar to GCNs: the mean aggregation method results in the convolution propagation rule of GCNs.

2.2 Medical data

We can call anything medical data that relates to human healthcare in one way or another. This most commonly means data collected by healthcare institutions about patients using sensor equipment, but it could also be information about a patient's habits or general environment, data from drug test trials or even location data in case of epidemiological contact tracing.

2.2.1 Legal and ethical concerns

This kind of data is often very personal and regarded as sensitive data: people are entitled to equal treatment regardless of medical status and as such information about the health status of an individual is protected. Many patients do not want employers or other third parties who could use such information against them to know details about their conditions and their treatment.

For this reason doctors are required to not give out patient information unless it is strictly necessary and/or they have the informed consent of the person. This puts medical researchers in an interesting position; medical researchers using machine learning even more so. Data collected specifically for experiments where subjects can consent to their data being used is a straightforward situation, but there is data collected every day, worldwide in hospitals that could be very useful for furthering medicine, but doing so poses data privacy concerns. This is especially important in case of machine learning and neural network research as these disciplines require a very large amount of data that is very hard to collect using only organized experiments.

Developments in IT have lead to concerns about the effectiveness of data protection laws and in the European Union, for the sake of a more consistent and comprehensive protection of personal data, a General Data Protection Regulation (GDPR) has been enacted[10]. There are fears that the combination of strict requirements and very limited research exemptions will stifle and severely restrict medical research, and that the current consent or anonymise approach is not suitable for data-intensive research[22].

On the consent side of the equation, there is great difficulty in obtaining consent for personal data to be available for linkage, re-use and analysis for largely undetermined future research purposes[3]. It is a question whether meaningful or freely given and informed consent can be given at the time of data collection as it may not be possible to foresee or comprehend the possible consequences of consenting: a person consenting to a DNA sample has no way of knowing what could be determined from their sample in 10 years time.

Since personal data is defined as data that can directly or indirectly identify the individual[6], the act of anonymisation seeks to remove identifiers that link datapoints to specific individuals to make the data suitable for use in research without infringing any of the patients' rights. A simple deletion of the name and address of the subject are usually not enough, as other characteristics can be enough to identify an individual. A commonly used technique is pseudonymisation where identifiers are removed and replaced with a unique keycode, but this is not truly anonymous as someone with the requisite key can link it back to the individual[28].

Anonymisation requires extensive stripping of datasets and makes data linkage and update impossible in almost all scenarios, which makes datasets much less useful for research networks and projects. In addition to this, even in anonymised datasets there can be information that could negatively affect groups and lead to discrimination and stigmatisation[21].

There is currently new EU regulations underway for the construction of a European Health Data Space Regulation (EHDS), which aims to establish a common framework for the use and exchange of health data across the EU[11]. This seeks to empower individuals to be in charge of their medical data and to enable secure and trustworthy reuse for research and innovation.

2.2.2 Types of medical data

We can organize most medical/healthcare data into 7 categories that have differing use-cases and are of different interest to researchers[1]. Electronic Health Records (EHR) encompass the digital version of a patient's chart in a doctor's office, containing medical and treatment histories, but take the concept a step further offering a comprehensive view of patient health. Administrative data is similarly collected at clinical offices, detailing admissions, home care and prescriptions.

Claims data refers to the data that is transferred to the insurance provider (public or private) for coverage and compensation. This contains information about procedures and tests as well as costs. Patient/disease registries help collect secondary data associated specifically with patients who share a particular diagnosis. These are particularly important for chronic conditions such as cancer or diabetes.

Health surveys are conducted in the general population to assess the health of the population and estimate the prevalence of certain diseases. Clinical trial data refers to data collected in experiments and studies specifically for the purpose of research. Genomic data involves the molecular sequence in an organism's genes, the role of each gene, the regulatory elements managing gene expression, and the connections among various genes.

All of these types of data have their place in healthcare research and they occasionally overlap in their subjects, but they generally fall under different providers and legal protections. The data types most commonly used in machine learning and big data applications are EHRs, aggregated clinical trial data, administrative data, genomic data[22] and medical imaging data[33].

2.2.3 Medical imaging

Non-invasive medical imaging techniques, such as MRI, X-ray, and CT scans, are essential tools in medicine, allowing physicians to understand the internal structures and functions of the body without the need for surgery. These techniques are invaluable for their ability to produce high-resolution images of different types of tissues, making it ideal for assessing the state and function of internal organs[18].

These techniques vary by advantages, disadvantages, risks, image quality, safety, costs and applications. Choosing the right technique for the examination of a particular patient or for data collection in an experiment is delicate balancing act that requires a physician well-versed in the particular niche.

The basic concept for a medical imaging system consists of a source of energy that can penetrate the human body and as the energy passes through the body it is absorbed or deflected at varying levels according to the density and structure of tissues and organs. The energy is then detected by special detectors compatible with the energy source and the signals are then converted into images using the correct algorithm matching the system. Techniques are often categorized based on the type of energy passing through the body.

The earliest discovered medical imaging tool was the X-ray, discovered by Röntgen in 1985, followed by the CT imaging technique in 1963. Research into the MRI started in the 1970s and the first prototypes were tested in 1980.

2.2.4 MRI and fMRI imaging

Magnetic Resonance Imaging (MRI) is a diagnostic technology that uses magnetic and radio frequency fields to image tissues[4]. It produces high-fidelity images, while using no ionizing radiation, making possible its repeated use in patients.

The procedure can be performed without contrast (no possible allergy problems) and is painless, but due to the long scan time and the design of machines it can induce claustrophobia and young children might require sedation to remain still. Comparatively to other imaging techniques MRI is also rather expensive, which is an important factor in deep learning applications where a large volume of data is needed.

MRI is well suited for examining abdominal organs, such as the liver, joints and finding tumours, cysts and other abnormalities and unhealthy tissue in various parts of the body. It is also good for providing a view of the cranium and examining the brain and the spinal cord.

Functional MRI (fMRI) extends the capabilities of MRI by measuring brain activity, via a method called Blood Oxygen Level Dependant Constant[14]. This technique works because of the haemodynamic response: when a part of the brain is activated, blood flows

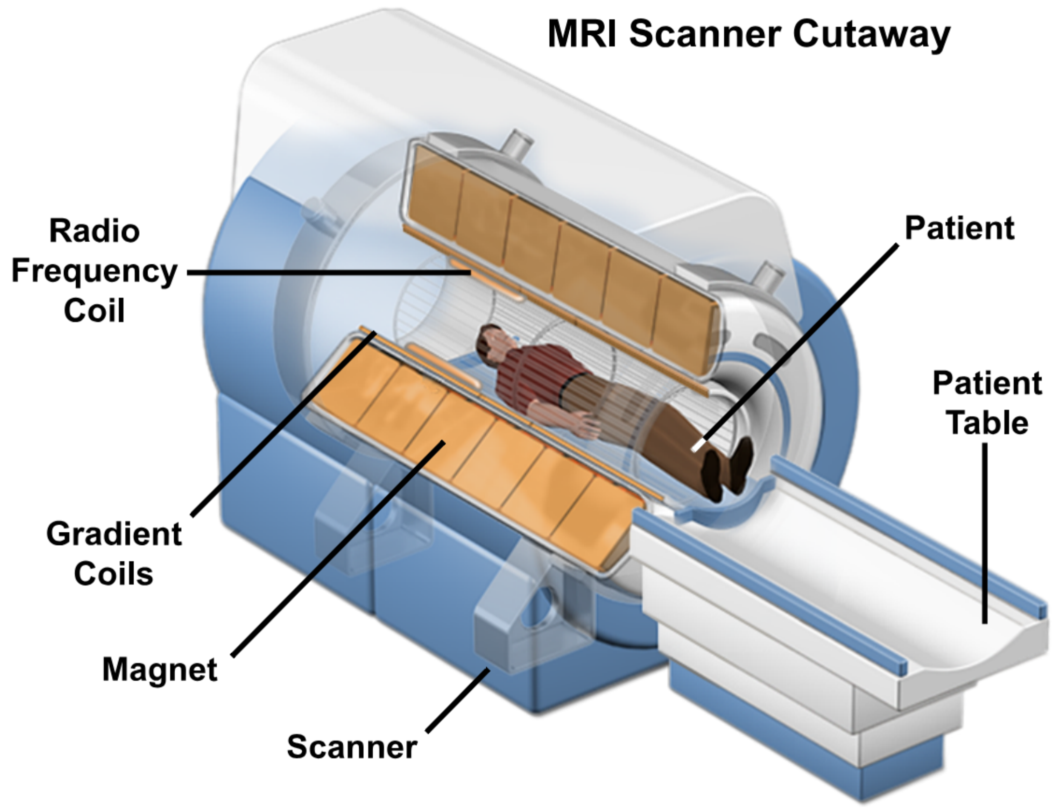


Figure 2.9: Schematic of an MRI machine[7]

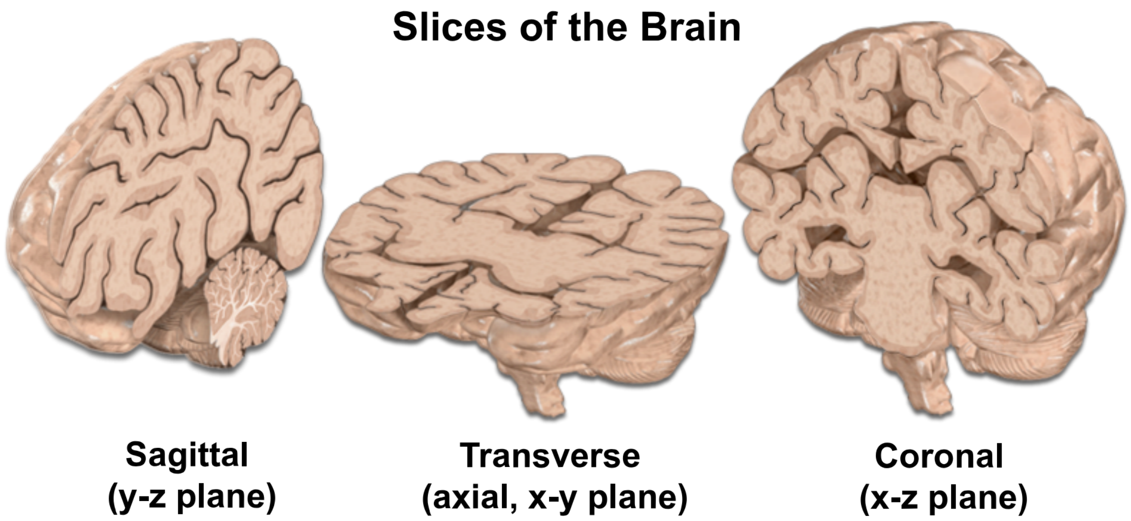


Figure 2.10: Types of slices in brain MRIs[7]

the region (after an initial dip). The surge of oxygenated blood increases the ratio of oxygenated hemoglobin.

Oxygenated hemoglobin has different magnetic properties than deoxygenated hemoglobin, which causes measurable changes in the magnetic field, thus changing the readings of the MRI in the activated region, thus allowing detection of neural activity. The ability to capture neural activity in a non-invasive way is incredibly valuable in both clinical and research settings.

There are two main types of fMRI scans: resting state and task-based fMRI. Resting state is self explanatory: patients are asked to remain still and relax while the scan is taken. For task-based scans, patients are given some sort of stimulus: either a task to complete, such as imagining moving a part of their body (for examining motor function activity) or holding a conversation, or simply being shown an image or text.

2.2.4.1 Clinical significance

2.2.4.2 Technical background

MRI technology builds on the physical phenomenon known as Nuclear Magnetic Resonance. Certain atomic nuclei, when exposed to a strong magnetic field, align themselves with this field. The hydrogen atom also shares this property and is abundant in all parts of the human body.

The aligned nuclei can then be perturbed by a radio frequency signal, which causes them to absorb the energy of the signal and go out of alignment with the original field. The nuclei then start to precess, during which they emit the absorbed energy, which is unique in different types of tissues in the body[25].

MRI machines provide a strong, constant magnetic field, then send short, localized pulses of radio frequency signals to cause the misalignment. As these atoms realign themselves the machine uses the signals emitted to construct a three dimensional image of the area, comprised of individual cuboid elements, called voxels, based on the measurements.

The number and size of voxels reflects the quality of the scan and high-resolution images permit the detailed analysis of brain responses. fMRIs recorded this way reflect neural activity: the presented task increases local neuron activity, which in turn increases metabolic rate, which also means increased blood flow, which translates into an increased MRI signal. The time it takes for this process to occur also means that the changes in the fMRI are delayed compared to the stimuli of the patient by 1-2 seconds[9].

2.2.4.3 Processing

An fMRI data set from a single session can either be thought of as t volumes, one taken every few seconds, or as v voxels, each with an associated time series of t time points[30]. This raw four-dimensional data needs to be thoroughly preprocessed as a first step, before any meaningful conclusions can be drawn from it. The beginning of this procedure is very similar to other image or medical imaging preprocessing methods. There is however

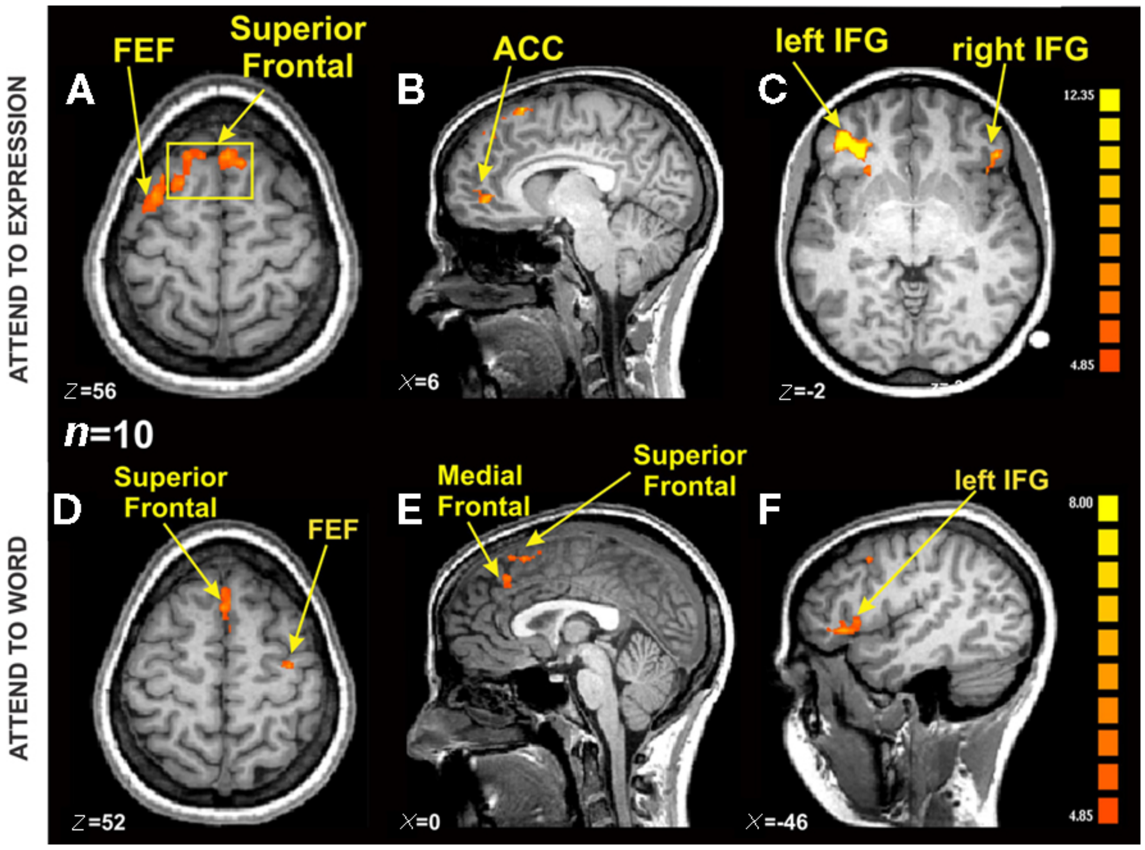


Figure 2.11: Example of an fMRI scan showing different activations on expressions and speech[24]

no consensus currently on the best methods for preprocessing fMRI data, so picking a strategy and parameters often works on a case-by-case basis, there are a few well-known pipelines that are available and commonly experimented with[**?**].

Despite their differences most pipelines accomplish most of the same few tasks: Slice-timing correction makes sure that when viewing a volume all of the voxels have been acquired at the same time, motion correction is applied to make sure each volume is aligned with every other one. Smoothing methods are applied to reduce noise and artifacts. Some methods also try to reduce variance in the scans by aligning them to a particular template.

2.2.4.4 ROIs and connectomes

For better analysis of the functional brain scans it is important to have a certain understanding of the functional areas of the brain to help understand the significance of certain regions firing. Brain atlases are sophisticated templates created by medical researchers as the average of many brains, giving a common coordinate system and also important information about the region at every voxel. With these atlases brain scans can be segmented into different functional centers which are commonly referred to as Regions of Interest (ROIs).

If the segmentation is completed for an entire scan, the average activation of these regions can be computed and used to draw conclusions about how this brain region works normally or how it participates in a given task. Even more information can be extracted if we are able to analyse how the regions interact with each other[26].

The correlations between the fluctuations of activation in regions can reveal information about how these segments of the brain interact. These correlations change depending on the subject or the performed task, showing that regions interact differently under different conditions. Several methods have been proposed for assessing connectivity, most commonly using the time series of ROI activations to calculate covariance or correlation.

The goal is to yield a complete matrix of connectivity values between the regions, which can be thought of as sort of weighted brain graph describing how connected these regions are, often referred to as connectomes. A typical brain connectome is comprised of nodes for each brain ROI and edge weights that denote the connectivity strength between two ROIs[5].

Chapter 3

Technical background

In my work I used the Python programming language both for implementing machine learning solutions and for creating pipelines; downloading, sorting and preparing data, as well as various scripting needs. For running deep learning tasks I have employed the Google Colab[1], an online compute platform that lets users run Jupyter notebooks with limitations, one of my own laptops, equipped with an Nvidia GTX 1050Ti portable, and for the particularly resource intensive diffusion task4.2 I have received help from a corporate compute cluster.

3.1 Used Frameworks and Packages

For machine learning tasks I have employed various libraries commonly used for such tasks to avoid reimplementing standard solutions, both for creating the different models for comparison and for data exploration and visualization purposes.

For the purpose of organizing the used Python packages and ensuring a consistent environment I have used a various virtual environment management tools in my work: venv, uv and conda. In these types of projects package management is a common painpoint. Since machine learning is a very active field with new discoveries and optimizations packages are updated quite often. Backwards compatibility is not guaranteed usually and issues of certain packages needing another package in a certain version, but not being labeled accurately come up commonly.

Even when all needed package versions are documented correctly in a project, there can still be problems for people trying to recreate the results. Older package versions often become unsupported and unavailable, and package managers might disallow certain combinations of installations.

[1]`https://colab.research.google.com/`

3.1.1 Main Deep Learning Framework

The main base deep learning framework I have used in all of my work is Pytorch. Pytorch is one of the most popular foundational machine learning libraries (alongside TensorFlow) used for a wide variety of applications, such as computer vision or NLP.

At it's core is the provided tensor computing capabilities and an automatic differentiation system, which enables backpropagation. Moreover, Pytorch also has a neural networks module with a lot of widely used lower level concepts: abstract model class, basic layers (e. g.: fully connected, convolutional), optimizers.

3.1.2 Graph Neural Network Frameworks

It is completely possible to implement graph neural network layers and complete graph networks only using the Pytorch library and I have also done so, but there are also frameworks built to enable users to create these networks in just a few lines. These libraries have implementations of commonly used GNN layers (such as GCN or GAT) and usually have their own format for graph storage. This can speed up model creation greatly but working purely from Pytorch offers the greatest flexibility.

I have tried to specific libraries in my work: DGL and PyG. DGL (Deep Graph Library)is a framework agnostic and scalable solution. It represents graphs with a DGLGraph object which stores all graphs as directed using an edge list. It is also possible to add both edge and node features.

PyG (Pytorch Geometric) was specifically created to enhance Pytorch with better GNN capabilities. It aims to follow the design principles of vanilla Pytorch and provide a good interface to both researchers and first-time users.

3.1.3 Other Machine Learning Tools

Sklearn (Scikit Learn) is a comprehensive library of machine learning solutions, offering a wide variety of models for classification, regression, clustering and dimensionality reduction, as well as model selection and preprocessing tools.

XGBoost (eXtreme Gradient Boosting) is a gradient boosting library that provides a well optimized and very performant gradient boosting machine implementation.

Denoising-diffusion-pytorch is a package commonly used for experimenting and training diffusion models. It provides an easy interface to train diffusion models with custom networks and enables easy DDIM sampling.

3.1.4 Supporting packages

Numpy is a commonly used computing framework with great features regarding N-dimensional arrays. Most machine learning use it to some degree and are compatible with its formats.

Matplotlib is a powerful plotting and visualisation library for Python. It helps in creating easy to understand and colourful graphs from data. Seaborn is an interface to use matplotlib for complex statistical graphics.

Nibabel and nilearn are specifically neuroimaging libraries: the first one for handling the associated file formats and providing access to neuroimages and the latter for enabling analysis of brain volumes.

Tsfresh is a tool designed to calculate time series characteristics. It also contains methods to evaluate the importance of these features for regression and classification tasks.

3.2 Using fMRI data

fMRIs produce a very special four dimensional data mass, that is not easily readable by default in most cases. Fortunately there have been specific data formats and libraries created to enable researchers to transport, view and extract information from these scans.

3.2.1 Data format

NIFTI (Neuroimaging Informatics Technology Initiative) is a file format created to store brain imaging data[38]. It was agreed upon as a replace the previous ANALYZE format, which was widespread but had problems: the main problem, that NIFTI aimed to solve was that there was not adequate information about orientation.

Instead of separate files for meta-information and the actual image (like in ANALYZE), this format allows storage as a single .nii file. Since there are very often large areas of a single colour or masked sections these files can be compressed with great results. Most available datasets are downloadable as nii.gz files.

The first three dimensions are reserved for spatial dimensions (x, y and z) and the fourth one is for time, t. The remaining dimensions from fifth to seventh can be used to store other information. The fifth dimension, however, can still have some predefined uses, such as to store voxel-specific distributional parameters or to hold vector-based data.

The first 348 bytes (or for the later/current version NIFTI-2 500 bytes) of the file are reserved for the metadata header which is full of information on how to interpret the actual data in the file. Great care has been to taken to make ANALYZE, NIFTI-1 and

NIFTI-2 as compatible with each other as possible, while advancing the user experience and representation capabilities of the file format. Nibabel is compatible with all of these formats, providing an easy way to access this data from Python.

3.2.2 Extracting ROI Values and Connectomes

Using the nilearn package it is possible to load brain atlases and use them to segment the brain scan loaded from a NIFTI file using nibabel. It is also possible to create masks and use only certain parts of the brain/atlas. This allows for the creation of average ROI activation timeseries for each region. This means we end up with a tensor that is number of patients $\times$ timesteps $\times$ number of ROIs in size.

Nilearn also has tools to perform the approximation of connectomes from these timeseries. The ConnectivityMeasure class capable of calculating various types of functional connectivity matrixes on multiple subjects. It can be parameterized with a covariance estimator object for more control over the resulting connectomes.

Chapter 4

Methods and Implementation

4.1 Graph Neural Networks on Connectomes

4.1.1 Dataset

The ABIDE (Autism Brain Imaging Data Exchange) dataset focuses on the furthering research into ASD (autism spectrum disorder) through neuro-imaging and neuroscience. The dataset contains in total 1112 resting state fMRIs of subjects both with ASD (539 individuals) and typically developing controls (573 individuals).

The imaging data has been collected by 16 institutions, who collaborated to create a publicly accessible anonymised dataset to allow the broader scientific community to take part in ASD related research while preserving the privacy of the participants. As such, datasets do not contain any protected health information.

Since fMRI preprocessing techniques are very complex image processing operations and heavily resource intensive, an initiative also formed to provide a preprocessed version of the dataset.

Since there is not a widely accepted best practice method for preprocessing fMRI images, the participating five teams have all used their preferred methods to clean the data. The participants have also provided a ROI activation timeseries calculated using 7 different brain atlases from each type of the preprocess method.

The very close ratio of affected vs control subjects (539 to 573) means the dataset is well-balanced in terms of a classification task on the diagnosis of subjects.

4.1.2 Methodology

In the paper [36] by Wang et al. researchers worked with the ABIDE dataset to create a graph convolutional network based architecture for predicting the diagnosis of a patient based on the resting state fMRI.

4.1.3 Results

4.1.4 Conclusions

4.2 Connectome-based diffusion

Deep learning methods are generally known for needing a large number of samples to produce the outstanding results they are capable of[2]. When working with naturally occurring data such as text and RGB images this is not quite as large of an issue, however medical imaging data, such MRI and CT scans, poses a problem, as it is much harder to access.

In case of fMRI scans this is partly due to the cost associated with the machines and taking scans and the cost of creating high-quality datasets, as they usually require many hours of work by experts in their respective fields. Furthermore, accessing these datasets is difficult due to privacy concerns and the sensitive nature of said data2.2.1.

Data augmentation techniques can be used to counteract these hurdles, an important one of these is data synthesis, creating entirely new samples based previously collected data. The goal of this project was to generate synthetic three dimensional fMRI brain scans. By employing diffusion generative models it is possible to produce high-fidelity images based on previous scans.

Generative models are capable of generating novel samples that they have not seen previously, based on the distribution learned from the training samples. The two types of generative models used in this model are Variational Autoencoders (VAEs) and diffusion models.

Variational Autoencoders combine deep neural networks with probabilistic modelling for image generation[19]. Instead of learning deterministic encoding, like standard autoencoders[**?**], VAEs learn a probabilistic representation, where the latent space follows a distribution which allows for controlled sampling and generation.

A VAE is made up of two parts, an encoder and a decoder. The encoder usually consists of a series of convolutional and pooling layers to downsample the data to create a bottleneck. From this latent space, the decoder works with convolutional and unpooling (upsampling) layers to try and reconstruct the input.

Once trained, the model is capable of generating new samples by sampling from the prior distribution of the latent variable $p(z) = \mathcal{N}(0, I)$ and passing in the sample at the bottleneck to the decoder part of the model.

Denoising Diffusion Probabilistic Models take inspiration from thermodynamics (particularly Langevin dynamics[37]) and instead of generating a sample in one step, they use a process of iterative noising and denoising[16]. These models have become state of the art for various image synthesis tasks, producing high-fidelity results. This does come at a cost though: these models are very computationally expensive, even with further improvements, such as DDIMs (Denoising Diffusion Implicit Models[31]).

During the forward noising process noise is sampled from a Gaussian distribution and added to the input image according to a noise scheduler. This process is repeated usually hundreds of times as the image shifts to a pure noise version (see Figure **??** for examples) as the time t runs from 0 to T.

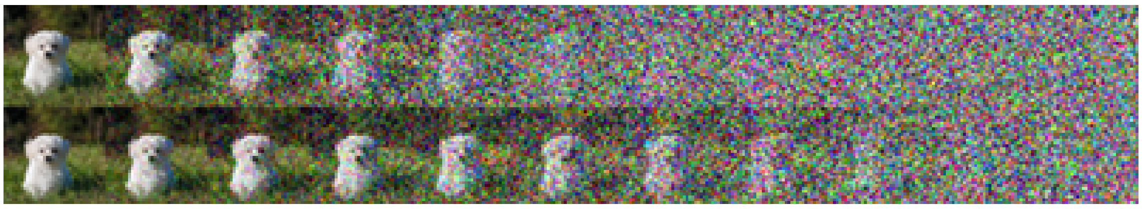


Figure 4.1: Linear (top) and cosine (bottom) noise schedulers being applied to an image of a small white dog [23]

The backwards process is the task of iteratively removing noise from the destroyed image, starting from random noise. This does not have an exact solution due to information loss, so it is approximated by a neural network. The network receives the current timestep in the noising process t and the image, aiming to calculate the noise, so it can be subtracted.

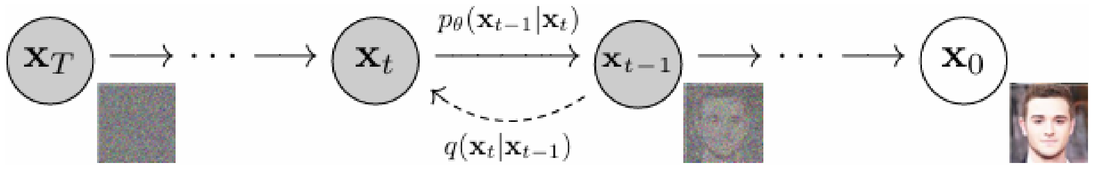

$\mathbf{x}_T$
$\mathbf{x}_t$
$p_\theta(\mathbf{x}_{t-1}|\mathbf{x}_t)$
$\mathbf{x}_{t-1}$
$q(\mathbf{x}_t|\mathbf{x}_{t-1})$
$\mathbf{x}_0$

Figure 4.2: The forward and backward process of diffusion[16]

Since generating in the original fMRI domain is incredibly resource intensive as the data is always at least 3 dimensional, only very small resolution samples could be generated within reasonable computation demands. The idea was to take the generative part of the model to a different space, by using a more compact representation of the brain scan: the average activation values of brain regions (ROIs) and the generated connectomes seemed like a good fit for this purpose.

4.2.1 Dataset

The HCP dataset (Human Connectome Project) was collected by WU-Minn HCP consortium. The goal of the human connectome project is to facilitate research into mapping out the human brain, for this reason the project collected high-quality neuroimaging data (MRI, PET and MEG scans) in various studies targeting a variety of problems: Alzheimer's disease, normal brain development or ageing. In doing so they have not only supplied high quality data for neurological researchers, but also established a protocol for such data collection[34].

One of the collected datasets is HCP Young Adult which has scans of over 1100 healthy young adults for the free use of researchers. The dataset encompasses both resting state and task-based MRIs and was collected by 10 institutions in different parts of the world.

This work focuses on the task-based portion of the dataset, where subjects were instructed to imagine moving various parts of their body while the fMRI took place, creating 5 classes in the data based on the body part - tongue, left and right hand, left and right foot respectively. The data can be downloaded from their ConnectomeDB[1] in an already image-wise preprocessed state as well.

Creating an easy to use dataset, where the end user wouldn't need to have all the necessary medical knowledge and computation power to do the initial, required preprocessing steps was also a focus of the project. For this purpose common automated preprocessing pipelines were created to accomplish low-level tasks, such as surface generation, artefact removal and alignment to standard space[13].

4.2.2 Methodology

The ROI+connectome representation first has to be extracted from each sample of the dataset and then they are used as the input for the diffusion model. Since the connectome at this point is not an image but rather a sort of brain graph, a decision has been made to create a model different from most DDPMs, by creating a specialized graph U-net, that can serve as the neural network inside the diffusion model that predicts the noise. To do this, I adapted the original graph U-net[12] to work in this setting, by adding time and class conditioning and edge prediction techniques.

The resulting generated connectomes and ROI values are then used as conditioning in the bottleneck of a VAE, which has been trained to generate MRI scans with the original images of the same dataset, using the original ROI and connectome values as conditioning during training.

[1]`https://db.humanconnectome.org/app/template/Login.vm`

Here, the goal was to make sure the VAE learns how to generate scans that correspond to the information fed at the bottleneck; this way it is possible to generate ROI and connectome samples, which can be turned into high-fidelity MRI samples with significantly less resource usage than if we were generating the MRIs in the first place.

4.2.2.1 Data Preprocessing

The architecture requires three separate types of data: the fMRI scan itself and the ROI values and connectomes corresponding to the scan. Of these the HCP dataset only contains the preprocessed fMRI scan, the other two needed to be computed in a preprocessing pipeline3.2.2. The ROI values were extracted with the help of an atlas, which describes the regions of the brain corresponding to the ROI values. Since the focus of the project was on sensory motor tasks, therefore we filtered for brain regions activated during these tasks (in practice this means 21 ROIs for each side of the brain, so 42 altogether).

With the filtered atlas, a mask can be constructed, to block out unwanted parts of the brain during the extraction. Understanding the format and meaning of the data used in the preprocessing (e.g.: What ROIs and connectomes are and how to calculate them) is difficult as it requires knowledge in the medical domain.

The correct preprocessing was possible thanks to communication from doctors, the documentation of the nilearn package and high-quality HCP dataset. Due to the preprocessing of the dataset, it works best with the Glasser360 brain atlas[32], which was constructed using HCP data and identifies 180+180 (in the left and right hemisphere) regions of interest according to changes in cortical architecture, function, connectivity, and/or topography.

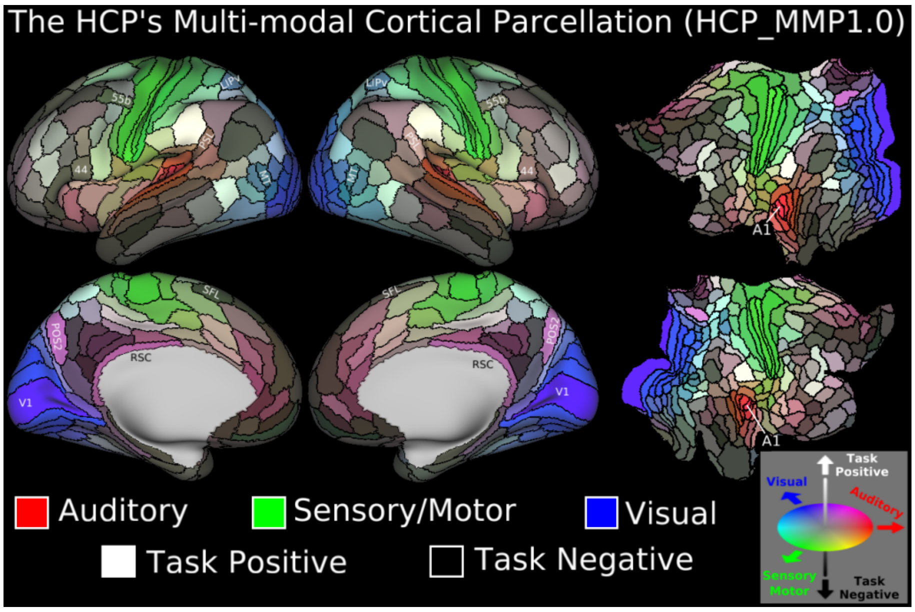


Figure 4.3: The Glasser360 brain atlas created based on the HCP healthy adults dataset[32]

After the extraction is complete we get a timeseries of ROI values for each patient. This means a tensor of size $284 \times 16 \times 42$, (time $\times$ repeatedMeasurements $\times$ ROIs). To validate the correctness of the preprocessing a Support Vector Machine was used to classify the ROIs based on the performed task. For this the ROI values need to be standardized per subject, and separated by task. For this a Python script from previous works was used that generates a csv file with the times slices and the task performed at that time interval.

This file is essential in separating the relevant parts of the scans from the noise where no task is performed. The scans contain measurements for 5 tasks performed twice, so in to 10 for each patient, with 16 repeated measurements take by the MRI machine (As a further preprocessing step, before the data was used in models the mean value of these 16 steps was taken). This means 124 of the 284 long scan is noise.

Once the preprocessing was correct the SVM achieved 90-95% accuracy on the classification task based on performed tasks. For calculating the connectomes from the ROI values the nilearn package's ConnectivityMeasure class was used.

The subject-wise calculated ROIs were separated by tasks and connectomes were calculated for each task separately. Since measurements across different tasks are differ significantly, calculating the connectomes with the tasks mixed would not yield meaningful results.

4.2.2.2 Proposed Architecture

The model consists of a diffusion block and an autoencoder. The diffusion block is trained on the ROI values and connectomes to also generate ROI values and connectomes, which are used to condition the autoencoder at the bottleneck.

A big advantage of this model is that the two parts of the model can be trained separately. The autoencoder is trained to be able to reproduce scans with the corresponding connectome. At sampling time the two models are used in conjunction: the diffusion model generates a connectome and only the decoder part of the autoencoder is used to generate novel samples.

For the DDPM and DDIM diffusion blocks the implementation in the denoising-diffusion-pytorch package was used to enable quick experimentation. The package provides an easy interface to train diffusion models with custom networks, though it required some unused parameters to be added to model, that were unnecessary in this use-case.

Another advantage of this package is that it enables easy DDIM sampling. This could be used with 200 steps, instead of the 1000 used for training the DDPM. This package sadly lacks the capability for training a network with class conditioning, so for experiments involving class conditioning it was extended manually.

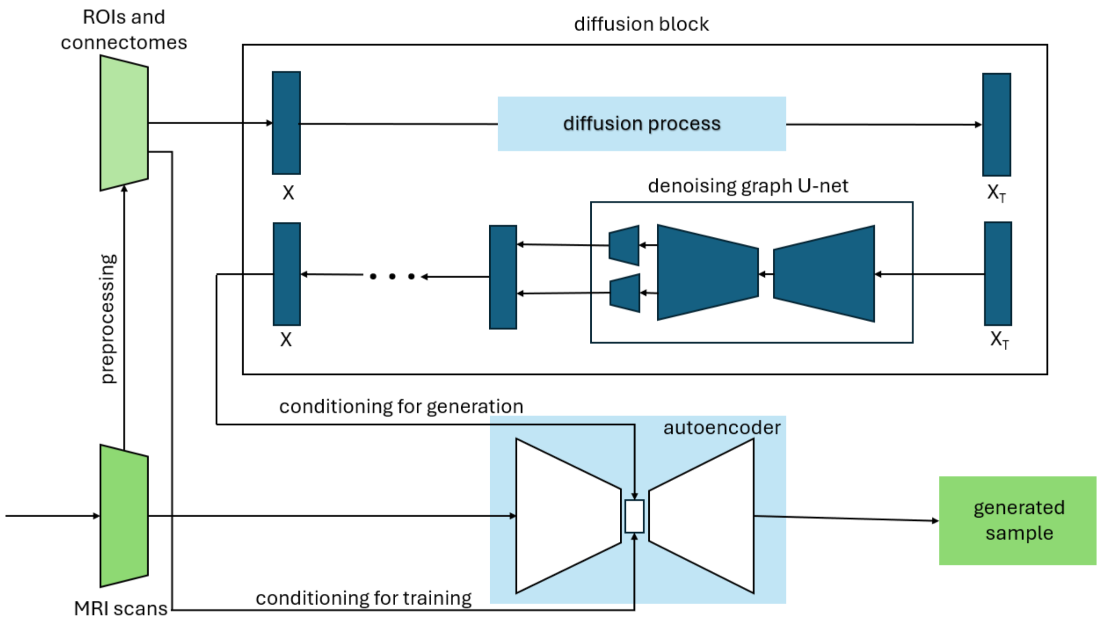


Figure 4.4: Visualization of the model architecture, focusing on the flow of data

For a VAE implementation, since it is not a focus of this work, a choice was made to use an implementation provided by the latent-diffusion Github repository[2]. The model this library provides has the option to train a VAE that learns representations on 3D data.

The class was modified in minor ways to fit in with the rest of the codebase and two major changes had to be made to it. Firstly, the loss function was recalibrated to enable the model to train on the given dataset. Secondly, it was extended with the capability to condition the bottleneck with the ROI values and connectomes.

For the conditioning a small CNN was implemented with a fully connected layer at the end, that takes the $2 \times 42 \times 42$ size ROI and connectome tensor and creates an embedding with the same size as the bottleneck. The conditioning is then implemented by performing Adaptive Instance Normalization[17] on the embedding and the values in the bottleneck.

4.2.2.3 Graph U-net

U-nets are a very successful subtype of CNN architectures, originally invented with the purpose of biomedical image segmentation[27]. It is widely used in segmentation tasks and other cases where the output needs to have the same dimensions as the input, due to its symmetrical nature. This is the same reason why they're commonly used in diffusion models as the noise prediction neural network[16].

[2]`https://github.com/CompVis/latent-diffusion`

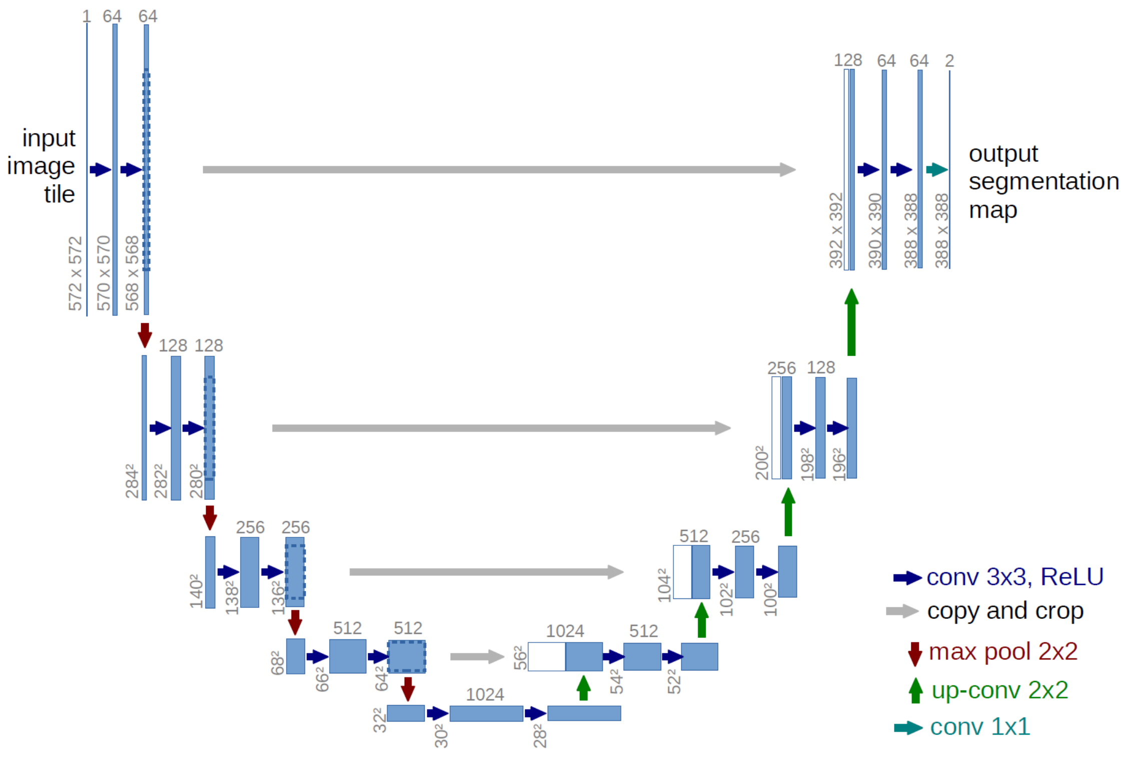


Figure 4.5: Visualisation of the U-net architecture[27]

U-nets are named due to their distinctive architecture: the network consists of a contracting and expansive path, which together form a u-shape (Figure **??**). The contracting path is a typical CNN with convolutions and max pooling. Conversely the expansive path has upsampling and convolutions, while skip connections are also utilized where dimensions match.

Since U-nets have been such a successful model type in convolutional networks, there has been an effort to find an analogue in graph convolutional networks as well. Graph U-nets[12] follow the original principles of the U-net architecture: the encoder-decoder structure, the skip connections and the general architecture, as seen on Figure **??**.

The convolutional layers are replaced by GCN layers and for standard pooling and unpooling layers which do not have an obvious graph equivalent, since they rely on locality in images that is not present in general graphs, novel layers have been developed for graph pooling and unpooling.

Graph pooling layers enable downsampling on graphs by adaptively selecting nodes to build a smaller graph from existing nodes, while preserving edges. The selection method is similar to a traditional k-max pooling: the highest k values are preserved after the operation. The top k nodes are selected based on a learned 1D projection, to a vector $\mathbf{p}$:

$$y_i = x_i\mathbf{p}/||\mathbf{p}||,$$

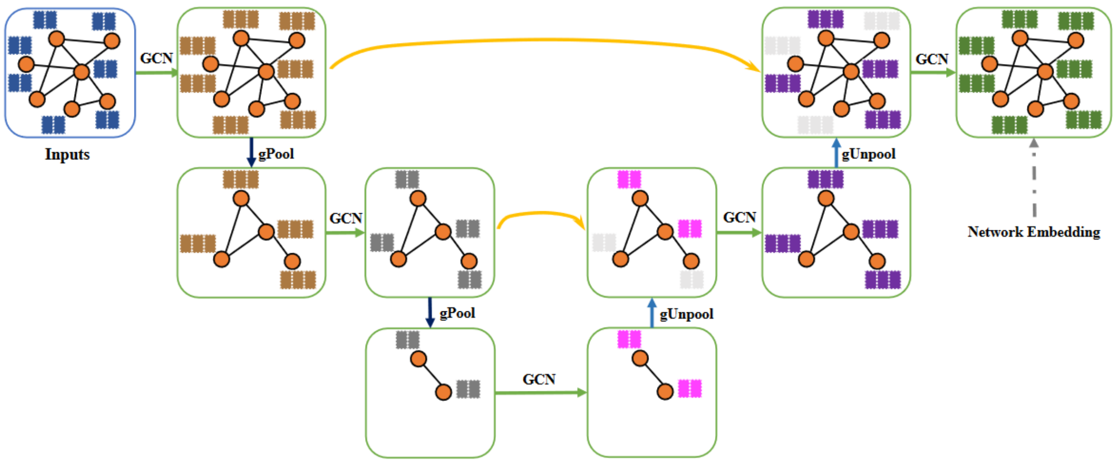


Figure 4.6: Visualization of the graph U-nets architecture[12]

the nodes are kept which retain most of their information when projected to $\mathbf{p}$. After the selection has been made, the adjacency matrix is updated to only include these nodes and the corresponding rows are extracted from the feature matrix via a gate operation, using matrix multiplication and a sigmoid operation, which makes p trainable by backpropagation.

The unpooling layer, unlike image upsampling layers, can only work in connection with a graph pooling layer for which it performs an opposite action: the adjacency matrix is restored to its state prior to the the pooling layer and the node feature matrix is distributed into a shape corresponding to this larger number of nodes, having zero values for newly reappearing nodes.

The graph U-net used in our model is a modified version of the architecture proposed in [12]. The connectome is used as the adjacency matrix in the U-net, the connectivity values between ROIs are the weights of edges in the graph.

Since we have singular ROI values available for each of the nodes these were distributed along an identity matrix to create a better node feature input for the network, and since the U-net requires symmetry and we will be looking to generate both connectomes and ROI values, the full input is the concatenation of the ROI filled identity matrix and the connectome.

In this case, since the connectomes are 42×42, the input of this part of the network is batch size $\times 42 \times 84$.

The model has the same number of blocks in both directions, comprised of one GCN layer and one graph pooling layer in the down direction and one graph unpooling layer and one GCN layer in the up direction, with a connecting bottom GCN layer.

Corresponding blocks of the same size are connected via skip connections. Conditioning (with the main concern being time conditioning) is applied at every block by calculating scale and shift vectors via a small MLP.

The output of the last block is thus the same size as the input of the first one and it gets separated into node features that are used for ROI values calculation and node features that are used for connectome calculation. The ones used for ROIs are sent through a final GCN layer where they are reduced to single value per node and then distributed along an identity matrix to preserve the symmetry of the data.

For the connectome however, as each value represents an edge between two regions, the other half of the output gets pairwise concatenated and passed through an MLP to determine one value per edge for the connectome.

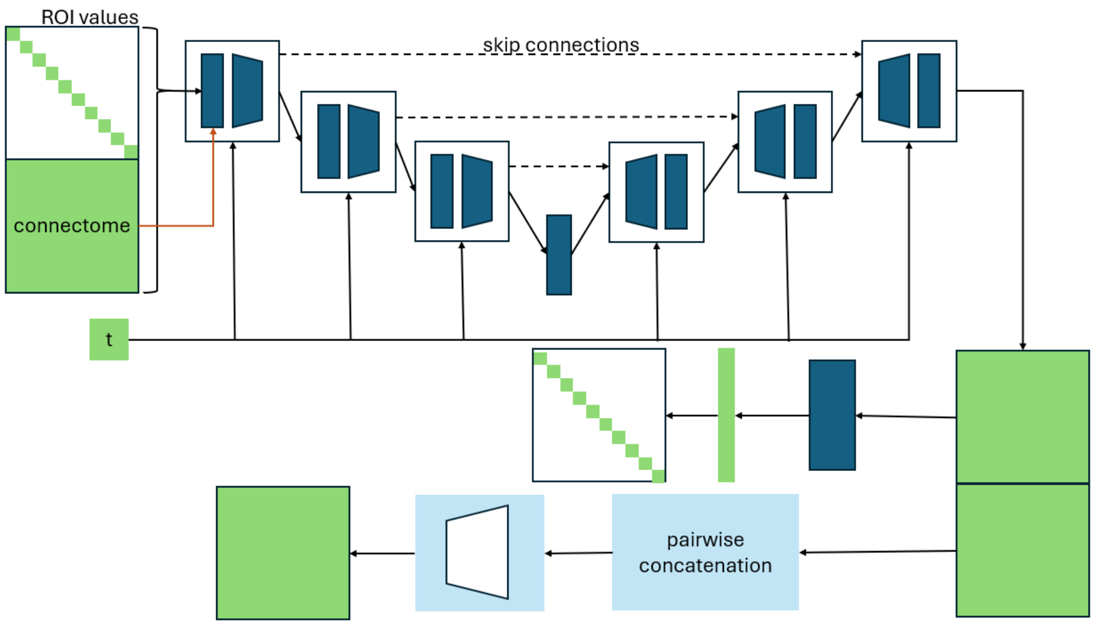


Figure 4.7: Visualization of the graph U-net architecture, focusing on the flow of data

4.2.2.4 Experiments

To find out which choices work best for the model, there were multiple experiments with various architecture and parameter ideas. This was mainly focused on the graph U-net, as it is the central part of the architecture. The VAE remained largely the same, with only minor changes to accommodate the conditioned training and sampling.

Since this kind of graph U-net and diffusion architecture does not show up in literature often, experiments were done to try and figure out which setups would lead to better generation results in the end. There were wide variety of options to try and the lack of guidance on which ones could be successful in which combination meant trying out new ideas in a fail fast method would be best: the results of the model were inspected after a set number of epochs and if the connectome visually worsened that method could be avoided in the future.

Since the base output of the model is just node features and there are various methods to convert them into edge predictions[20] we had to try a few to see what would work well with the model. In the implementation of the graph U-net we included a switch that can take None, 'scalar' and 'mlp' values and makes experimenting with this easier. The None value keeps the original vector of the output, without any transformations applied. In this setting the network tries to learn the noise/denoised values as if they were features of the nodes. This is the easiest method to implement but this way it is not taken into consideration how each edge represents the connection between two edges. To better accomplish this, the 'scalar' method takes each resulting node embedding and computes the scalar product of each node pair to build the new matrix. In case of the 'mlp' option the node embedding get pairwise concatenated to represent their respective edges, and then are passed through a small MLP to calculate an edge strength for the pair of nodes. Here the concatenation makes sure that both nodes are considered in the calculation. In the final version we ended up using the MLP to calculate the connectome as it yielded better results than the base method and with the scalar product approach we discovered that it led to generated samples looking almost identical. 4.4.1.2 Class Conditioning As mentioned before in Section 2.5.2, introducing a class conditioning in a DDPMs can give a significant boost to their performance. Since our dataset had labels in the form of the body part corresponding to each sample that the subject was instructed to imagine raising, we could try this method in our model as well. We implemented a very simple version of this by extending the diffusion package we used with the class conditioning being passed to the U-net and in the GNN concatenated the one-hot vector representing the class to the time conditioning to execute both kinds at once. Sadly this implementation did not result in improved performance, the resulting generated samples tended to have prominent bands of a single value instead of resembling the input connectomes. We have not yet found the reason for this development and we believe it still might be a good idea to further experiment with conditioning, perhaps trying to create a more refined implementation. 4.4.1.3 Loss Calculation Since the connectome is supposed to be a symmetric matrix, we tried changing the graph Unet's loss function to reflect this in some form. We implemented a version of loss calculation where only the lower triangle matrix of both the result and target were considered during calculation: keeping the generation of the upper half as a sanity check, but concentrating on the lower half. This did not noticeably impact the resulting generated samples, so we opted for removing it, choosing to keep the model simpler if possible. 20

4.2.3 Results

Evaluating image synthesis tasks requires multiple different approaches. For one, quantitative metrics like Fréchet Inception Distance[19] (FID), Peak Signal to Noise Ratio[21] (PSNR), and Structural Similarity Index Measure[21] (SSIM) are key to obtaining an objective view of the quality of the generated samples. On the other hand, especially in image synthesis tasks, qualitative evaluation is important to assess the visual quality of the samples, as numerical measurements don't always align with visual representations. FID uses an InceptionV3 network to calculate the distance between the extracted features of real and generated samples, PSNR measures the ratio of the signal (i.e., the important parts of the image) and noise in the reconstructed image, and SSIM, unlike PSNR, measures not the pixel values, but structural aspects of the samples.

As the numerical metrics show, our model fails to produce any meaningful results with the VAE conditioned by the generated ROIs and connectomes. However, these metrics are not an accurate representation of our results. For that we need to look at some generated samples.

4.2.4 Conclusions

4.3 Minimising device-to-device differences

4.3.1 Dataset

4.3.2 Methodology

4.3.3 Results

4.3.4 Conclusions

Chapter 5

Future work

5.1 Future work

5.2 Conclusions

Acknowledgements

Ez nem kötelező, akár törölhető is. Ha a szerző szükségét érzi, itt lehet köszönetet nyilvánítani azoknak, akik hozzájárultak munkájukkal ahhoz, hogy a hallgató a szakdolgozatban vagy diplomamunkában leírt feladatokat sikeresen elvégezze. A konzulensnek való köszönetnyilvánítás sem kötelező, a konzulensnek hivatalosan is dolga, hogy a hallgatót konzultálja.

Bibliography

[1] Types of healthcare data: A comprehensive overview. URL `https://kms-healthcare.com/blog/types-of-healthcare-data/`.

[2] Laith Alzubaidi, Jinshuai Bai, Aiman Al-Sabaawi, Jose Santamaría, Ahmed Shihab Albahri, Bashar Sami Nayyef Al-Dabbagh, Mohammed A Fadhel, Mohamed Manoufali, Jinglan Zhang, Ali H Al-Timemy, et al. A survey on deep learning tools dealing with data scarcity: definitions, challenges, solutions, tips, and applications. *Journal of Big Data*, 10(1):46, 2023.

[3] RJ Anderson. The collection, linking and use of data in biomedical research and health care: ethical issues. 2015.

[4] M Stella Atkins and Blair T Mackiewich. Fully automatic segmentation of the brain in mri. *IEEE transactions on medical imaging*, 17(1):98–107, 2002.

[5] Danielle S Bassett and Olaf Sporns. Network neuroscience. *Nature neuroscience*, 20 (3):353–364, 2017.

[6] European Commission. Opinion 4/2007 on the concept of personal data., 2007. URL `https://ec.europa.eu/justice/article-29/documentation/opinion-recommendation/files/2007/wp136_en.pdf`.

[7] Kristen Coyne. Mri: A guided tour. URL `https://nationalmaglab.org/magnet-academy/read-science-stories/science-simplified/mri-a-guided-tour/`.

[8] Ameya Daigavane, Balaraman Ravindran, and Gaurav Aggarwal. Understanding convolutions on graphs. *Distill*, 2021. DOI: `10.23915/distill.00032`. https://distill.pub/2021/understanding-gnns.

[9] Edgar A DeYoe, Peter Bandettini, Jay Neitz, David Miller, and Paula Winans. Functional magnetic resonance imaging (fmri) of the human brain. *Journal of neuroscience methods*, 54(2):171–187, 1994.

[10] European Parliament and Council of the European Union. Regulation (EU) 2016/679 of the European Parliament and of the Council. URL `https://data.europa.eu/eli/reg/2016/679/oj`.

[11] European Parliament and Council of the European Union. Regulation (eu) 2025/327 of the european parliament and of the council of 11 february 2025 on the european health data space and amending directive 2011/24/eu and regulation (eu) 2024/2847 (text with eea relevance), 2025. URL `https://eur-lex.europa.eu/legal-content/EN/TXT/?uri=OJ:L_202500327`.

[12] Hongyang Gao and Shuiwang Ji. Graph u-nets. In *international conference on machine learning*, pages 2083–2092. PMLR, 2019.

[13] Matthew F Glasser, Stamatios N Sotiropoulos, J Anthony Wilson, Timothy S Coalson, Bruce Fischl, Jesper L Andersson, Junqian Xu, Saad Jbabdi, Matthew Webster, Jonathan R Polimeni, et al. The minimal preprocessing pipelines for the human connectome project. *Neuroimage*, 80:105–124, 2013.

[14] Gary H Glover. Overview of functional magnetic resonance imaging. *Neurosurgery Clinics of North America*, 22(2):133, 2011.

[15] William L. Hamilton, Rex Ying, and Jure Leskovec. Inductive representation learning on large graphs, 2018. URL `https://arxiv.org/abs/1706.02216`.

[16] Jonathan Ho, Ajay Jain, and Pieter Abbeel. Denoising diffusion probabilistic models. *Advances in neural information processing systems*, 33:6840–6851, 2020.

[17] Xun Huang and Serge Belongie. Arbitrary style transfer in real-time with adaptive instance normalization. In *Proceedings of the IEEE international conference on computer vision*, pages 1501–1510, 2017.

[18] Hany Kasban, MAM El-Bendary, DH Salama, et al. A comparative study of medical imaging techniques. *International Journal of Information Science and Intelligent System*, 4(2):37–58, 2015.

[19] Diederik P Kingma, Max Welling, et al. Auto-encoding variational bayes, 2013.

[20] Juanhui Li, Harry Shomer, Haitao Mao, Shenglai Zeng, Yao Ma, Neil Shah, Jiliang Tang, and Dawei Yin. Evaluating graph neural networks for link prediction: Current pitfalls and new benchmarking. *Advances in Neural Information Processing Systems*, 36:3853–3866, 2023.

[21] Brent Daniel Mittelstadt and Luciano Floridi. The ethics of big data: current and foreseeable issues in biomedical contexts. *The ethics of biomedical big data*, pages 445–480, 2016.

[22] Menno Mostert, Annelien L Bredenoord, Monique CIH Biesaart, and Johannes JM Van Delden. Big data in medical research and eu data protection law: challenges to the consent or anonymise approach. *European Journal of Human Genetics*, 24(7): 956–960, 2016.

[23] Alexander Quinn Nichol and Prafulla Dhariwal. Improved denoising diffusion probabilistic models. In *International conference on machine learning*, pages 8162–8171. PMLR, 2021.

[24] Shima Ovaysikia, Khalid A Tahir, Jason L Chan, and Joseph FX DeSouza. Word wins over face: emotional stroop effect activates the frontal cortical network. *Frontiers in human neuroscience*, 4:234, 2011.

[25] Donald B Plewes and Walter Kucharczyk. Physics of mri: a primer. *Journal of magnetic resonance imaging*, 35(5):1038–1054, 2012.

[26] Baxter P Rogers, Victoria L Morgan, Allen T Newton, and John C Gore. Assessing functional connectivity in the human brain by fmri. *Magnetic resonance imaging*, 25 (10):1347–1357, 2007.

[27] Olaf Ronneberger, Philipp Fischer, and Thomas Brox. U-net: Convolutional networks for biomedical image segmentation. In *Medical image computing and computer-assisted intervention–MICCAI 2015: 18th international conference, Munich, Germany, October 5-9, 2015, proceedings, part III 18*, pages 234–241. Springer, 2015.

[28] John Mark Michael Rumbold and Barbara Pierscionek. The effect of the general data protection regulation on medical research. *J Med Internet Res*, 19(2):e47, Feb 2017. ISSN 1438-8871. DOI: `10.2196/jmir.7108`. URL `http://www.jmir.org/2017/2/e47/`.

[29] Benjamin Sanchez-Lengeling, Emily Reif, Adam Pearce, and Alexander B. Wiltschko. A gentle introduction to graph neural networks. *Distill*, 2021. DOI: `10.23915/distill.00033`. https://distill.pub/2021/gnn-intro.

[30] Stephen M Smith. Overview of fmri analysis. *The British Journal of Radiology*, 77 (suppl_2):S167–S175, 2004.

[31] Jiaming Song, Chenlin Meng, and Stefano Ermon. Denoising diffusion implicit models. *arXiv preprint arXiv:2010.02502*, 2020.

[32] Olaf Sporns, Giulio Tononi, and Rolf Kötter. The human connectome: a structural description of the human brain. *PLoS computational biology*, 1(4):e42, 2005.

[33] Kenji Suzuki. Overview of deep learning in medical imaging. *Radiological physics and technology*, 10(3):257–273, 2017.

[34] David C Van Essen, Kamil Ugurbil, Edward Auerbach, Deanna Barch, Timothy EJ Behrens, Richard Bucholz, Acer Chang, Liyong Chen, Maurizio Corbetta, Sandra W Curtiss, et al. The human connectome project: a data acquisition perspective. *Neuroimage*, 62(4):2222–2231, 2012.

[35] Petar Veličković, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Lio, and Yoshua Bengio. Graph attention networks. *arXiv preprint arXiv:1710.10903*, 2017.

[36] Lebo Wang, Kaiming Li, and Xiaoping P Hu. Graph convolutional network for fmri analysis based on connectivity neighborhood. *Network Neuroscience*, 5(1):83–95, 2021.

[37] Max Welling and Yee W Teh. Bayesian learning via stochastic gradient langevin dynamics. In *Proceedings of the 28th international conference on machine learning (ICML-11)*, pages 681–688. Citeseer, 2011.

[38] Anderson Winkler. The nifti file format, 2012. URL `https://brainder.org/2012/09/23/the-nifti-file-format/`.

[39] Jie Zhou, Ganqu Cui, Shengding Hu, Zhengyan Zhang, Cheng Yang, Zhiyuan Liu, Lifeng Wang, Changcheng Li, and Maosong Sun. Graph neural networks: A review of methods and applications. *AI Open*, 1:57–81, 2020. ISSN 2666-6510. DOI: `https://doi.org/10.1016/j.aiopen.2021.01.001`. URL `https://www.sciencedirect.com/science/article/pii/S2666651021000012`.