Kamilė Stankevičiūtė

Graph neural networks for age prediction from neuroimaging data

Computer Science Tripos – Part II

Gonville & Caius College

2020

Declaration of originality

I, Kamilė Stankevičiūtė of Gonville & Caius College, being a candidate for Part II of the Computer Science Tripos, hereby declare that this dissertation and the work described in it are my own work, unaided except as may be specified below, and that the dissertation does not contain material that has already been used to any substantial extent for a comparable purpose.

I, Kamilė Stankevičiūtė of Gonville & Caius College, am content for my dissertation to be made available to the students and staff of the University.

Signed *Kamilė Stankevičiūtė*

Date 23rd June 2020

Acknowledgements

I would like to express my deepest gratitude to my supervisors, Mr Tiago Azevedo and Mr Alexander Campbell, for their patient guidance and their endless support and encouragement. Without them this project would not have been what it is today.

I am also grateful to Prof Pietro Liò for his extreme faith in me as well as his experienced advice and insight.

I would like to extend my thanks to Dr Richard Bethlehem for his kind assistance with collecting and preprocessing the dataset, and his invaluable expertise in the fascinating field of neuroimaging.

I am very thankful to Dr Timothy Jones and Dr Graham Titmus for taking their time and effort to read through my dissertation and give constructive feedback.

Finally, I would like to thank my family, Prof Jelena Stankevičienė, Dr Andžej Stankevič, and Mr Martynas Stankevičius, for their academic advice, unconditional support and understanding.

Proforma

Candidate number:	2419E
Project title:	Graph neural networks for age prediction from neuroimaging data
Examination:	Computer Science Tripos – Part II 2020
Word count:	11438[1]
Lines of code:	1208
Project originator:	Mr Tiago Azevedo
Supervisors:	Mr Tiago Azevedo, Mr Alexander Campbell

Original aims of the project

To approach the brain age prediction problem using population graph data structure, combining the richness of multi-modal structural and functional neuroimaging as well as non-imaging data in a single flexible pipeline. To implement and apply different graph neural network architectures – Graph Convolutional Network (GCN) and Graph Attention Network (GAT) – which would use the population graphs for the brain age prediction task. Evaluate the two models using standard metrics for regression problems.

Work completed

A flexible pipeline for construction of custom population graphs from minimally pre-processed neuroimaging data. Implementation of the two graph neural network architectures, trained on population graphs to predict subject brain age. Evaluation of the models against standard metrics and an additional robustness measurement framework, measuring the tolerance of the models to population graph node and edge noise.

Special difficulties

None.

[1] Counted using the LaTeX Utilities extension in Visual Studio Code.

Contents

1 Introduction

Many common neurological and neurodegenerative disorders, such as Alzheimer's disease, schizophrenia and multiple sclerosis, have been associated with abnormal patterns of apparent ageing of the brain [1]. This link has inspired numerous studies in *brain age estimation* using neuroimaging data [2]. The *brain age gap*, defined as the discrepancy between the estimated brain age and the true chronological age, is a powerful biomarker not only for understanding the biological pathways behind the ageing process, but also for assessing an individual's risk to various brain disorders and identifying new personalised treatment strategies [3].

Numerous studies exist applying machine learning algorithms to the problem of brain age estimation, typically using structural magnetic resonance imaging (MRI) and genetic data [2]. They tend to model healthy controls separately from individuals with brain disorders, and often develop separate models for each sex [1, 4] without explicitly considering potential variation in ageing patterns across different subgroups of subjects. Moreover, these studies rarely include other important brain imaging modalities such as functional MRI (fMRI) time-series data, or clinical expertise of neurologists and psychiatrists, even though a combination of different modalities has been shown to improve the results [4].

This dissertation proposes a pipeline that can flexibly combine the richness of minimally preprocessed neuroimaging and non-imaging modalities to predict brain age in a clinically relevant fashion. The modalities are combined using the *population graph* representation of patients. In the population graph, the nodes contain subject-specific neuroimaging data, and edges capture pairwise subject similarities of non-imaging data. In addition to controlling for confounding effects, these similarities help to exploit neighbourhood information when predicting node labels – an approach that has successfully been applied to a variety of problems in both medical and non-medical domains [5, 6, 7].

The population graph is used as input to two types of neural networks that can operate on graph-structured data – the *graph convolutional* [8] and the *graph attention* [9] networks – to predict the brain age. This is similar to the approach of Parisot et al. [10, 5], where population graphs are used to classify patients as healthy or suffering from a brain disorder (i.e. autism spectrum disorder and Alzheimer's disease).

Unlike in other state-of-the-art models, graph neural networks trained on population graphs have the potential to learn from the entire cohort of healthy and affected subjects of both sexes at once, capturing a wide range of confounding effects [11, 12] and detecting the variation in brain age trends of different sub-populations of subjects.

2 Preparation

This chapter presents the brain age estimation problem (Section 2.1) in more depth and the dataset used for the brain age estimation task (Section 2.2). It also gives background on population graphs (Section 2.3) as well as the neural network architectures for implementing them (Sections 2.4 and 2.5).

2.1 Brain age estimation

The *brain age* is defined as the *apparent* age of the brain, compared to the person's true (or *chronological*) age. For example, a highly atrophied brain of a young person may appear to have a higher brain age than a healthy brain of an older person, because gradual atrophy is related to the normal brain ageing process.

Formally, the brain age y_b can be expressed as the sum of the known chronological age y_c and the unknown *brain age gap* ε_g that is defined as the discrepancy between the chronological and the brain age [4]:

$$y_b = y_c + \varepsilon_g. \tag{2.1}$$

It is generally assumed [2] that a typical healthy person has a normally ageing brain, so the brain age corresponds to chronological age:

$$y_b \approx y_c. \tag{2.2}$$

Our goal is to estimate brain age y_b as a function $f(\cdot)$ of brain imaging feature vector $\mathbf{x}$:

$$y_b = f(\mathbf{x}) + \varepsilon_e, \tag{2.3}$$

where ε_e is the prediction error, while the estimate of *chronological* age is instead (from Equations 2.1 and 2.3)

$$y_c = f(\mathbf{x}) + \varepsilon, \tag{2.4}$$

where $\varepsilon \coloneqq \varepsilon_e - \varepsilon_g$ is the error term consisting of both the brain age gap and the model prediction error.

Table 2.1: Summary of the features derived from structural MRI, reported separately for every brain region.

Structural feature	**Description**
Cortical thickness	Average thickness of the cerebral cortex in a brain region.
Surface area	Surface area of the cerebral cortex in a brain region.
Grey matter volume	Total volume of the grey matter component of a brain region.

Since the brain age y_b is unknown, any (semi-)supervised machine learning model can only use chronological age as a predicted variable, following Equation 2.4. However, if the model is trained on healthy subjects only, $f(\cdot)$ can explain both the apparent brain age *and* the chronological age with $\mathbf{x}$, since for healthy subjects $y_b \approx y_c$ (Equation 2.2) and any variance in ε is assumed to contain just the prediction error ε_e. When the same model is applied to non-healthy subjects, $f(\cdot)$ explains the chronological age assuming the brain is healthy, and any *additional* unexplained variance in ε is assumed to be the brain age gap. On the other hand, if the model is trained on both healthy and non-healthy subjects at the same time, it might learn the combined confounding effects of both normal (chronological) and disease-related (brain) ageing, thus hiding the brain age gaps [13].

An alternative method that does not restrict training data only to healthy subjects is proposed in Niu et al. [4]. However, it requires experimentally verifying (e.g. through subjects' performance in cognitive behaviour tests) that ε depends primarily on the brain age gap ε_g and not the brain age prediction error ε_e, which is out of the scope of this dissertation.

2.2 Dataset

For this project I will use the The United Kingdom Biobank (UKB) [14], which is a continuous population-wide study of over 500,000 participants containing a wide range of measurements. Of particular relevance to this dissertation are the UKB participants with structural and functional magnetic resonance imaging (MRI) data, a total of (at the time of writing) 17,550 subjects. The features that have been extracted from these subjects are explained in more detail in this section.

2.2.1 Structural features

Structural MRI is used to analyse the anatomy of the brain. The two main MRI modalities included in UKB are T1-weighted and T2-weighted FLAIR scans, each capturing a different type of tissue contrast (see Figure 2.1 for T1 and T2-weighted MRI images). The structural features used in this project – *cortical thickness*, *surface area* and *grey matter volume* (see Table 2.1) – have been extracted from structural MRI images using the Human Connectome Project (HCP) Freesurfer pipeline (see Glasser et al. [15] for further discussion of features and preprocessing steps).

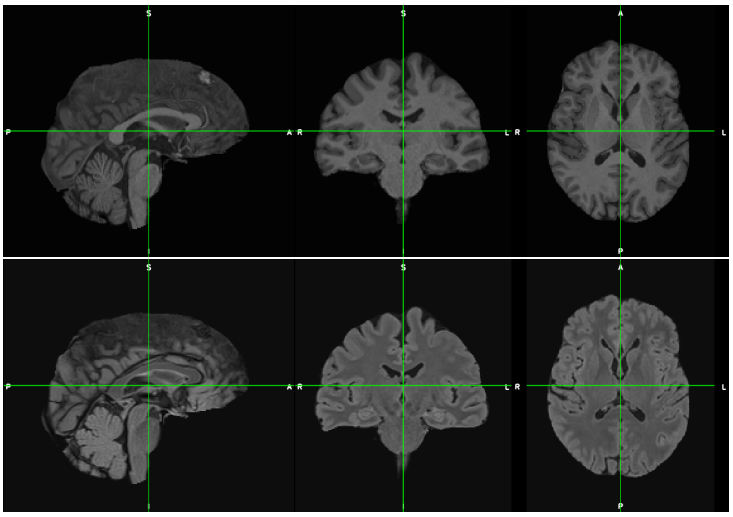


Figure 2.1: Axial slices of the T1 (top) and T2-weighted FLAIR MRI images of a brain. Left to right: the view from the side, front, and top of the brain.

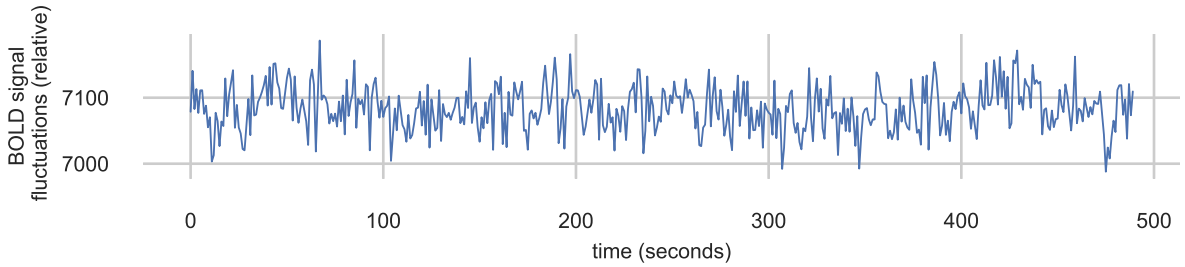


Figure 2.2: An example BOLD time-series for a single brain region, showing fluctuations in relative units.

2.2.2 Functional features

Functional MRI (fMRI) data indirectly measures brain activity over time. It is represented by *blood oxygenation level dependent* (BOLD) time-series (Figure 2.2) that show changes in oxygenation of the blood vessels. Sudden increases in oxygen demand are related to higher brain activity in the corresponding brain region.

The two types of an fMRI image are *task* fMRI, when the subject is asked to perform a specific cognitive task, and *resting state* fMRI (rs-fMRI), when the subject is not performing any particular task. This project will use rs-fMRI data.

For the purposes of machine learning analysis, *functional connectivity matrices* will be derived from rs-fMRI and used as input features to the graph neural network models. Functional connectivity is conceptualised as the correlation between the time-series of every pair of brain regions together under the assumption that parts of the brain that have related functions would also have similar activity patterns (highly correlated time-series). This is a highly researched common practice for representing brain connectivity [16, 17].

2.2.3 Euler indices

The Euler index [18] is a quality control metric which represents the number of times the HCP Freesurfer brain reconstruction software failed to seamlessly connect two 2D slices of an MRI image into the 3D representation of the brain. The higher the Euler index, the worse is the quality of the scan. Euler indices might be used to remove the subjects with low-quality scans to avoid them affecting the analysis [1] (in this project this was not needed because the indices were generally low). Alternatively, Euler indices can be used as a covariate in a machine learning model (as a brain similarity metric or a node feature) to correct for any scan quality-related bias in prediction. This project will use Euler indices as node features for consistency in keeping neuroimaging features in nodes and non-imaging features in edges.

2.2.4 Neuroimaging data preprocessing

All MRI and fMRI data in UKB was preprocessed (denoised and motion-corrected) with standard UK Biobank pipelines[1]. For the purpose of this study, the data was further *parcellated* at the Department of Psychiatry by Dr Richard Bethlehem, Dr Rafael Romero-Garcia and Dr Lisa Ronan[2].

A *parcellation* splits an image of a brain into biologically meaningful regions for downstream analysis, compressing per-voxel[3] measurements into per-parcel summaries (see Figure 2.3 for an example). In this dissertation, the neuroimaging data will be parcellated with one of the most common parcellations developed by Glasser et al. [19], which divides the brain into 360 cortical regions and 16 subcortical regions.

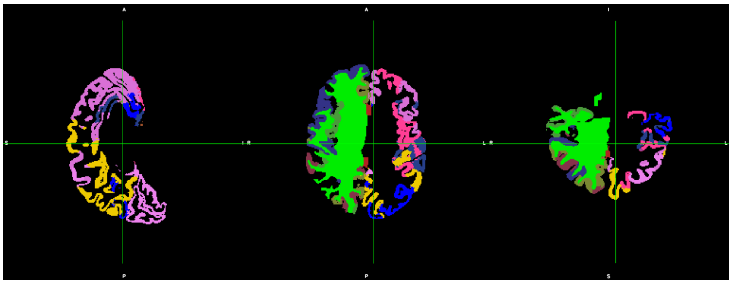


Figure 2.3: Axial slices of an MRI image of a brain (left to right: from the side, from the top, from the back), with parcels highlighted in different colours.

2.2.5 Non-imaging features

In this dissertation, non-imaging data refers to all subject data that does not come from structural MRI and fMRI scans. The features useful to this project include the subject's binary (biological) sex, the psychiatric disorder diagnoses, mental health status, education, and other variables that might have a confounding effect on structure and functional

[1] https://biobank.ctsu.ox.ac.uk/crystal/crystal/docs/brain_mri.pdf
[2] https://github.com/ucam-department-of-psychiatry/UKB
[3] A *voxel* is a discrete volumetric element.

connectivity of the brain [11] and consequently affect the brain age. Table 2.2 summarises the non-imaging features used in this project. The features have been selected as very likely to affect brain ageing and/or neuroimaging scans, and cited as common confounders in literature (see Table 2.2 for explanations).

Table 2.2: Summary of the non-imaging features used in this project.

Code	Non-imaging feature	Explanation
`AGE`	Chronological age	Used as the training label.
`FI`	Fluid intelligence score	Measures cognitive performance. Related to increased brain activity [20].
`FTE`	Years of full-time education	Associated with brain age gaps [21] and other brain health conditions [22].
`ICD10`	Mental and brain health (from ICD10 diagnosis code data)	Subject mental health and nervous system disease diagnoses that might affect the structure and function of the brain [1]. Diagnoses were grouped by categories following the ICD10 system[4].
`MEM`	Prospective memory result	Memory generally declines with age, and is related to changing brain activity patterns [23, 24].
`SEX`	Binary sex (male or female)	Highly affects the size and volume of the brain [11].

The non-imaging data is used to compute the inter-subject similarity score, which will determine the edges of the population graph.

2.3 Population graphs

One useful way to combine the multiple types of neuroimaging data and represent the relations between subjects for downstream machine learning tasks is a *population graph* data structure, as it is presented by Parisot et al. [5].

The set of N subjects S is connected into an undirected population graph $G = (V, E)$, where V is the set of graph nodes (with one node uniquely representing one subject, $|S| = |V|$), and E is the set of edges (representing the similarity of subjects).

Each node $v \in V$ is a vector containing the individual subject's neuroimaging data, whether structural, functional, or both. The edge $(v, w) \in E$ connects subjects $s_v, s_w \in S$ based on some *similarity metric* that uses the non-imaging information of the subjects to create edges between the nodes.

Defining a good similarity metric is important to account for the confounding effects on the feature vectors (e.g. the subject's sex affects the brain volume) as well as to cluster

[4] https://icd.who.int/browse10/2019/en

subjects into the most informative neighbourhoods. For example, in this dissertation the neighbourhoods that have similar brain age gaps could be useful. If carefully defined, the similarity metrics could include the domain expertise of neurologists and psychiatrists as well.

Similarity metrics are defined using a *similarity function* $\mathrm{sim}(\cdot, \cdot)$ which takes two subjects and returns the similarity score between them (the higher the score, the more similar the subjects):

$$\mathrm{sim}(s_v, s_w) = \frac{1}{n} \sum_{i=1}^{n} \mathbf{1}[M_i(s_v) = M_i(s_w)]. \tag{2.5}$$

Here $\{M_1, \ldots, M_n\}$ is a set of non-imaging features that are used to compute subject similarity and $\mathbf{1}[\cdot]$ is an indicator function, in this case returning a non-zero value when the values for a given non-imaging feature M_i match for the two subjects s_v and s_w. In practice, if the metric is a real number, "matching" could be defined in terms of non-imaging features being within some constant $\epsilon > 0$.

To avoid memory issues when $|E| \sim O(N^2)$ and minimise the size of the neighbourhood to only highly similar subjects, a *similarity threshold* μ is used such that

$$(v, w) \in E \iff \mathrm{sim}(s_v, s_w) \geq \mu. \tag{2.6}$$

The population graphs can only be effectively accepted as input to machine learning models that are capable of operating on graph-structured data. The two following sections will discuss two types of *graph neural networks* (GNNs) that will be implemented in this dissertation for this purpose, namely graph convolutional and graph attention networks. The choice of these particular GNN architectures was based on them being the most popular in their very different approaches – the former works in spectral (Fourier) domain, and the latter in spatial domain.

2.4 Graph convolutional networks

One of the approaches of how the graphs can be represented in neural networks is based on spectral graph theory [25]. The main advantage of this strategy is that it makes the operations applied to every node independent of the local graph topology (the number of neighbours of a particular node). This is easier to optimise as the same operation can be applied to all nodes.

The graph is therefore first transformed from spatial (Euclidean) to spectral (Fourier) domain. The more expensive convolution operation in the Euclidean domain corresponds to a cheaper multiplication operation in the Fourier domain.

2.4.1 Graph spectral decomposition

In spectral analysis, a graph $G = (V, E)$ is represented by its *normalised graph Laplacian* matrix [26]:

$$\mathbf{L} = \mathbf{I} - \mathbf{D}^{-1/2}\mathbf{A}\mathbf{D}^{-1/2}, \tag{2.7}$$

where $\mathbf{I}$ is the identity matrix, $\mathbf{D} \in \mathbb{N}^{N \times N}$ is the diagonal degree matrix of the graph nodes, and $\mathbf{A} \in \{0, 1\}^{N \times N}$ is the adjacency matrix such that $a_{ij} = \mathbf{1}[(i, j) \in E]$. The graph Laplacian uniquely represents the graph as it is based on the graph's topology.

The positive semidefinite graph Laplacian matrix is decomposed as

$$\mathbf{L} = \mathbf{U}\mathbf{\Lambda}\mathbf{U}^{\mathrm{T}}, \tag{2.8}$$

where $\mathbf{\Lambda}$ is the diagonal eigenvalue matrix, and $\mathbf{U}$ is the eigenbasis defining the graph Fourier domain: for a signal $\mathbf{x} \in \mathbb{R}^F$ (i.e. a graph node with F features), $\hat{\mathbf{x}} = \mathbf{U}^{\mathrm{T}}\mathbf{x}$ is the *graph Fourier transform* of $\mathbf{x}$, and $\mathbf{x} = \mathbf{U}\hat{\mathbf{x}}$ is its inverse [27].

2.4.2 Spectral graph convolution

For a filter $\mathbf{g} \in \mathbb{R}^m$ with a diagonal matrix $\hat{\mathbf{G}} = \mathrm{diag}(\mathbf{U}^{\mathrm{T}}\mathbf{g})$ containing the filter's spectral coefficients, the graph convolution of a signal $\mathbf{x}$ is defined as

$$\mathbf{x} *_G \mathbf{g} = \mathbf{U}((\mathbf{U}^{\mathrm{T}}\mathbf{x}) \odot (\mathbf{U}^{\mathrm{T}}\mathbf{g})) = \mathbf{U}\hat{\mathbf{G}}\mathbf{U}^{\mathrm{T}}\mathbf{x} \tag{2.9}$$

where $\odot$ is the element-wise (Hadamard) product [27].

2.4.3 Optimising graph convolutions

This section reviews the steps towards optimised graph convolutional networks (GCNs) as proposed by Kipf and Welling [8].

ChebNets

To avoid the $O(N^2)$ filtering operation due to the matrix-vector multiplications in Equation 2.9, Defferrard et al.'s [26] ChebNets use recursively defined *Chebyshev polynomials*

$$T_k(\mathbf{M}) = 2\mathbf{M}T_{k-1}(\mathbf{M}) - T_{k-2}(\mathbf{M}) \tag{2.10}$$

$$T_1 = \mathbf{M} \tag{2.11}$$

$$T_0 = \mathbf{I} \tag{2.12}$$

to approximate the filter $\hat{\mathbf{G}}$ as

$$\hat{\mathbf{G}} \approx \sum_{k=0}^{K} \theta_k T_k(\tilde{\mathbf{\Lambda}}) \tag{2.13}$$

where $\tilde{\mathbf{\Lambda}} = 2\mathbf{\Lambda}/\lambda_{\max} - \mathbf{I}$, $\lambda_{\max}$ is the largest eigenvalue in $\mathbf{\Lambda}$, and θ_k are coefficients such that $\hat{\mathbf{G}} = \sum_k \theta_k \mathbf{\Lambda}^k$ [27]. The truncation coefficient (polynomial order) K corresponds to the neighbourhood of at most K hops away from the node of interest, and such convolution is said to be K-localised.

Next, it can be proved [28] that

$$T_k(\tilde{\mathbf{L}}) = \mathbf{U} T_k(\tilde{\mathbf{\Lambda}}) \mathbf{U}^{\mathrm{T}} \tag{2.14}$$

with $\tilde{\mathbf{L}} = 2\mathbf{L}/\lambda_{\max} - \mathbf{I}$. The result is the approximated spectral graph convolution:

$$\mathbf{x} *_G \mathbf{g} \approx \sum_{k=0}^{K} \theta_k T_k(\tilde{\mathbf{L}}) \mathbf{x}. \tag{2.15}$$

GCNs

Kipf and Welling's GCNs [8] introduce further optimisations to ChebNets:

1. Taking $K = 1$ makes Equation 2.15 linear, which is more suitable for neural network architectures with linear layers and non-linearities between them.
2. To minimise the number of trainable parameters, θ_0 and θ_1 become functions of a single parameter θ: $\theta = \theta_0 = -\theta_1$.
3. Further assuming $\lambda_{\max} \approx 2$ simplifies the computation of $\tilde{\mathbf{L}}$, since this eliminates the need for eigendecomposition of the Laplacian: $\tilde{\mathbf{L}} = 2\mathbf{L}/\lambda_{\max} - \mathbf{I}$ becomes $\tilde{\mathbf{L}} \approx \mathbf{L} - \mathbf{I}$.

This results in convolution approximation:

$$\begin{aligned} \mathbf{x} *_G \mathbf{g} &\approx \theta_0 T_0(\tilde{\mathbf{L}})\mathbf{x} + \theta_1 T_1(\tilde{\mathbf{L}})\mathbf{x} && (2.16)\\ &\approx \theta \mathbf{I}\mathbf{x} - \theta(\mathbf{L} - \mathbf{I})\mathbf{x} && (2.17)\\ &= \theta(\mathbf{I} + \mathbf{D}^{-1/2}\mathbf{A}\mathbf{D}^{-1/2})\mathbf{x} && (2.18) \end{aligned}$$

Finally, to avoid the numerical instability of repeated convolutions due to the eigenvalues of the matrix in Equation 2.18 ranging in $[0, 2]$, a *renormalisation trick* is applied by adding self-loops to the graph nodes: taking $\tilde{\mathbf{A}} = \mathbf{A} + \mathbf{I}$ and $\tilde{\mathbf{D}}$ the degree matrix of $\tilde{\mathbf{A}}$:

$$\mathbf{x} *_G \mathbf{g} \approx \theta(\tilde{\mathbf{D}}^{-1/2}\tilde{\mathbf{A}}\tilde{\mathbf{D}}^{-1/2})\mathbf{x}. \tag{2.19}$$

2.4.4 Graph convolutional layer

The graph convolution operation in Equation 2.19 is generalised to the entire layer of the network as follows.

Let $\mathbf{H}^{(l-1)} \in \mathbb{R}^{N \times C_{l-1}}$ be the input layer of a neural network with C_{l-1} input channels, $\mathbf{H}^{(l)} \in \mathbb{R}^{N \times C_l}$ be the GCN layer with C_l output channels, and $\mathbf{H}^{(0)} = \mathbf{X} \in \mathbb{R}^{N \times C_0}$ containing C_0 input features for each of the N nodes in the population graph (in this case the neuroimaging data for N subjects). For the weight matrix $\mathbf{\Theta}^{(l)} \in \mathbb{R}^{C_{l-1} \times C_l}$ and an activation function $\sigma(\cdot)$:

$$\mathbf{H}^{(l)} \leftarrow \sigma(\tilde{\mathbf{D}}^{-1/2}\tilde{\mathbf{A}}\tilde{\mathbf{D}}^{-1/2}\mathbf{H}^{(l-1)}\mathbf{\Theta}^{(l)}). \tag{2.20}$$

2.5 Graph attention networks

In contrast to the *spectral* GCN that relies on the computation within the Fourier domain using the graph Laplacian, the graph attention network (GAT) [9] is a *spatial* architecture that operates in the Euclidean domain. Here, the mechanism that incorporates the neighbourhood information is defined to work on the neighbourhoods of different sizes, with each node learning how much importance (*attention*) it should assign to each one of its neighbours.

2.5.1 Graph attentional layer

Analogously to the previous section, we consider an input layer $\mathbf{H}^{(l-1)} \in \mathbb{R}^{N \times C_{l-1}}$ (with $\mathbf{H}^{(0)} = \mathbf{X} \in \mathbb{R}^{N \times C_0}$ as before) and the GAT layer $\mathbf{H}^{(l)} \in \mathbb{R}^{N \times C_l}$. Denote the input layer $\mathbf{H}^{(l-1)}$ as $[\mathbf{h}_1^{(l-1)} \cdots \mathbf{h}_N^{(l-1)}]^{\mathrm{T}}$, and $\mathbf{H}^{(l)}$ as $[\mathbf{h}_1^{(l)} \cdots \mathbf{h}_N^{(l)}]^{\mathrm{T}}$. For a trainable weight matrix $\mathbf{W}^{(l)} \in \mathbb{R}^{C_l \times C_{l-1}}$ and an *attentional mechanism* $a : \mathbb{R}^{C_l} \times \mathbb{R}^{C_l} \to \mathbb{R}$ we compute (unnormalised) *attentional coefficients* e_{ij}:

$$e_{ij} = a(\mathbf{W}^{(l)}\mathbf{h}_i^{(l-1)}, \mathbf{W}^{(l)}\mathbf{h}_j^{(l-1)}). \tag{2.21}$$

In Veličković et al. [9], the attention mechanism is implemented as a linear transformation using a learnable weight vector $\mathbf{a} \in \mathbb{R}^{2C_l}$, followed by a non-linearity LeakyReLU$(\cdot)$ with slope 0.2 for negative input:

$$e_{ij} = a(\mathbf{W}^{(l)}\mathbf{h}_i^{(l-1)}, \mathbf{W}^{(k)}\mathbf{h}_j^{(l-1)}) \tag{2.22}$$

$$= \text{LeakyReLU}(\mathbf{a}^{\mathrm{T}}[\mathbf{W}^{(l)}\mathbf{h}_i^{(l-1)} \parallel \mathbf{W}^{(l)}\mathbf{h}_j^{(l-1)}]), \tag{2.23}$$

where $\parallel$ denotes concatenation.

The graph topology is accounted for by discarding any coefficients e_{ij} where $(i, j) \notin E$, and normalising the coefficients $\alpha_{ij}, (i, j) \in E$ for the rest using the softmax function:

$$\alpha_{ij} = \underset{j:(i,j)\in E}{\text{softmax}}\left(e_{ij}\right) \tag{2.24}$$

$$= \underset{j:(i,j)\in E}{\text{softmax}}\left(\text{LeakyReLU}(\mathbf{a}^{\mathrm{T}}[\mathbf{W}^{(l)}\mathbf{h}_i^{(l-1)} \parallel \mathbf{W}^{(l)}\mathbf{h}_j^{(l-1)}])\right) \tag{2.25}$$

$$= \frac{\exp\left(\text{LeakyReLU}(\mathbf{a}^{\mathrm{T}}[\mathbf{W}\mathbf{h}_i^{(l-1)} \parallel \mathbf{W}^{(l)}\mathbf{h}_j^{(l-1)}])\right)}{\sum_{k:(i,k)\in E}\exp\left(\text{LeakyReLU}(\mathbf{a}^{\mathrm{T}}[\mathbf{W}^{(l)}\mathbf{h}_i^{(l-1)} \parallel \mathbf{W}^{(l)}\mathbf{h}_k^{(l-1)}])\right)}. \tag{2.26}$$

The coefficients α_{ij} and the weight matrix are used to compute the output features for another non-linearity σ:

$$\mathbf{h}_i^{(l)} = \sigma\left(\sum_{j:(i,j)\in E}\alpha_{ij}\mathbf{W}^{(l)}\mathbf{h}_j^{(l-1)}\right) \tag{2.27}$$

2.5.2 Multi-head attention

The above attention mechanism can be repeated several times to stabilise the performance, where one independent application of attention is called an *attention head*. The outputs of the independent attention heads are concatenated or averaged together until the last layer of the complete neural network architecture when they are averaged into a single output. For M attention heads, the results of Equation 2.27 are concatenated in non-final layers:

$$\mathbf{h}_i^{(l)} = \Big\Vert_{m=1}^{M}\sigma\left(\sum_{j:(i,j)\in E}\alpha_{ij,m}\mathbf{W}_m^{(l)}\mathbf{h}_j^{(l-1)}\right), \tag{2.28}$$

or averaged if GAT is the final (L-th) layer of the network:

$$\mathbf{h}_i^{(L)} = \sigma\left(\frac{1}{M}\sum_{m=1}^{M}\sum_{j:(i,j)\in E}\alpha_{ij,m}\mathbf{W}_m^{(L)}\mathbf{h}_j^{(L-1)}\right). \tag{2.29}$$

2.6 Training task

For node label prediction tasks such as brain age estimation, the population graphs are trained in a *semi-supervised* manner: while the entire dataset (every node and edge) is included in the graph, only a subset of the nodes is labelled, with the goal being to learn the labels for the remaining nodes [8]. At each training step (epoch), the feedback from nodes with available labels is used to update the parameters for the entire graph, which could be seen as information being "propagated" from a labelled node to the neighbours

that are similar to it (as defined by the similarity metric). After the model is trained, every node in the population graph has a prediction for its label.

The predictive power of a model can be evaluated based on its performance on a set of test nodes, for which the labels had been invisible to the model at the training stage. For the brain age estimation task, the metrics used to evaluate the performance are Pearson's correlation r and coefficient of determination r^2 [4] (see Section 4.2).

2.7 Requirements analysis

The project has three main stages that directly correspond to the Success Criteria of the Project Proposal (Appendix B):

S1. Implementation of the population graph data structure, with nodes containing the individual neuroimaging data and edges representing associations between them based on pairwise similarity. This involves:

  S1.1. Preprocessing of the functional, structural imaging and non-imaging data of the UKB subjects.

  S1.2. Implementation of the similarity functions that would compare two subjects and return similarity scores in increasingly flexible manner: the basic function as defined in Equation 2.5, and extensions for accepting any linear combinations of similarity features and arbitrary similarity functions.

  S1.3. A pipeline that would generate the population graph for a given set of subjects, their modalities, similarity metric and similarity threshold.

S2. Implementation of the graph convolutional network (GCN) and graph attention network (GAT) architectures for the age regression task.

S3. Training and evaluation of the two graph neural network architectures on the UKB population graph. This includes the extension of implementing the robustness measurement framework.

I expect stage S1 to be the most complex for the following additional requirements.

R1. Care must be taken to make sure that the data is handled *correctly*. This will require testing of the components which transform the data to the population graph data structure.

R2. The pipeline must be *flexible* enough to support the various combinations of modalities, subjects, and similarity metrics, which will be ensured by independent support of every modality and the *extension* of the project to support custom similarity functions.

R3. The pipeline has to be *modular* enough to be easily extended to more datasets, modalities and processing options in the future. To achieve this, the pipeline will be

split into independent components with their own designated functionality, so that they are easier to build upon independently.

These considerations by themselves largely fulfil the proposed *extension* of implementing the pipeline that can handle raw data at one of its earliest processing stages in an end-to-end manner – indeed any earlier stage would require advanced neuroimaging processing expertise that would be out of scope of a Computer Science Part II project.

The full design of the neuroimaging processing pipeline with respect to those requirements, as well as the GCN and GAT architectures and their evaluation, is discussed in Chapter 3.

2.8 Software engineering practice

Since every stage discussed in the previous section depends on the implementation of the stage before it, the straightforward workflow is to carry out the three stages in order (except for the extensions, which would be implemented last). This fits best with the *waterfall model* of software development because the project is relatively small and has a straightforward set of requirements, the design of the pipeline directly corresponds to the three project stages, and implementation depends on how the different pipeline stages are designed to interact with each other. Chapter 3 walks through the schematic design diagrams that were used as foundation for the component implementation.

To track the workflow, prevent loss of data and ease the readability and/or any future extensions to this project, the common practices of consistent code style (adhering to PEP8[6] style guide), comprehensive documentation, unit testing, code reuse, version control (through a `git` repository), and backup strategies (regular backups to an external disk, GitHub repository, and Google Drive) will be applied.

2.9 Choice of tools

Because of its popularity and wide community support in both machine learning and neuroimaging, I chose Python as the implementation language for this interdisciplinary project.

This project will use *PyTorch Geometric* [29], a popular extension library for *PyTorch* [30] that provides out-of-the-box graph neural network architecture implementations and has the advantage over other libraries (and other machine learning frameworks such as TensorFlow) of being easy to learn and use.

The neuroimaging machine learning library *Nilearn* will be used for some of the neuroimaging data processing and visualisation [31, 32].

For scientific computing, I will use common Python packages such as *scikit-learn* [33], *pandas* [34], and *NumPy* [35].

For testing, I will use Python's `unittest` library.

[6] https://www.python.org/dev/peps/pep-0008/

For model logging, visualisation and hyperparameter tuning, I will use the free *Weights & Biases* [36] framework because of its clean, intuitive interface and automated machine learning capabilities.

All third-party software used has been released under New BSD licence, except PyTorch Geometric and Weights & Biases, which have been released under MIT licence. The licences have to be included if the implementation of this project is distributed as a binary that includes the third-party software. The licences do not have to be included if the project is distributed as source code that does not contain copies of third-party software.

2.10 Starting point

The discussion of the architectures proposed in this section assumes or uses the knowledge of Computer Science Tripos courses: IA NST Mathematics, IA Scientific Computing, IA Machine Learning and Real-world Data, IB Foundations of Data Science, IB Artificial Intelligence, II Data Science: Principles and Practice, II Machine Learning and Bayesian Inference. The conceptual understanding of the brain age estimation, neuroimaging data processing and graph neural networks (none of which I had previous experience with) required the study of papers cited as well as other supporting resources.

The code builds on implementations provided by the packages listed in the previous section. I had no previous experience with most of them.

The preprocessed dataset has been provided by Dr Richard Bethlehem of Department of Psychiatry as part of ongoing methods development in UKB project 20904[7], which allows me to use the data but not openly share it.

[7] https://www.ukbiobank.ac.uk/2018/06/professor-simon-baron-cohen-genetic-and-phenotypic-investigation-of-neural-brain-network-connectivity/

3 Implementation

3.1 Overview

This project can be divided into five key components, as illustrated in Figure 3.1:

1. Preparation of the United Kingdom Biobank (UKB) dataset;
2. Intermediate population graph construction;
3. Population graph transformation for training;
4. Training on graph neural network architectures;
5. Evaluation of the graph neural network performance.

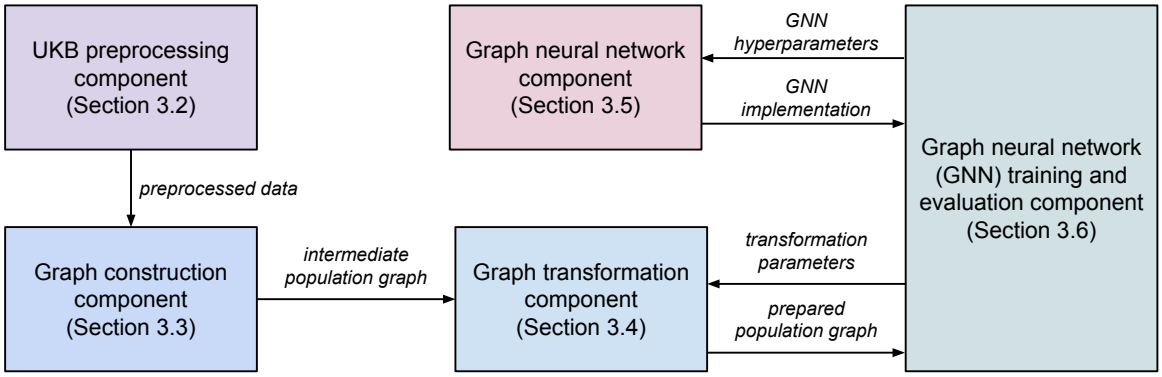


Figure 3.1: Overview of the key project components.

The work was split into these particular components so that each of them can carry out a single task independently of the other parts of the program (other than the clearly defined input/output communication), following requirement R3 (see Section 2.7). This makes the overall project easier to implement, understand and extend in the future, generalising it to other datasets and preprocessing methods.

This chapter will explain in detail the implementation behind each of the components.

3.2 UKB preprocessing component

The main function of the UKB preprocessing component is to prepare the raw or partially preprocessed UKB data for population graph construction. In particular, to improve

the computational efficiency of operations later in the pipeline (potentially saving hours or days of computation time), this component preprocesses the data in a convenient form. This involves filtering the dataset, precomputing similarity matrices and functional connectivity matrices. The schematic diagram of these steps is shown in Figure 3.2, which will be referred to throughout this section.

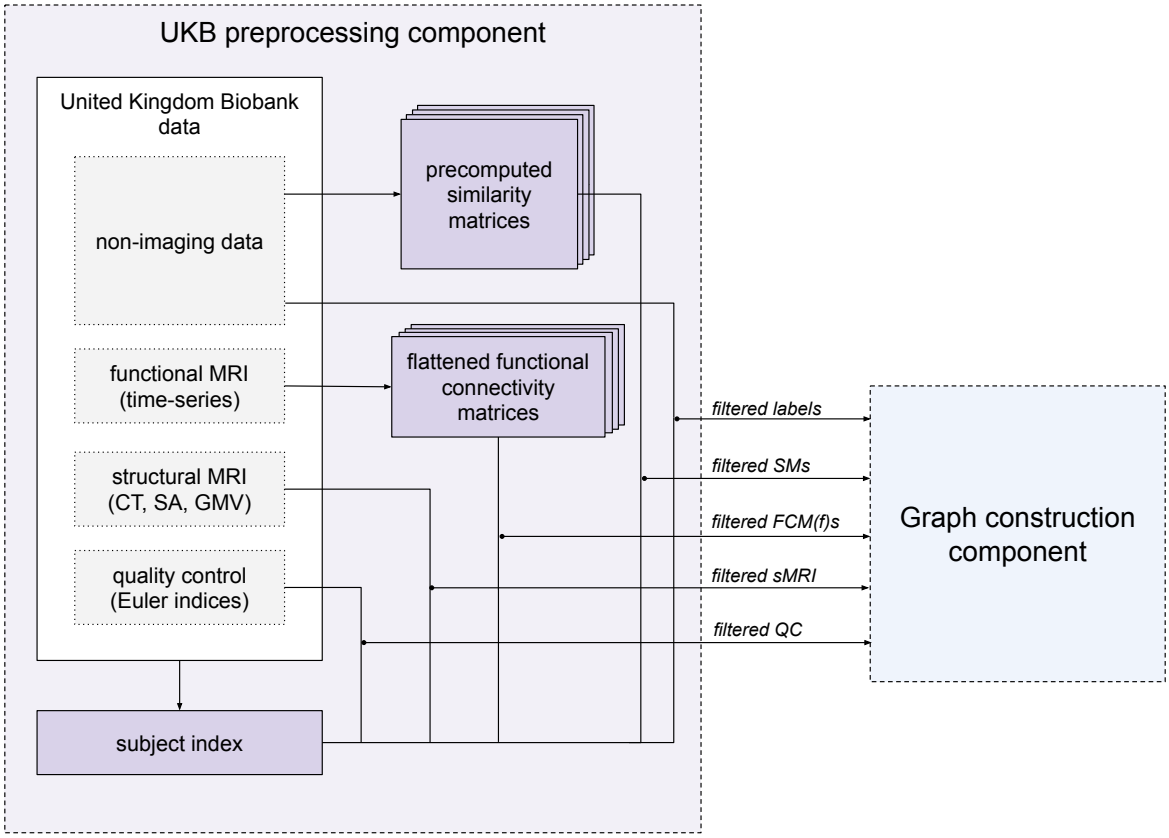


Figure 3.2: *Overview of the UKB preprocessing component.* This component precomputes the similarity matrices from non-imaging data, and functional connectivity matrices from fMRI data. A unifying subject index is collected to keep track of the subjects across modalities and filter the data for the graph construction component.

3.2.1 Cleaning the dataset

The different modalities of the UKB data, shown in the white box in Figure 3.2, have been provided separately from each other, with a small minority of subjects having the data available for some of the modalities but not others (for example, due to data corruption or its retraction by the participant). To ensure consistent and smooth processing, the subjects with partially missing data (236 in total) have been excluded from further processing. The remaining 17,314 subjects with well-defined modalities have been collected into a *subject index* (bottom left of Figure 3.2), which was used to select, filter and index the data in the subsequent components of the pipeline.

3.2.2 Precomputing connectivity matrices

The connectivity matrices involve computing pairwise correlations of 376 time-series (for 360 cortical and 16 subcortical regions) for every subject. With 20 gigabytes of raw time-series data across over 17,000 subjects, this introduces a high computational overhead (a few hours on a CPU[1] per population graph) if the matrices have to be recomputed every time the graph is constructed. To avoid this, the matrices are computed once for every subject, flattened and their lower triangles stored as `numpy` arrays as part of the preprocessing component.

3.2.3 Precomputing similarity matrices

The computation of pairwise similarity scores is quadratic in both time and space with respect to the number of subjects. Depending on the exact method of how the similarity function is computed, a general-purpose (non-accelerated) processor might take hours or even days to process the entire dataset. The repeated computation would also be wasteful, since the similarities for a single non-imaging feature do not change over different population graphs: the variation comes from different *selections* of subjects and non-imaging features, their relative weighting, and similarity thresholds.

The similarity matrices – one for each non-imaging feature in Table 2.2 – have therefore been computed in advance. Unlike the functional connectivity data that is used directly for population graph node features (and is flattened and reduced to avoid redundancy), the feature-wise similarity matrices are sliced and filtered depending on the selection of subjects. In this case, it is more practical to store the full matrix, so that the integer indices into the similarity matrix directly correspond to the subject index in other components of the population graph data structure (see Sections 3.2 and 3.3.5).

Having computed the feature-wise similarity matrices, their linear combination for a full similarity score (by default adding matrices together and dividing the result by a constant) can efficiently make use of vectorised matrix operations.

For the `ICD10` metric, the subjects were considered to be `ICD10`-similar whenever they had at least one shared mental health or nervous system diagnosis. Two patients without any mental health or nervous system diagnoses were *not* considered to be similar, however, since similarity in having a healthy brain is extremely common (thus not in itself informative) while causing memory issues (see Section 3.6.1).

The similarity computation was vectorised in order to make use of hardware acceleration and reduce the compute time: for the boolean `ICD10`-lookup matrix $\mathbf{F}_{\text{ICD10}}$ with rows indexed by subjects and columns by relevant `ICD10` diagnoses, the pairwise similarity matrix $\mathbf{M}_{\text{ICD10}}$ computation corresponds to

$$\mathbf{M}_{\text{ICD10}} = \mathbf{1}\left[\mathbf{F}_{\text{ICD10}}\mathbf{F}_{\text{ICD10}}^{\text{T}} \geq 1\right] \tag{3.1}$$

[1] Measured on a Computational Biology lab machine.

with the indicator function $\mathbf{1}[\cdot]$ applied element-wise.

For the remaining metrics (e.g. years of full-time education, `FTE`) there is only one integer or floating-point value per subject, with values compared for equality. The computation is vectorised by exploiting tensor broadcasting semantics[2] that copy rows and columns as necessary for the matrix dimensions to match. For the vector of subject `FTE`s, $\mathbf{f}_{\text{FTE}}^{\text{T}} \in \mathbb{R}^{N\times 1}$ and $\mathbf{F}_{\text{FTE}} = [\mathbf{f}_{\text{FTE}}^{\text{T}} \cdots \mathbf{f}_{\text{FTE}}^{\text{T}}] \in \mathbb{R}^{N\times N}$, `FTE`-similarity matrix is defined as

$$\mathbf{M}_{\text{FTE}} = \mathbf{1}\left[\mathbf{F}_{\text{FTE}} = \mathbf{F}_{\text{FTE}}^{\text{T}}\right]. \tag{3.2}$$

3.3 Graph construction component

The next stage of the pipeline involves constructing the "intermediate representation" of the population graph. The intermediate representation contains the graph topology and node features, but is not prepared for training – it is not split into training, validation and test sets, and its features are not normalised. The two stages are separate because this allows for the reuse of the same intermediate representation for different dataset splits and other training parameters without having to reconstruct the edges in $O(N^2)$ time for N subjects. The steps for the data processing in this component are schematically visualised in Figure 3.3.

3.3.1 Inputs

The inputs to the graph construction component can be categorised into four types as shown in the box at the bottom left of Figure 3.3:

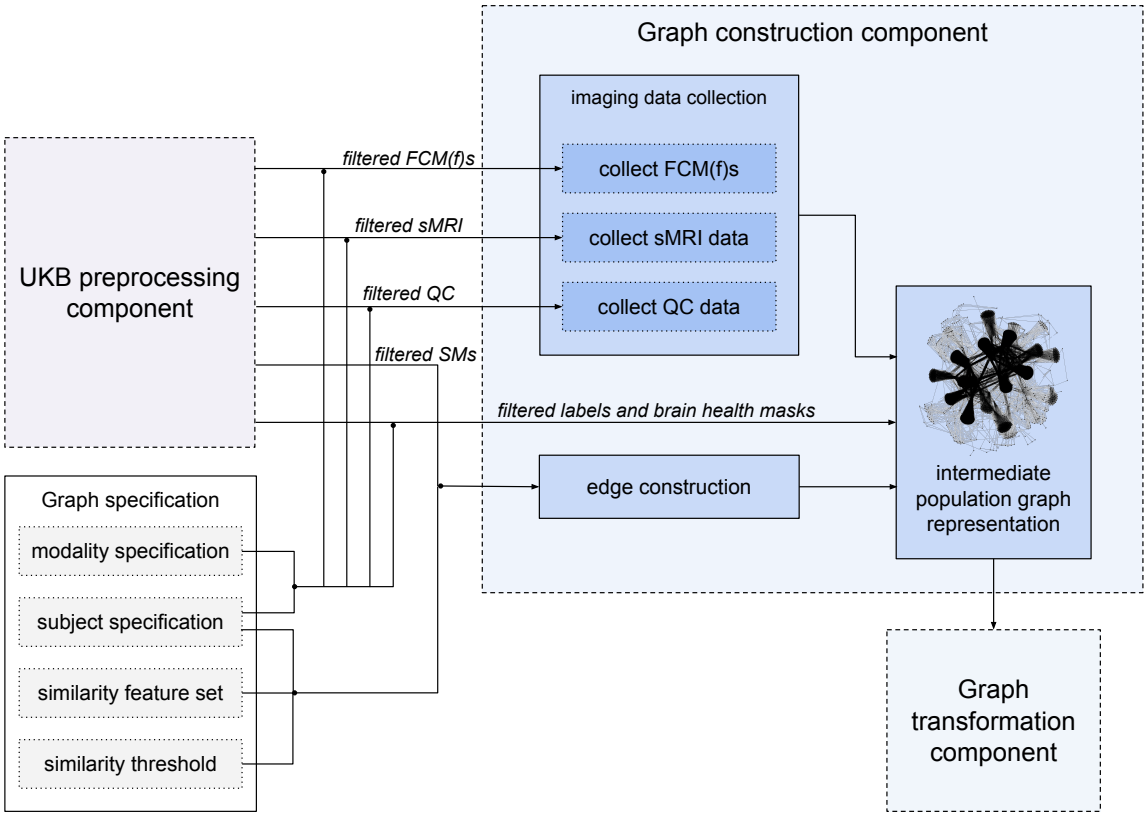
1. *Modality specification* describes which of the neuroimaging modalities should be used as node features. This could be any combination of functional, structural and quality control data, following the pipeline flexibility requirement R2 (Section 2.7).

2. *Subject specification* is an optional argument overriding the default option to use the entire dataset (filtered by the subject index described in the previous component) when constructing the population graph. In this case, a number of subjects or a list of particular UKB identifiers could be provided.

3. *Similarity specification.* In its default implementation, the similarity score is computed as the average over a set of similarity features $\{M_1, \ldots, M_n\}$ (see Equation 2.5), in which case it is sufficient to specify the similarity feature set to be used. An extension to this is to accept an arbitrary linear combination of various similarity features, allowing for much richer similarity metrics, again increasing the flexibility of the pipeline (see Requirement R2 in Section 2.7).

4. *Similarity threshold.* A number $\mu \in [0, 1]$ defining the threshold for the similarity metric above which an edge will be added to the graph (see Equation 2.6).

[2] https://pytorch.org/docs/stable/notes/broadcasting.html



FCM(f): functional connectivity matrix (flattened), SM: similarity matrix, sMRI: structural MRI, QC: quality control data

Figure 3.3: *Overview of the graph construction component.* The raw and preprocessed UKB data is collected according to the graph specification, and combined into the intermediate population graph representation containing the different modalities of brain imaging features, the node labels and the edges.

3.3.2 Imaging data collection

Based on the subject and modality specification, the relevant imaging data is collected from the raw UKB files (first filtered by subject index) and stored in the intermediate representation as a *dataframe* (`pandas.DataFrame` object) indexed by UKB subject identifier. This is represented by the connections between UKB preprocessing component, graph specification and the blue imaging data collection box in Figure 3.3.

If a particular modality is unused, the dataframe stored is empty.

3.3.3 Edge construction

The edge construction component uses the similarity specification and the filtered similarity matrix data to find the similarity scores for each pair of subjects determined by the subject specification. Whenever the similarity score between two subjects exceeds the threshold, an undirected edge is added to the population graph. The edges are stored in a *tensor* (`torch.tensor`), a PyTorch datatype for multi-dimensional matrices[3].

[3]https://pytorch.org/docs/stable/tensors.html

3.3.4 Brain health mask computation

Following the brain age estimation method discussed in Section 2.1, the machine learning model can only be trained on subjects with healthy brains, although the population graph may contain both healthy and non-healthy subjects. The *brain health mask* is therefore computed to determine which subjects can be used to train the model and which cannot. In particular, non-healthy subjects are not included in loss function computation, so that the direction of parameter update depends on healthy subjects only (though the parameters are updated for the entire graph). As a result, the age predictions are available for both healthy and non-healthy subjects, but the use of a brain health mask ensures that the prediction corresponds to the brain age rather than chronological age, as discussed in Section 2.1.

In this project, the brain health is approximated by the absence of diagnoses related to mental health or nervous system disorders, defined by the `ICD10`-similarity metric (see Table 2.2).

3.3.5 Population graph representation

The population graph is stored in an extended `torch_geometric.Data` object[4], with its most important fields listed in Table 3.1. The intermediate population graph representation has all its entries defined except for the feature vector `x` and the training, validation and test masks.

Table 3.1: The population graph data structure (excludes helper or utility fields).

Field name	Type	Description
`subject_index`	string array	UKB identifiers of the subjects. Stored in the same subject order as training masks, feature and label tensors; corresponds to the edge start and end values.
`edge_index`	$2 \times 2\|E\|$ long tensor	`edge_index`$[0][i] = s_v$ and `edge_index`$[1][i] = s_w$ indicate a directed edge $s_v \rightsquigarrow s_w$. Following the PyTorch Geometric API, to represent the undirected edge $(s_v, s_w) \in E$, another directed edge $s_w \rightsquigarrow s_v$ is added.
`functional_data`	dataframe	Row-indexed by subject with columns containing the flattened functional connectivity matrix entries. Empty if no functional data is used in the population graph.

Continued on next page

[4]https://pytorch-geometric.readthedocs.io/en/latest/modules/data.html

Table 3.1 – *Continued from previous page*

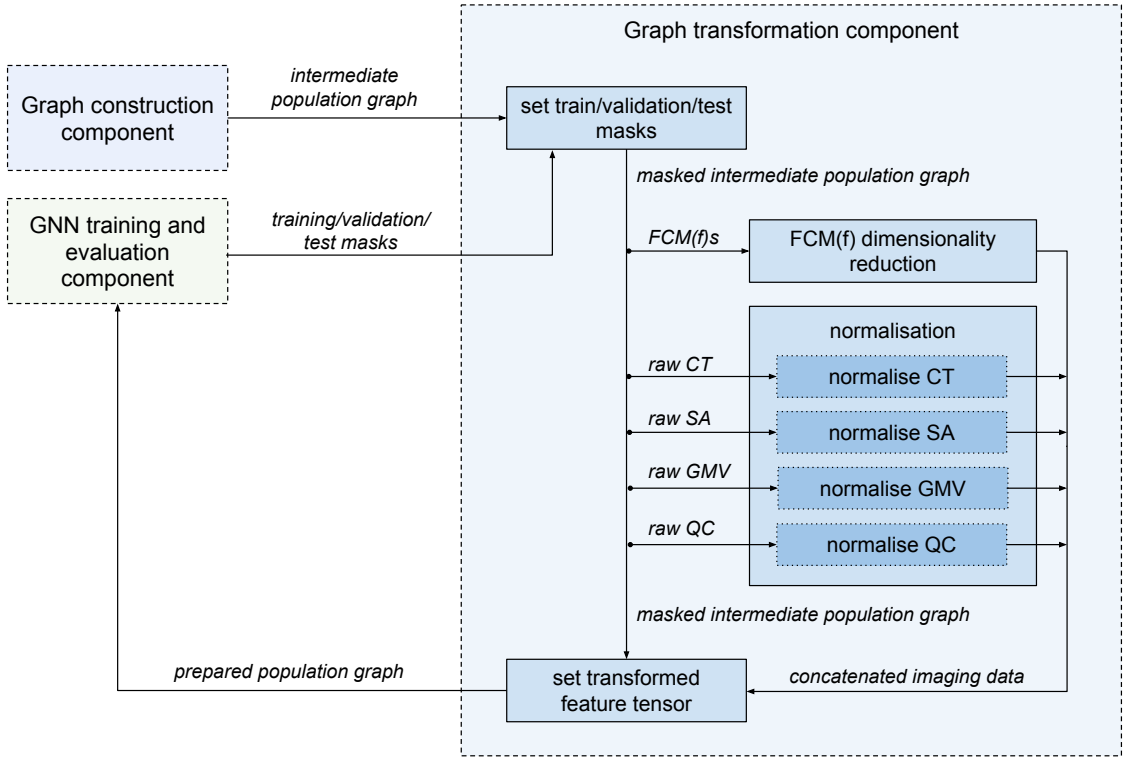
Field name	Type	Description
`structural_data`	dictionary of dataframes	Dictionary is indexed by the structural data modality, in this case cortical thickness, surface area, and grey matter volume. The corresponding dataframes are row-indexed by subject with columns containing the features of the relevant structural data modality. The dataframes are empty if no structural data is used.
`quality_control_data`	dataframe	Row-indexed by subject with two columns containing Euler indices for the left and right hemispheres of the brain. Empty if no quality control data is used.
`x`	$N \times F$ float tensor	*Unused at the intermediate stage.* Contains the full normalised feature vector (of F features) for every graph node (subject).
`y`	$N \times 1$ float tensor	Contains the labels of training data, in this case chronological age.
`brain_health_mask`	boolean array	`True` indicates that the subject has a healthy brain and can be used for training, and `False` otherwise.
`train_mask`	boolean tensor	*Unused at the intermediate stage.* `True` if the subject belongs to the training set, and `False` otherwise.
`validation_mask`	boolean tensor	*Unused at the intermediate stage.* `True` if the subject belongs to the validation set, and `False` otherwise.
`test_mask`	boolean tensor	*Unused at the intermediate stage.* `True` if the subject belongs to the test set, and `False` otherwise.

3.3.6 Testing

As discussed in requirement R1 for this project (see Section 2.7), it is important to ensure that the data is preprocessed and collected into the population graph correctly. To this end, both the UKB preprocessing (Section 3.2) and the graph construction components have unit test modules asserting the correctness of the similarity matrices and example graph topologies.

3.4 Graph transformation component

The graph transformation component is responsible for preparing the intermediate population graph representation for training by defining its normalised, concatenated feature vector as well as training, validation and test masks, using the parameters provided by the training component. The schematic diagram representing the population graph transformations in this component is shown in Figure 3.4.



CT: cortical thickness, FCM(f): functional connectivity matrix (flattened), GMV: grey matter volume, SA: surface area, QC: quality control

Figure 3.4: *Overview of the graph transformation component.* This component first sets the training, validation and test masks of the intermediate population graph. The masks are further used to reduce the dimensionality of fMRI data and normalise the remaining brain imaging data. The different brain imaging data modalities are combined into a single (node) feature tensor and assigned to the intermediate population graph (now masked with training sets), preparing it for training in the GNN training and evaluation component.

3.4.1 Setting the training masks

The first operation for preparing the population graph for training involves setting the `train_mask`, `validation_mask` and `test_mask` fields of the intermediate population graph data structure (see Table 3.1). The masks are defined in the GNN training and evaluation

component as this depends on the training procedure, not the structure of the population graph itself.

Following the brain age estimation method (Section 2.1), the masks are intersected with the `brain_health_mask`. Similarly to the function of the brain health mask, the three training masks are used to determine which nodes should be used to compute the loss function during various stages of training. For example, the model parameters are only updated based on the loss function for the subjects filtered with the `training_mask`, and validation loss is computed only for the subjects filtered with the `validation_mask`.

3.4.2 Functional connectivity matrix dimensionality reduction

The flattened functional connectivity matrix results in over 70,000 features per patient. Since the population graph cannot be split into parts and during training must be kept entirely in memory, along with all model parameters, this quickly runs into memory issues when the entire dataset is used. To mitigate this issue, some techniques may be applied to reduce the dimensionality of functional data. This project uses principal component analysis (PCA) as the dimensionality reduction technique of choice as it is one the simplest to implement. When the functional data is used to construct the population graph, PCA transformation is fitted to the training set as specified by the `training_mask`, and the same transformation is applied to the remaining subjects. This dimensionality reduction technique is optional in the pipeline, and it is possible to define how many principal components should be kept after the transformation to control the information loss. By default, only the most important 1% of the components is kept so that the number of functional and structural features have the same order of magnitude.

3.4.3 Structural MRI and quality control data normalisation

Next, the raw features stored under `structural_data` and `quality_control_data` are normalised to be in the range between -1 and 1 with mean 0. Similar to the previous section, in order to avoid data leakage a standard scaler is fitted to the training set only (as specified by the `training_mask`) and then applied to the validation and test sets. A separate transformation is applied to each structural data modality (cortical thickness, surface area, grey matter volume) and the quality control modality.

3.4.4 Setting the transformed feature tensor

The transformed functional, structural and quality control features are concatenated together into a single tensor, which is assigned to `x` in the population graph data structure (Table 3.1). This completes graph transformation for training.

Since the original neuroimaging dataframes and brain health masks are kept unchanged in the data structure, the same population graph can be prepared for, say, a different training fold by simply going through the transformation component again but with a different set of training masks, which will reset the values in the corresponding population graph fields.

3.5 GNN architecture component

The graph neural network component contains implementations for the graph neural network (GNN) architectures used in this project: the graph convolutional network (GCN, Section 2.4) and the graph attention network (GAT, Section 2.5). The networks are implemented as two PyTorch modules called `BrainGCN` and `BrainGAT`, extending a shared `BrainGNN` module. Table 3.2 presents the parameters used to define a `BrainGNN` instance. Then instantiating the subclasses `BrainGCN` and `BrainGAT` simply amounts to setting the `conv_type` parameter to either `GCN` or `GAT`, and setting the convolutional layers to either `torch_geometric.nn.GCNConv` or `torch_geometric.nn.GATConv` respectively, reusing the rest of the code. The implementations for `GCNConv` and `GATConv` are available in the PyTorch Geometric library.

Table 3.2: The parameters for the `BrainGNN` module.

Parameter	Type	Description
`conv_type`	string	Indicates the type of graph convolutional layer to be used (graph convolution or graph attentional layer).
`layer_sizes`	integer array	Lists the number of units in every hidden layer. The length of the array corresponds to the total number of hidden layers.
`n_conv_layers`	integer	Number of convolutional layers. Must be in range $[0, \texttt{len(layer\_sizes)}]$. The sizes of those layers are determined by the first `n_conv_layers` values of the `layer_sizes` and `n_node_features` parameters.
`num_node_features`	integer	Indicates the number of input features.
`dropout_p`	float $\in [0, 1]$	The probability of ignoring the node in a hidden layer at training time. Used as a regularisation technique to reduce overfitting.

Listing 3.1 shows how the layers are combined in the `BrainGNN` architecture for a given set of parameters in Table 3.2. Hyperbolic tangent ($\tanh(\cdot)$) was chosen as the non-linearity (activation function) between each layer. This is because the population graph features and weights in neurons may be negative, in which case other popular activation functions such as ReLU($\cdot$) may have an undesirable asymmetric response.

A *dropout* (`torch.nn.Dropout`) layer was added between every fully connected layer in line 21 as a *regularisation* technique to avoid overfitting. At each training step, a unit's weight is zeroed with probability `dropout_p`, so that the neural network does not rely on

any particular units when predicting age and learns more robust features.

Listing 3.1: Simplified code snippet for `BrainGNN` instantiation and training.

```
class BrainGNN(torch.nn.Module):
    def __init__(self, conv_type, n_node_features, n_conv_layers,
                 layer_sizes, dropout_p):

        # Refer to the layers of the same type through a list.
        self.conv = torch.nn.ModuleList() # Convolutional layers.
        self.fc = torch.nn.ModuleList() # Fully connected layers.
        self.dropout = torch.nn.ModuleList() # Dropout layers.

        # Add convolutional layers of conv_type
        # ...
        # Add remaining fully connected and dropout layers.
        # ...

    # Forward propagation shows how the layers are applied.
    def forward(self, graph):
        x, edge_index = graph.x, graph.edge_index
        for i in range(len(self.conv)):
            x = self.conv[i](x, edge_index)
            x = torch.tanh(x)
        for i in range(len(self.fc) - 1):
            x = self.fc[i](x)
            x = torch.tanh(x)
            x = self.dropout[i](x)

        # Apply the last fully connected layer without the activation
        ↪ function or dropout.
        x = self.fc[-1](x)
        return x
```

3.6 GNN training and evaluation component

The main function of the GNN training and evaluation component is to select the best combination of population graph and GNN hyperparameters for each of the GCN and GAT architectures, and to report the results of the best model. It also contains the *robustness evaluation* component, where robustness in this project is conceptualised as the predictive power drop when noise is added to the population graph nodes and/or edges. The schematic overview of the GNN training and evaluation component is shown in Figure 3.5.

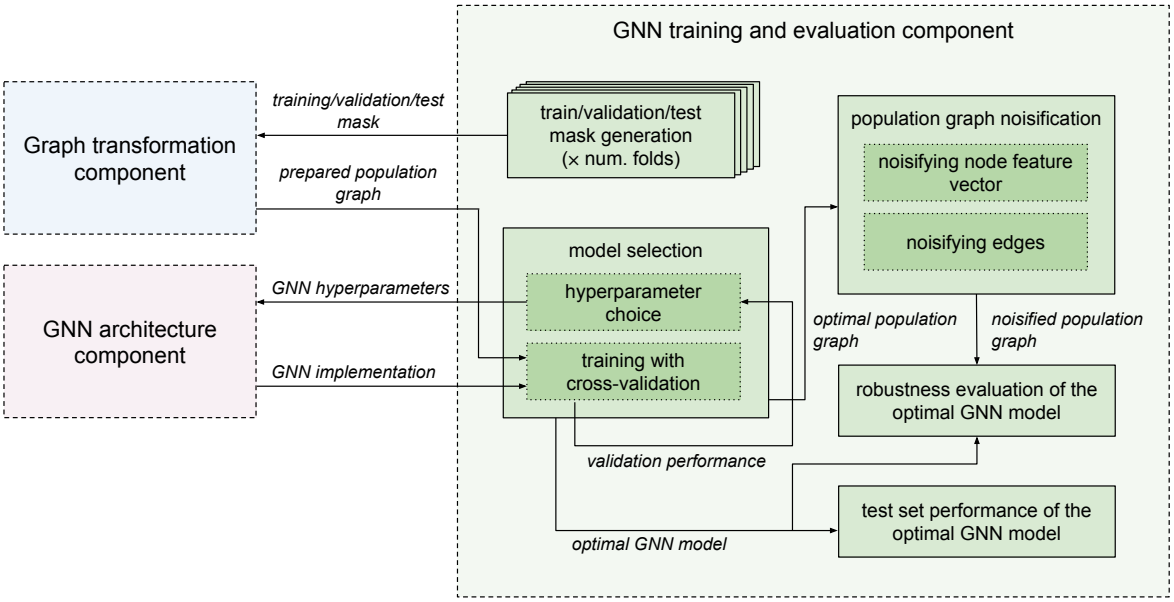


Figure 3.5: *Overview of the GNN training and evaluation component.* The component first generates sets of training, validation and test masks for a required number of folds, used to prepare the population graphs for training. Next, the model selection sub-component repeatedly generates new sets of hyperparameters, training various GNN and population graph combinations, and collecting their cross-validation performance scores on the validation set. As the hyperparameter tuning converges, the optimal GNN model is evaluated on the test set. Finally, the robustness measurement framework noisifies the nodes and edges of the optimal population graph, and evaluates the change in performance of the optimal GNN model.

3.6.1 Model selection

The brain age modelling task is challenging because of a vast combination of possible population graph and GNN hyperparameter choices. First, there is a choice of a neuroimaging modality combination for population graph node features (from functional, structural, and quality control data). Second, there are at least 32 combinations of possible non-imaging features used for similarity metrics (see Table 2.2), with each feature combination having a range of possible similarity thresholds. This does not account for the extension of using arbitrary linear combinations of non-imaging features for a similarity metric, and the fact that the UKB has over 760 more features that could potentially be included in similarity computation. Third, having fixed the population graph parameters, `BrainGNN` is in theory unlimited in its design and parameterisation possibilities (Table 3.2). Fourth, having fixed population graph and GNN hyperparameters, there are additional hyperparameters related to GNN training, such as the number of epochs and learning rate.

Having considered this, this section discusses further the motivation and reasoning behind most of the hyperparameter constraints and the training method. The full list of hyperparameter ranges that have been searched is included in Listing A.1.

Memory constraints

In the brain age estimation task with population graphs, the entire population graph containing node neuroimaging features, edges, GNN model training parameters and intermediate training states must be stored in memory at the same time. In addition, the population graph cannot be split into smaller independent batches since all nodes of the graph must be updated in a single training step, and it would be a non-trivial task to keep track of the edges and other interactions between the batches.

As a result, many hyperparameter combinations cause the model to run into memory issues. In practice, the models were limited to up to 8 gigabytes of GPU memory. This motivated the following hyperparameter constraints:

1. *Excluding functional data modality.* From preliminary experiments, using only functional and quality control data resulted in a poor graph neural network performance for the brain age estimation task. This is supported by the literature, such as the work of Niu et al. [4], which concluded that features derived from functional data were either not useful or even had a negative effect on predictive power, while structural features (particularly grey matter volume) were the most useful. Most importantly, functional data modality has over 70,000 features per subject (compared to 1,084 features for all remaining modalities combined), which results in an explosion in parameters for small estimated gain in performance.

   Dimensionality reduction is possible, but from experience in the Data Science unit of assessment, cutting out half of the low-variance principal components can even worsen the predictive power and training time of the model instead of improving it.

   While the functional data modality and dimensionality reduction technique were implemented – indeed the goal of the pipeline was to support *flexibility* to construct any population graph if needed (requirement R2, see Section 2.7) – they were not used in training.

2. *Fixing the model size to a small set of shallow networks with a small number of units.* The learning rates were decreased and number of epochs increased to compensate for the smaller number of training parameters. The exact architecture is provided in Listing A.1.

3. *Fixing the set of possible similarity feature sets and simlarity thresholds.* The default mode of averaging the feature-wise scores was used instead of arbitrary linear combinations, because, having no domain-specific knowledge of the relative importance of different confounders, this was the most non-informative, unbiased choice. From running some initial models, the feasible similarity thresholds have been all above 0.6, with even 0.7 or 0.8 often causing memory issues. This additionally motivated the use of as many non-imaging features as possible. `SEX` was always included in the feature sets because it is known to be an important confounder, significantly affecting the volume of the brain and sometimes even requiring a separate model for every sex [1]. The full list of population graphs tested can be found in Listing A.1.

Hyperparameter tuning

The hyperparameters were tuned using the Bayesian optimisation strategy provided by the *Weights & Biases* (`wandb`) [36] machine learning tracking and optimisation framework. Given the initial distributions of hyperparameters that should be searched (see Listing A.1) and a command-line script that accepts hyperparameter combinations as arguments, `wandb` automatically selects the next set of hyperparameters that it believes is the most likely to improve a given performance metric. The choice of hyperparameters is based on both the initial hyperparameter distribution and the feedback from previously attempted values. For more information on Bayesian hyperparameter optimisation see Snoek et al. [37].

Unlike in other hyperparameter tuning techniques such as grid search, Bayesian optimisation can use hyperparameter distributions defined over all real values in a given range. Consequently, hyperparameter search cannot be exhaustive and there is no obvious stopping point. In this project, hyperparameter tuning was run on each of the architectures until the models seemed to converge to similar performance with only marginal or no improvements as more hyperparameter combinations were tried. In general, this meant at least 100 hyperparameter combinations per GNN architecture, each hyperparameter combination trained five times for each cross-validation fold (see the next section), amounting to close to two weeks of total GPU compute time.

Training procedure

Before running the model selection and hyperparameter tuning procedure, the dataset is split into 90% training/validation, and 10% hold-out test set, stratifying by subject age and sex (as features that are the most likely to affect the brain age prediction). The test set is never used in training and hyperparameter tuning, and is only looked at during the evaluation procedure after the best hyperparameters for each of GCN and GAT have been selected (as indicated by the box at the bottom right in Figure 3.5). The subjects in the test set correspond to the `test_mask` field of the population graph data structure as discussed in Table 3.1 and Section 3.4.1. While it is common to have multiple test sets (in techniques such as nested cross validation), this project does not use it because of the high computational cost and because the dataset is much larger than a typical brain imaging dataset (below 1,000 samples [5, 38]), making it more representative of the population.

The remaining data is used for hyperparameter tuning. Each hyperparameter combination is trained using a stratified five-fold *cross-validation*: the training/validation dataset is split into five folds with 90% training and 10% validation subjects, again stratifying by age and sex. These correspond to `train_mask` and `validation_mask` of the population graph data structure. This model selection strategy to a large extent follows the one in Raschka [39], Section 3.7.

The summary of the above steps is shown in Figure 3.5: the data splits are visualised with a set of different training/validation/test masks at the top left corner of the GNN training

and evaluation component (all of them having the same test mask but different training and validation masks). The masks of each fold are used to transform the graph (in the graph transformation component), and for each fold the prepared population graph is trained on a GNN implementation with a given set of hyperparameters (as suggested by the `wandb` framework). The optimal GNN model and population graph hyperparameters are selected based on which hyperparameter combination gives the smallest mean squared error (MSE), averaged over the validation sets of each fold. MSE was chosen as the standard loss function used in regression tasks.

To prevent overfitting and optimise the training procedure, for every fold the models are trained with *early stopping*: while validation loss itself is not used to update the parameters, the training stops as validation loss begins to increase with decreasing training loss (indicating overfitting on training data). At this point the weights are reverted back to when the validation loss was the smallest. In this project, the models are trained for at least 1,000 epochs and stopped early if MSE does not decrease by at least 0.005 over 100 consecutive iterations.

As the hyperparameter tuning converges, the most promising model for each of the `BrainGCN` and `BrainGAT` architectures is selected to be tested on the hold-out test set. The metrics most commonly used in the literature for evaluating performance for the brain age (or any other regression) task [40, 4] are Pearson's correlation r and coefficient of determination r^2 (which will be further discussed in Section 4.2).

3.6.2 Robustness measurement

The proposed *extension* for the training and evaluation component was to evaluate the robustness of the GNN models to the noise in the population graph, measured by the rate of performance drop with increasing noise. In this project, robustness is evaluated in two ways, testing for two different effects of noise: one of them associated with the noisy neuroimaging data, and the other with the population graph representation of the dataset.

MRI data can be very noisy [40, 4], and most neuroimaging preprocessing pipelines (such as the ones used for UKB dataset) use various techniques to remove noise caused by subject head motion, image distortion and other factors [15]. This does not eliminate all noise, however, which motivates the development of models that could correct for those errors. To test how the model performance changes with noise, increasing proportions of nodes could be corrupted. One way to do this would be to add Gaussian noise, but this leaves it unclear how to choose the optimal noise amplitude. A more aggressive way would be to corrupt the nodes completely by, for example, permuting their features.

Another advantage of using the population graph (rather than treating every subject as an independent example) is that the associations between subjects (captured through graph edges and defined by subject similarities) incorporate additional non-imaging information which could improve the predictions and account for confounding effects. To test how much the model relies on edge information, some fraction of edges that should have

been present in the population graph could be removed, observing the change in model performance.

3.7 Repository overview

The repository follows the structure of the five key project components shown in Figure 3.1. Its overview is presented in Table 3.3.

Table 3.3: Repository overview.

Component	**Filename**	**Notes**
UKB preprocessing	`phenotype.py`	Contains enumeration of non-imaging features, mappings of non-imaging feature names to their UKB codes.
	`ukb_preprocess.py`	Functionality described in Section 3.2.
	`ukb_preprocess_test.py`	Tests for similarity matrix code.
Graph construction	`graph_construct.py`	Functionality described in Section 3.3.
	`graph_construct_test.py`	Tests for example population graph correctness.
Graph transformation	`graph_transform.py`	Functionality described in Section 3.4.
GNN architectures	`brain_gnn.py`	Implementation of `BrainGNN` and subclasses `BrainGAT`, `BrainGCN` (Section 3.5).
GNN training	`brain_gnn_evaluate.py`	Population graph noisification and robustness measurement procedure (Section 3.6).
	`brain_gnn_train.py`	Cross-validation training procedure (Section 3.6).
	`wandb_sweep.yaml`	Initial hyperparameter distributions used by `wandb` (Listing A.1).
	`wandb_train.py`	Command line script used by `wandb` to automatically test new hyperparameters.

4 Evaluation

This chapter presents the best-performing GCN and GAT models (Section 4.1) and evaluates their performance (Section 4.3) against standard evaluation metrics (Section 4.2). It additionally analyses the significance of these results (Section 4.4), the model robustness to population graph node noise (Section 4.5), and their dependence on population graph edges (Section 4.6). Finally, it discusses these findings from a broader perspective (Section 4.7).

4.1 Model ranking and selection

After the hyperparameter tuning process from Section 3.6.1, models were selected according to the following procedure (applied separately to the GCN and GAT model families):

1. Models were ranked by ascending average MSE loss. The model with the lowest average MSE was chosen as the reference model.
2. Models whose one standard deviation interval from their MSE did not overlap with the one standard deviation interval of the reference model MSE were excluded from ranking.

Cross-validation performances of the best-scoring models selected are shown in Figure 4.1. The hyperparameters for each of the short-listed models are listed in Tables A.3 and A.4 (Appendix A).

Table 4.1: Best performing population graph and GNN model parameter combinations during the model selection process.

Hyperparameter	**GCN1**	**GAT1**
Similarity feature set	`FI`, `FTE`, `ICD10`, `MEM`, `SEX`	`FI`, `ICD10`, `MEM`, `SEX`
Similarity threshold	0.9	0.8
Layer sizes	[1024, 512, 512, 256, 256, 1]	[2048, 1024, 512, 256, 128, 1]
# convolutional layers	5	2
Dropout	3.22×10^{-1}	3.14×10^{-3}
Learning rate	6.98×10^{-3}	1.34×10^{-2}
Weight decay	1.31×10^{-2}	6.05×10^{-4}

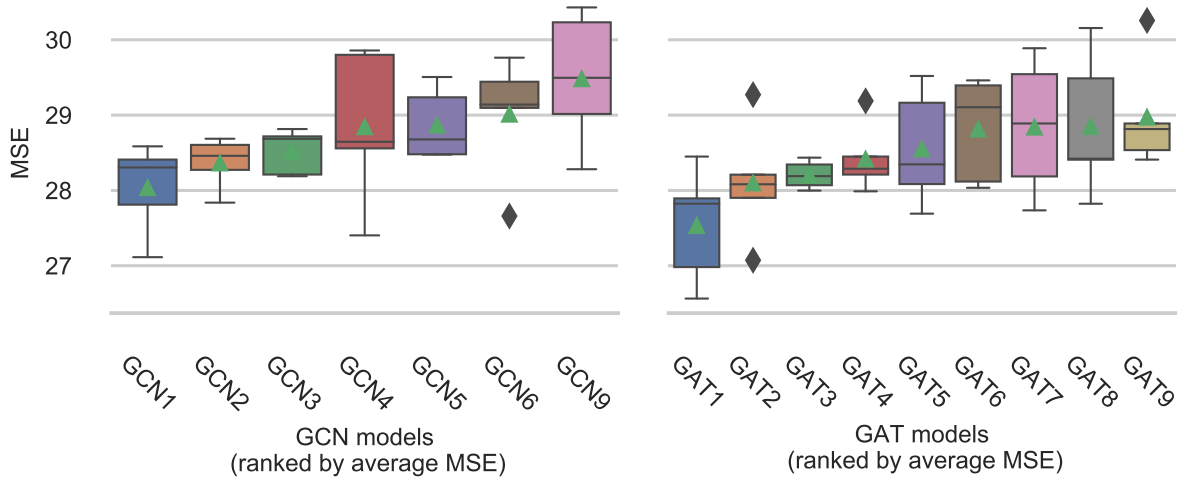


Figure 4.1: Highest scoring population graph and GNN parameter combinations for GCN (left) and GAT (right). The models are named according to their convolution type and ranked by ascending average MSE loss (indicated by the green triangle).

The best-ranked GCN1 and GAT1 models seemed to be the most promising and therefore have been selected for further evaluation. Their population graph specification, GNN architecture and hyperparamters are listed in Table 4.1. While the hyperparameters for the two architectures are very different, it is hard to make a definite qualitative comparison between them because of high and non-systematic variation in hyperparameter combinations over all short-listed models (Appendix A).

4.2 Evaluation metrics

The main performance metrics used for regression problems, including the brain age estimation task, are *Pearson's correlation* and *coefficient of determination*.

Pearson's correlation

For sets of true labels $\mathbf{y} = [y_1 \dots y_N]$ with mean $\bar{y}$ and predicted labels $\hat{\mathbf{y}} = [\hat{y}_1 \dots \hat{y}_N]$, *Pearson's correlation* is computed as

$$r(\mathbf{y}, \hat{\mathbf{y}}) = \frac{\text{cov}(\mathbf{y}, \hat{\mathbf{y}})}{\sigma_{\mathbf{y}} \sigma_{\hat{\mathbf{y}}}}, \tag{4.1}$$

where $\text{cov}(\cdot, \cdot)$ denotes covariance and σ stands for standard deviation.

Coefficient of determination

The *coefficient of determination* indicates how much variance between predicted ($\hat{\mathbf{y}}$) and actual ($\mathbf{y}$) labels could be explained by the model. It is computed as

$$r^2(\mathbf{y}, \hat{\mathbf{y}}) = 1 - \frac{\sum_i (y_i - \bar{y})^2}{\sum_i (y_i - \hat{y}_i)^2}. \tag{4.2}$$

Higher values for both metrics (with maximum 1) indicate a higher level of agreement between the true and predicted labels and therefore higher predictive power.

4.3 Test set performance of selected models

In general, after using cross-validation for model selection, the model is retrained on the entire dataset before giving a point estimate on a hold-out test set [39]. This is because training on more data, especially when the dataset is small, gives better generalisation and therefore better predictions on the unseen data. However, in this project the validation set was also used for early stopping since neural networks are especially prone to overfitting [41]. Investigation of hyperparameter tuning has shown that applying the stopping criteria discussed in Section 3.6.1 on just the training set would have still led to convergence only after the model had already overfit on the unseen validation labels. On the other hand, it is unclear how the stopping criteria should be adjusted when the training set size increases.

Considering that the UKB dataset is large and that retraining the model using more data but without early stopping might not improve generalisation, all cross-validation folds were kept for test set performance estimation. Table 4.2 gives the hold-out test set estimates for the metrics discussed in Section 4.2.

Table 4.2: Test set performance of GCN1 and GAT1 models (over the five early stopping folds of the training set).

Model	MSE	r	r^2
GCN1	28.045 ± 0.595	0.675 ± 0.008	0.445 ± 0.010
GAT1	27.543 ± 0.758	0.670 ± 0.005	0.477 ± 0.008

The following sections will consider a single fold of GCN1 and GAT1 models (having the same training/validation/test split) to ensure that any variation is due to the experimental setup and not the model weights or distributions of subjects across the folds. The fold was selected arbitrarily to be the first one returned by the stratified splitting procedure.

4.4 Significance testing of model performance

4.4.1 Experimental setup

A classical test for assessing whether the models have truly learnt the relationships between the input features and the response variable is called a *permutation test* [42]. The null hypothesis assumes that features and labels are independent (i.e. that there is no relationship between the neuroimaging and non-imaging data and brain age), and the distribution corresponding to this hypothesis is estimated by randomly permuting the labels in the dataset. The p-value for this test is computed as

$$p = \frac{\sum_{i=1}^{k} \mathbf{1}\left[\mathcal{L}(\hat{\mathbf{y}}, \pi(\mathbf{y})) \leq \mathcal{L}(\hat{\mathbf{y}}, \mathbf{y})\right] + 1}{k + 1} \tag{4.3}$$

where $\mathcal{L}(\cdot, \cdot)$ is the error function used to train the model (in this project MSE), $\pi(\cdot)$ is a function that uniformly at random returns a permutation of the argument, and k is the number of samples.

In this project, $k = 1000$ samples will be taken for each model family and the results will be claimed significant if the p-value in Equation 4.3 falls below the significance level $\alpha = 0.05$.

4.4.2 Results

The null distributions of GCN and GAT model errors with permuted labels are shown in Figure 4.2, with the MSE for the original dataset indicated by the red line.

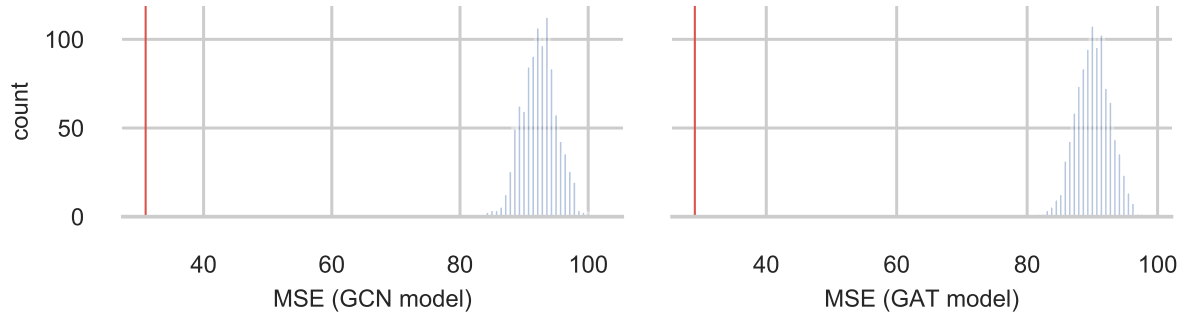


Figure 4.2: The null distribution of the label permutation test for GCN (left) and GAT (right) models. The red lines indicate MSE for the original dataset.

The p-values, representing how likely it is to get the MSE lower than the original dataset MSE purely by chance, were equal to $p = 0.001$ for both GNN models. This is lower than the chosen significance level $\alpha = 0.05$ so the null hypothesis is rejected and the performance of both models is statistically significant.

4.5 GNN robustness to population graph node feature noise

A desirable property for real-world machine learning models is their robustness, here defined as tolerance to the noise and inconsistency in input data. For population graphs trained on graph neural networks, this could be estimated by adding noise to an increasing proportion of nodes.

4.5.1 Experimental setup

For the feature noise robustness evaluation, an increasing proportion of population graph nodes is corrupted by randomly permuting their features. Then the model is retrained and tested on the hold-out test set, measuring the change in performance. To make sure that any effect on the evaluation metrics is due to added noise and not the changing dataset splits, the model is trained on a single dataset split while the noise is added to different subjects. Moreover, to ensure that the effect on test set performance is due to the interaction with neighbourhoods and not due to the individual node features, only the nodes in the training set will be corrupted. For each of the GCN and GAT models, the experiment is repeated five times for each noise level (1%, 5%, 10%, 20%, 30%, 50%, 80%, and 95% of training nodes).

4.5.2 Results

The results of node feature corruption on the predictive power of GNN models are shown in Figure 4.3.

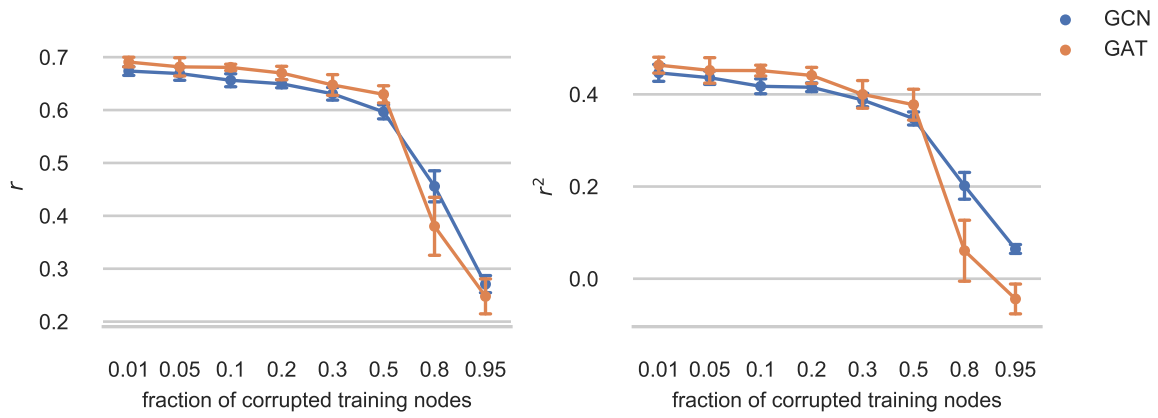


Figure 4.3: The effect of permuting node features on r (left) and r^2 (right) performance metrics, with error bars representing one standard deviation.

As expected, for both GCN and GAT models the performance was decreasing as more training nodes were corrupted, and then dropped drastically when more than half of the training nodes had their features permuted. In case of the GAT model, the r^2 metric fell below 0 (bottom of Figure 4.3), which means that no variance could be explained and that the mean of the observed subject ages in the dataset is a better estimate of the brain age than the predictions of the GAT model.

4.6 GNN dependence on population graph topology

4.6.1 Experimental setup

The assumption behind the population graph model is that the edge structure helps to control for confounding effects while giving additional information useful for brain age prediction. One experiment to test this is to remove an increasing proportion of edges from the population graph (here the proportions of removed edges being the same as in the previous section). The training procedure is then repeated five times at each percentage of removed edges using a different random seed. The more edges are removed, the less neighbourhood structure the graph neural network models can exploit, having to rely on individual node features.

4.6.2 Results

The effect of removing the edges on predictive power of the GNN models is shown in Figure 4.4.

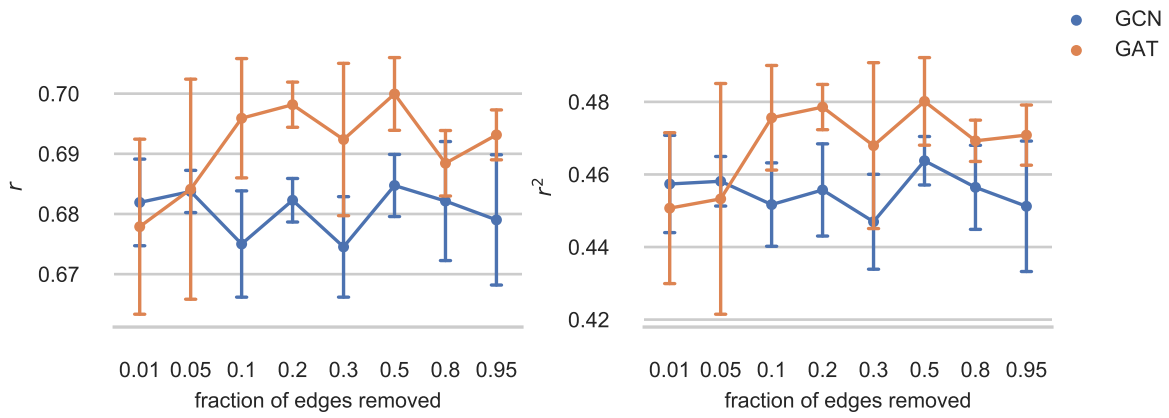


Figure 4.4: The effect of removing edges on r (left) and r^2 (right) performance metrics, with error bars representing one standard deviation.

Compared to the results in Section 4.5, where the predictive power drastically dropped with increased noise, the loss of information contained in edges and neighbourhoods of similar nodes did not affect the predictive power of the models. This is inferred from the standard deviation intervals being quite wide for both evaluation metrics, overlapping across most, if not all, edge loss levels.

4.7 Discussion

The node feature noise experiment shows that high levels of node corruption in the training set could drastically worsen the predictions for the uncorrupted test nodes. This result is expected as not only does the noise propagate to neighbourhoods affecting individual predictions, but there is also less useful training data available for the GNN architectures to learn from.

The edge removal experiment shows that the models rely more on the features of individual nodes rather than the graph structure defined by the similarity metrics. One explanation could be that the brain age depends more on the feature interactions within a single brain rather than the more universal signs of ageing; however, it seems more likely that the similarity metrics used (and the simple averaging technique to combine them) were not informative enough to allow for effective sharing of feature and label information. For example, the work by Parisot et al. [5], which used population graphs to achieve state-of-the-art results in brain disorder classification, shows that results can vary significantly based on the selection of similarity features alone, with up to 20% difference in mean accuracy scores.

In literature on brain age estimation, many alternative models perform better than the proposed GNN models, including an XGBoost model in Kaufmann et al. [1] with $r = 0.93$ (female) and $r = 0.94$ (male), a Gaussian process regression model in Cole et al. [38] with $r = 0.94, r^2 = 0.88$, and similar results in a variety of models using the BrainAGE technique, summarised in Franke & Gaser [2]. However, these approaches often eliminate important confounding (e.g. sex and brain health) effects by fitting separate models, use very small (i.e. a few hundred people) and consequently less diverse datasets, and filter out low-quality scans. While this improves the performance, it might affect the applicability of these models to real clinical settings, where data quality is less consistent. The approach used in this dissertation aims for a more general solution by training a single model on a large, diverse, and minimally preprocessed dataset while having considerable resource (memory) and modelling (similarity metric) constraints.

One advantage of the proposed population graph approach is its ability to model both imaging and non-imaging modalities for different sub-populations at once. This might be not as feasible or practical with alternative brain age prediction approaches (as it is harder to logically separate and to control the relative importance of the few non-imaging features among thousands of imaging features, if they are stored as a single vector), but might become more important with future advancements in neuroscience, growth of neuroimaging datasets, and growth in computational resources to support their processing. Addressing the GNN constraints outlined above, training on more data (while managing memory; UKB has more than doubled in size since the start of this project), incorporating additional (e.g. genetic [5]) modalities, and trying alternative state-of-the-art graph neural network architectures could give a much better picture of the potential of this approach.

5 Conclusion

5.1 Success criteria

The project has achieved and exceeded all of its success criteria and requirements (Section 2.7 and Project Proposal, Appendix B), representing the UKB dataset as a population graph, implementing the two GNN frameworks, and evaluating their results. A lot of progress on the proposed extensions (such as measuring the significance and robustness of the GNN models and increasing the flexibility of the preprocessing pipeline to more preprocessing options) has also been made.

5.2 The main contribution

The main success and contribution of this project was its step towards a preprocessing pipeline that could leverage several brain imaging and non-imaging modalities at once in a unified and consistent manner. This is important in neuroimaging research regardless of the downstream task or analysis method, as there is a widespread community effort to combat the reproducibility crisis in both neuroimaging [43] and machine learning[1] caused by, among other factors, the lack of transparency in preprocessing methods and software errors due to bad software engineering practices [44].

While the efforts to improve reproducibility of neuroimaging research are currently targeted at consistent processing of functional and structural MRI with libraries like *fMRIPrep* [45] and *Pypes* [46], this project, to the best of my knowledge, is one of the first to additionally incorporate non-imaging data modalities. While it was designed to prepare the data specifically for population graphs, due to its modular structure it has sufficient flexibility to be extended to more preprocessing options and modalities, and adapted to work independently of the downstream analysis method.

5.3 The main lessons

This interdisciplinary project has explored the brain age estimation problem in the context of graph neural networks. Both of the machine learning and neuroimaging fields are currently at the height of their ongoing research, with some of the literature this project relied on being only a few months old [1, 4, 40]. The continuous collaboration with experts from both disciplines was an enriching experience; more importantly, however,

[1] https://reproducibility-challenge.github.io/neurips2019/

this guidance was crucial to ensure that my work is not only a technically challenging, but also a clinically relevant contribution to the neuroscience community.

At the same time, this project taught me first-hand an important lesson in approaching predictive analysis problems in real-world settings. While the goal of this project was to learn more about the advanced, cutting-edge methods in computational neuroscience and to get hands-on experience with the tools for implementing them, the development of models *in practice* should consider the simple approaches first and only then apply the more sophisticated techniques. This was demonstrated by conceptually simpler methods in literature outperforming the more complex models proposed in this project. I am particularly proud of implementing what I called the robustness measurement framework – it showed that the graph neural network models did not rely on the similarity metrics as it was expected, naturally suggesting a direction for future work.

5.4 Directions for future work

To make this work more accessible to the wider neuroimaging community, the most important future direction would be to create an interface to the preprocessing pipeline that is agnostic to the downstream analysis task and make it publicly available online.

While it was not practical to do this extension for the UKB dataset, that uses its own organisation and preprocessing steps, the preprocessing pipeline could be adapted to work with raw magnetic resonance images, which would make it applicable to many more neuroimaging datasets and parcellations (that would now be a part of the preprocessing framework). This could be especially useful in a more general clinical setting where image acquisition is not standardised and the neuroimaging preprocessing expertise might not be available.

Some parts of the preprocessing component (e.g. fMRI preprocessing) could be extended to use other existing libraries for reproducible research, with this pipeline offered as an extension. In case there are no existing libraries, the proposed pipeline could offer additional methods in a consistent and easy-to-use manner. Support for additional modalities – such as genetic data that has already proved to be useful for neuroimaging tasks [38, 5] – could also be implemented.

Bibliography

[1] Tobias Kaufmann et al. "Common brain disorders are associated with heritable patterns of apparent aging of the brain". In: *Nature Neuroscience* 22.10 (2019), pp. 1617–1623. ISSN: 1546-1726. DOI: 10.1038/s41593-019-0471-7. URL: https://doi.org/10.1038/s41593-019-0471-7.

[2] Katja Franke and Christian Gaser. "Ten years of BrainAGE as a neuroimaging biomarker of brain aging: What insights have we gained?" In: *Frontiers in Neurology* 10 (2019), p. 789.

[3] Birkan Tunç et al. "Deviation from normative brain development is associated with symptom severity in autism spectrum disorder". In: *Molecular Autism* 10.1 (2019), p. 46.

[4] Xin Niu et al. "Improved prediction of brain age using multimodal neuroimaging data". In: *Human Brain Mapping* (2019).

[5] Sarah Parisot et al. "Disease prediction using graph convolutional networks: Application to Autism Spectrum Disorder and Alzheimer's disease". In: *Medical Image Analysis* 48 (2018), pp. 117–130. DOI: https://doi.org/10.1016/j.media.2018.06.001.

[6] Tong Tong et al. "Multi-modal classification of Alzheimer's disease using nonlinear graph fusion". In: *Pattern recognition* 63 (2017), pp. 171–181.

[7] Zhengxia Wang et al. "Multi-modal classification of neurodegenerative disease by progressive graph-based transductive learning". In: *Medical image analysis* 39 (2017), pp. 218–230.

[8] Thomas N. Kipf and Max Welling. "Semi-Supervised Classification with Graph Convolutional Networks". In: *International Conference on Learning Representations (ICLR)*. 2017.

[9] Petar Veličković et al. "Graph Attention Networks". In: *International Conference on Learning Representations* (2018). URL: https://openreview.net/forum?id=rJXMpikCZ.

[10] Sarah Parisot et al. "Spectral Graph Convolutions on Population Graphs for Disease Prediction". In: *MICCAI* (2017).

[11] Amber NV Ruigrok et al. "A meta-analysis of sex differences in human brain structure". In: *Neuroscience & Biobehavioral Reviews* 39 (2014), pp. 34–50.

[12] The Lancet Psychiatry. "Sex and gender in psychiatry". In: *Lancet Psychiatry* 3.11 (2016), p. 999.

[13] Juergen Dukart et al. "Age correction in dementia–matching to a healthy brain". In: *PloS one* 6.7 (2011).

[14] Cathie Sudlow et al. "UK Biobank: an open access resource for identifying the causes of a wide range of complex diseases of middle and old age". In: *PLOS Medicine* 12.3 (2015), e1001779.

[15] Matthew F. Glasser et al. "The minimal preprocessing pipelines for the Human Connectome Project". In: *NeuroImage* 80 (2013). Mapping the Connectome, pp. 105 –124. ISSN: 1053-8119. DOI: https://doi.org/10.1016/j.neuroimage.2013.04.127. URL: http://www.sciencedirect.com/science/article/pii/S1053811913005053.

[16] Bharat Biswal et al. "Functional connectivity in the motor cortex of resting human brain using echo-planar MRI". In: *Magnetic resonance in medicine* 34.4 (1995), pp. 537–541.

[17] Stephen M Smith et al. "Correspondence of the brain's functional architecture during activation and rest". In: *Proceedings of the National Academy of Sciences* 106.31 (2009), pp. 13040–13045.

[18] Adon FG Rosen et al. "Quantitative assessment of structural image quality". In: *Neuroimage* 169 (2018), pp. 407–418.

[19] Matthew F Glasser et al. "A multi-modal parcellation of human cerebral cortex". In: *Nature* 536.7615 (2016), p. 171.

[20] Jeremy R Gray, Christopher F Chabris and Todd S Braver. "Neural mechanisms of general fluid intelligence". In: *Nature neuroscience* 6.3 (2003), pp. 316–322.

[21] Jason Steffener et al. "Differences between chronological and brain age are related to education and self-reported physical activity". In: *Neurobiology of aging* 40 (2016), pp. 138–144.

[22] Carol Brayne et al. "Education, the brain and dementia: neuroprotection or compensation? EClipSE Collaborative Members". In: *Brain* 133.8 (2010), pp. 2210–2216.

[23] Cheryl L Grady and Fergus IM Craik. "Changes in memory processing with age". In: *Current opinion in neurobiology* 10.2 (2000), pp. 224–231.

[24] Matthias Kliegel and Theodor Jager. "Delayed–execute prospective memory performance: The effects of age and working memory". In: *Developmental neuropsychology* 30.3 (2006), pp. 819–843.

[25] David K Hammond, Pierre Vandergheynst and Rémi Gribonval. "Wavelets on graphs via spectral graph theory". In: *Applied and Computational Harmonic Analysis* 30.2 (2011), pp. 129–150.

[26] Michaël Defferrard, Xavier Bresson and Pierre Vandergheynst. "Convolutional neural networks on graphs with fast localized spectral filtering". In: *Advances in Neural Information Processing Systems*. 2016, pp. 3844–3852.

[27] Felix Wu et al. "Simplifying graph convolutional networks". In: *arXiv preprint arXiv:1902.07153* (2019).

[28] Zonghan Wu et al. "A comprehensive survey on graph neural networks". In: *arXiv preprint arXiv:1901.00596* (2019).

[29] Matthias Fey and Jan E. Lenssen. "Fast Graph Representation Learning with PyTorch Geometric". In: *ICLR Workshop on Representation Learning on Graphs and Manifolds*. 2019.

[30] Adam Paszke et al. "PyTorch: An Imperative Style, High-Performance Deep Learning Library". In: *Advances in Neural Information Processing Systems 32*. 2019, pp. 8024–8035. URL: http://papers.neurips.cc/paper/9015-pytorch-an-imperative-style-high-performance-deep-learning-library.pdf.

[31] Alexandre Abraham et al. “Machine learning for neuroimaging with scikit-learn”. In: *Frontiers in Neuroinformatics* 8 (2014), p. 14. ISSN: 1662-5196. DOI: 10.3389/fninf.2014.00014. URL: https://www.frontiersin.org/article/10.3389/fninf.2014.00014.

[32] Alexandre Gramfort et al. *Nilearn: Machine learning for Neuro-Imaging in Python*. Package available at https://nilearn.github.io. URL: https://nilearn.github.io.

[33] Fabian Pedregosa et al. “Scikit-learn: Machine learning in Python”. In: *Journal of machine learning research* 12.Oct (2011), pp. 2825–2830.

[34] Wes McKinney. “Data Structures for Statistical Computing in Python”. In: *Proceedings of the 9th Python in Science Conference*. Ed. by Stéfan van der Walt and Jarrod Millman. 2010, pp. 56 –61. DOI: 10.25080/Majora-92bf1922-00a.

[35] Stéfan van der Walt, S Chris Colbert and Gael Varoquaux. “The NumPy array: a structure for efficient numerical computation”. In: *Computing in Science & Engineering* 13.2 (2011), pp. 22–30.

[36] Lukas Biewald. *Experiment Tracking with Weights and Biases*. Software available from wandb.com. 2020. URL: https://www.wandb.com/.

[37] Jasper Snoek, Hugo Larochelle and Ryan P Adams. “Practical Bayesian optimization of machine learning algorithms”. In: *Advances in neural information processing systems*. 2012, pp. 2951–2959.

[38] James Cole et al. “Brain age predicts mortality”. In: *Molecular Psychiatry* 23.5 (2018), pp. 1385–1392.

[39] Sebastian Raschka. “Model evaluation, model selection, and algorithm selection in machine learning”. In: *arXiv preprint arXiv:1811.12808* (2018).

[40] Usama Pervaiz et al. “Optimising network modelling methods for fMRI”. In: *NeuroImage* 211 (2020), p. 116604. ISSN: 1053-8119. DOI: https://doi.org/10.1016/j.neuroimage.2020.116604. URL: http://www.sciencedirect.com/science/article/pii/S1053811920300914.

[41] Lutz Prechelt. “Automatic early stopping using cross validation: quantifying the criteria”. In: *Neural Networks* 11.4 (1998), pp. 761–767.

[42] Markus Ojala and Gemma C Garriga. “Permutation tests for studying classifier performance”. In: *Journal of Machine Learning Research* 11.Jun (2010), pp. 1833–1863.

[43] Krzysztof J Gorgolewski and Russell A Poldrack. “A practical guide for improving transparency and reproducibility in neuroimaging research”. In: *PLoS biology* 14.7 (2016).

[44] Russell A Poldrack et al. “Scanning the horizon: towards transparent and reproducible neuroimaging research”. In: *Nature reviews neuroscience* 18.2 (2017), p. 115.

[45] Oscar Esteban et al. “fMRIPrep: a robust preprocessing pipeline for functional MRI”. In: *Nature methods* 16.1 (2019), pp. 111–116.

[46] Alexandre M Savio et al. “Pypes: workflows for processing multimodal neuroimaging data”. In: *Frontiers in neuroinformatics* 11 (2017), p. 25.

A Hyperparameter search

This Appendix contains the hyperparameter search configuration and the hyperparameters for the best performing models as selected in Evaluation procedure (Chapter 4, Section 4.1).

A.1 Hyperparameter tuning configuration

Listing A.1: Hyperparameter search configuration for the GCN and GAT model families.

```yaml
method: bayes
metric:
  goal: minimize
  name: cv_validation_average_mse
parameters:
  dropout:
    distribution: uniform
    max: 0.5
    min: 0
  epochs:
    value: 10000
  layer_sizes:
    distribution: categorical
    values:
    - [1024, 512, 512, 256, 256, 1]
    - [2048, 1024, 512, 256, 128, 1]
    - [1024, 512, 512, 512, 256, 256, 1]
    - [1024, 512, 512, 256, 256, 128, 128, 1]
    - [512, 512, 512, 256, 128, 1]
    - [1024, 512, 256, 128, 128, 1]
  learning_rate:
    distribution: log_uniform
    max: -2.995
    min: -9.904
  n_conv_layers:
    distribution: int_uniform
    max: 5
    min: 1
```

```
  similarity:
    distribution: categorical
    values:
    - (['SEX', 'ICD10', 'FI', 'FTE', 'MEM'], 0.8)
    - (['SEX', 'ICD10', 'FI', 'FTE', 'MEM'], 0.9)
    - (['SEX', 'FTE', 'FI', 'MEM'], 0.8)
    - (['SEX', 'ICD10', 'MEM', 'FTE'], 0.8)
    - (['SEX', 'ICD10', 'MEM', 'FI'], 0.8)
  weight_decay:
    distribution: log_uniform
    max: -2.995
    min: -9.904
```

A.2 Hyperparameters of shortlisted models

Tables A.3 and A.4 use encodings given in Tables A.1 and A.2 for similarity feature sets and layer sizes respectively.

Table A.1: Similarity feature set encoding.

Encoding	FI	FTE	ICD10	MEM	SEX
SF1	Yes	Yes	Yes	Yes	Yes
SF2	Yes	No	Yes	Yes	Yes
SF3	No	Yes	Yes	Yes	Yes
SF4	Yes	Yes	No	Yes	Yes

Table A.2: Layer size encoding.

Encoding	**Layer sizes**
LS1	[1024, 512, 512, 256, 256, 1]
LS2	[1024, 512, 512, 512, 256, 256, 1]
LS3	[1024, 512, 256, 128, 128, 1]
LS4	[2048, 1024, 512, 256, 128, 1]
LS5	[512, 512, 512, 256, 128, 1]

Table A.3: Shortlisted population graph and GCN model parameter combinations during the model selection process.

Hyperparameter	**GCN1**	**GCN2**	**GCN3**	**GCN4**	**GCN5**	**GCN6**	**GCN9**
Similarity feature set	SF1	SF3	SF3	SF2	SF2	SF2	SF2
Similarity threshold	0.9	0.8	0.8	0.8	0.8	0.8	0.8
Layer sizes	LS1	LS2	LS3	LS5	LS3	LS3	LS4
# convolutional layers	5	3	1	2	5	3	4
Dropout	0.321941	0.042080	0.048596	0.237940	0.375442	0.386998	0.426491
Learning rate	0.006984	0.006187	0.005095	0.004731	0.015796	0.010273	0.003504
Weight decay	0.013118	0.002084	0.016171	0.002517	0.003114	0.005341	0.018943

Table A.4: Shortlisted population graph and GAT model parameter combinations during the model selection process.

Hyperparameter	**GAT1**	**GAT2**	**GAT3**	**GAT4**	**GAT5**	**GAT6**	**GAT7**	**GAT8**	**GAT9**
Similarity feature set	SF2	SF1	SF2	SF1	SF1	SF1	SF1	SF2	SF2
Similarity threshold	0.8	0.9	0.8	0.9	0.9	0.9	0.9	0.8	0.8
Layer sizes	LS4	LS3	LS3	LS5	LS3	LS3	LS3	LS5	LS5
# convolutional layers	2	2	3	2	3	2	3	3	3
Dropout	0.003142	0.306806	0.104624	0.327091	0.407471	0.323481	0.291117	0.455777	0.381829
Learning rate	0.013365	0.001679	0.003412	0.002482	0.003246	0.001462	0.006769	0.006813	0.003820
Weight decay	0.000605	0.002071	0.036676	0.001549	0.006715	0.002475	0.000844	0.001483	0.003226

B Project proposal

Computer Science Tripos – Part II – Project Proposal

Graph neural networks for age prediction from neuroimaging data

2419E

Project Originator: Tiago Azevedo

Project Supervisors: Tiago Azevedo, Alexander Campbell, Prof Pietro Liò

Project Overseers: Prof Jon Crowcroft, Dr Thomas Sauerwald

Introduction

A graph neural network (GNN) is a type of a neural network that operates on graph inputs and is used for tasks like node classification, link prediction and clustering (geometric deep learning). GNNs have recently become popular and proved successful in a broad range of real-world applications, such as text and image classification, knowledge graphs, and interaction modelling in physical and biological systems. [1]

One domain where graphs offer a natural representation is social networks and *populations*, with nodes representing individuals (their features and labels), and edges corresponding to associations between individuals according to some heuristic or a formally defined similarity metric. The reason why such graph representation is considered to be useful in the geometric deep learning context is that the network can make use of both the individual features (node feature vectors) and the overall trends in the population through pairwise similarities (graph edges), [2] inferring the label of an individual node both from the node itself and from its neighbourhood.

Project description

This project was inspired by Parisot et al.'s [2, 3] state-of-the-art application of a type of a GNN called Graph Convolutional Network (GCN) to the population graphs of healthy controls and patients with neurological or neurodegenerative disorders. In these papers, the GCN (adapted from Kipf and Welling [4]) was used in a semi-supervised manner for

two classification tasks: 1) prediction of autism spectrum disorder (ASD) from the ABIDE dataset and 2) prediction of a progressive form of Mild Cognitive Impairment (MCI) that develops into Alzheimer's disease (AD) from the ADNI dataset.

Moreover, a recent paper by Kaufmann et al. [5] has linked the incidence of common brain disorders, including ASD, MCI, and AD as well as others, to the deviation between chronological and biological brain ageing. These results suggest that being able to estimate the subject's age from the neuroimaging data may be important in understanding the mechanisms of those disorders and helping physicians to find more effective treatments.

The aim of this project will therefore be to adapt the population graph approach of Parisot et al. [2, 3] to a regression task on the UK Biobank dataset, predicting the subject's age based on neuroimaging data, and comparing it to another successful geometric deep learning architecture such as the Graph Attention Network. [6] The performance of the networks will be evaluated on the standard metrics, e.g. the coefficient of determination r^2.

Starting point

The source code for the implementation of Kipf and Welling's [4] GCNs and Parisot et al.'s [2, 3] first classification task is publicly available online.[1,2]

I will be using PyTorch for this project because of its support for machine learning on structured graph data. In particular, PyTorch Geometric (PyG)[3] – a geometric deep learning extension library – will make the implementation, iteration and extensions to the model more flexible in addition to performance improvements and simplified APIs.

I have experience with the basics of TensorFlow[4,5] and no experience with PyTorch or graph neural networks. I have attended or will study (possibly in advance) the CST courses related to the subject of this project, such as IA Machine Learning and Real-World Data, IB Artificial Intelligence, II Data Science, II Bioinformatics, and II Machine Learning and Bayesian Inference.

Dataset

I will be using the data from the UK Biobank, kindly preprocessed and provided by Dr Richard Bethlehem of the Department of Psychiatry.

The UK Biobank is a large dataset containing comprehensive phenotypic, genetic, MRI and other data from the total of over 500,000 participants.[6] In this project, I will be using the subset of this dataset with only those subjects who had the neuroimaging data collected and preprocessed (approximately 20,000 participants). This includes both

[1] https://github.com/tkipf/gcn
[2] https://github.com/parisots/population-gcn
[3] https://github.com/rusty1s/pytorch_geometric
[4] Five-course Deep Learning specialisation by deeplearning.ai on Coursera
[5] Google's Machine Learning Crash Course and follow-up courses.
[6] https://www.ukbiobank.ac.uk/participants/

structural (T1, T2 FLAIR) and functional (resting state fMRI) data, preprocessed with the standard UK Biobank pipelines[7] and additionally denoised and parcellated (in several common parcellations) by Dr Bethlehem.

Work to be done

The following are lists of explicit deliverables to be implemented.

Graph neural network framework

G1. The data is preprocessed into features and is ready for analysis.

G2. Definition of the similarity metric to be used in connecting the graph. The graph is connected based on that similarity metric to be processed by graph neural networks.

G3. Implementation of Kipf's GCN [4] for the age regression task.

G4. Implementation of the Graph Attention Network for comparing its performance to the Graph Convolutional Network.

Evaluation framework

E1. Comparison of the alternative graph neural network models using the coefficient of determination r^2.

Success criteria

The project will be successful if the following items will have been implemented.

SC1. Representation of the UK Biobank data as a population graph with nodes representing the individuals and edges representing associations between them based on pairwise similarity.

SC2. The Graph Convolutional Network for age regression on the population graph.

SC3. The Graph Attention Network for the same task.

SC4. The evaluation framework for comparing the performance of the two graph neural networks.

Evaluation of the project

The performance of the graph neural networks will be measured across several metrics. The main metric to evaluate a regression task, in contrast the classification in Parisot et al. [3], is the coefficient of determination r^2.

[7] https://biobank.ctsu.ox.ac.uk/crystal/crystal/docs/brain_mri.pdf

Possible extensions

PE1. An additional metric that could be used to evaluate the performance of the networks is *robustness* to missing or noisy data. Robustness, which could be defined as *the rate at which the predictive power drops as more information is removed from the nodes*, would reveal how important is the neighbourhood (edge) information for accurate predictions compared to the node features only.

PE2. Implement spectral filter computation with *Cayley polynomials* instead of using Chebyshev polynomials. Cayley polynomials have been introduced in a paper by Levie et al. [7] and were mentioned in Parisot et al. [3] as a possible improvement.

PE3. The main implementation of the graph neural network relies on manually handcrafted features from preprocessed brain imaging data. Time permitting, an extension could be to create a package that can be used after any standard neuroimaging preprocessing pipeline (e.g. with results in BIDS[8] format) to extract these features, and possibly improve upon as well as create new ones. This would make execution of the model more efficient, robust and generalisable.

PE4. Implement weighted edges in the Graph Convolutional Network and Graph Attention Network, as the main implementations will have binary edges.

Timetable and milestones

Michaelmas weeks 0–1

- Work on project proposal.

Milestones. Submit Phase 1 report by 14/10/2019. Submit draft proposal by 18/10/2019.

Michaelmas weeks 2–4

- Get access to the UK Biobank data and get familiar with its features.
- Define a possible graph similarity metric.

Milestones. Submit final project proposal by 25/10/2019.

Michaelmas weeks 5–6

- Write code for connecting the nodes based on a similarity metric.
- Connect the nodes (with their features) into a graph.
- Start working on the implementation of the Graph Convolutional Network (e.g. define loss and random label removal for semi-supervised training, start implementing the layers).

Michaelmas weeks 7–8

[8] https://bids.neuroimaging.io

- Work on the implementation of layers for the Graph Convolutional Network. Compute the general performance metrics.
- Start working on Graph Attention Network implementation for the same task.

Michaelmas vacation

- Continue working on and finish the neural network implementations, compute performance metrics.
- Work on graph neural network evaluation: implement the robustness measurement framework.
- Measure the robustness of the neural networks.
- Start writing the dissertation and the project progress report.

Milestones. Complete the implementation of the main part of the project.

Lent weeks 0–2

- Finish the progress report, prepare for the presentation.
- Implement Cayley polynomials.
- Start working on the data preprocessing pipeline.

Milestones. Submit progress report by 31/01/2020.

Lent weeks 3–5

- Continue implementing the data preprocessing pipeline.
- Start working on the implementation of weighted edges.

Lent weeks 6–8

- Finish implementing the data preprocessing pipeline.
- Finish implementing the weighted edges.
- Continue working on the dissertation write-up.

Easter vacation

- Complete the dissertation draft and send it for review.
- Edit the draft based on the feedback received.

Milestones. Send out the complete draft for review by 27/03/2020. Submit dissertation early by 20/04/2020.

Easter weeks 0–2

Time reserved for any unexpected issues.

Resource declaration

For this project I will be using my personal MacBook Pro (2019, with 1.4 GHz Quad-Core Intel Core i5 processor and 8GB of RAM). Training the model will require the use of GPUs provided by the Computational Biology Group (as confirmed by Prof Pietro Liò). To prevent any loss of data, both the source code and the LaTeX source will be stored on my machine, private GitHub repositories, and Google Drive, as well as regularly backed up on an external HDD.

References

[1] Jie Zhou et al. "Graph Neural Networks: A Review of Methods and Applications". In: *arXiv preprint arXiv:1812.08434* (2018). arXiv: 1812.08434. URL: http://arxiv.org/abs/1812.08434.

[2] Sarah Parisot et al. "Spectral Graph Convolutions on Population Graphs for Disease Prediction". In: *MICCAI* (2017).

[3] Sarah Parisot et al. "Disease prediction using graph convolutional networks: Application to Autism Spectrum Disorder and Alzheimer's disease". In: *Medical Image Analysis* 48 (2018), pp. 117–130. DOI: https://doi.org/10.1016/j.media.2018.06.001.

[4] Thomas N. Kipf and Max Welling. "Semi-Supervised Classification with Graph Convolutional Networks". In: *International Conference on Learning Representations (ICLR)*. 2017.

[5] Tobias Kaufmann et al. "Common brain disorders are associated with heritable patterns of apparent aging of the brain". In: *Nature Neuroscience* 22.10 (2019), pp. 1617–1623. ISSN: 1546-1726. DOI: 10.1038/s41593-019-0471-7. URL: https://doi.org/10.1038/s41593-019-0471-7.

[6] Petar Veličković et al. "Graph Attention Networks". In: *International Conference on Learning Representations* (2018). URL: https://openreview.net/forum?id=rJXMpikCZ.

[7] Ron Levie et al. "CayleyNets: Graph Convolutional Neural Networks with Complex Rational Spectral Filters". In: *arXiv preprint arXiv:1705.07664* (2017). arXiv: 1705.07664. URL: http://arxiv.org/abs/1705.07664.