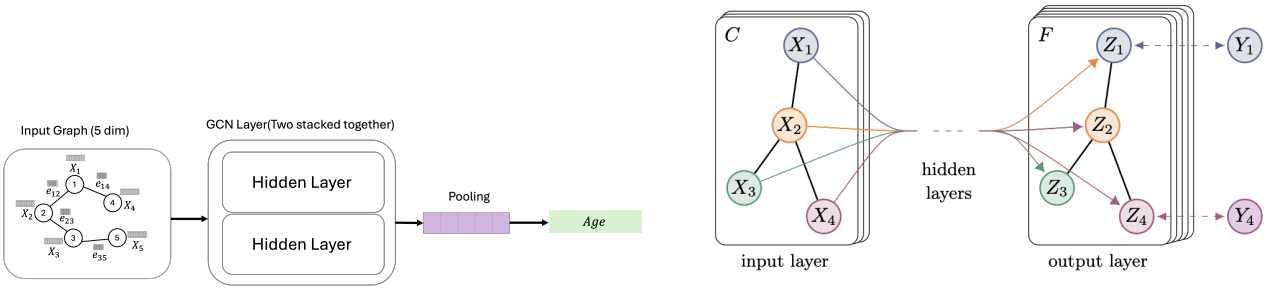
Deep Learning Neural Network

In this final project, our team tried several deep learning techniques, including graphical convolutional neural network(GCN), graphical attention neural network (GAT), multilayer perceptron (MLP) and autoencoder. We will introduce each of those algorithms in the following sections.

How to Prepare Data for a Graphical Neural Network

Since GNN cares more about the graphical structure of the correlation matrix, we excluded metadata when feeding data into the neural network. Initially, the raw correlation vector, representing the upper triangle of the connectivity matrix (19900 connections between 200 brain regions), undergoes preprocessing: values are clipped to prevent infinities, Fisher z-transformed to stabilize variance, and then standardized (mean-centered and scaled to unit variance). Following this, the full symmetric 200×200 adjacency matrix is reconstructed from the processed vector. The edges of the graph are determined by thresholding this matrix; Only connections with an absolute standardized correlation that exceeds a predefined threshold are retained. The indices of these connections form the edge index, and their corresponding standardized correlation values become the edge weights. Self-loops are also added to each node. Concurrently, node-level features are computed for each brain region based on its connectivity profile within the thresholded graph: these include total connection strength (sum of absolute weights), degree (number of connections), positive strength (sum of positive weights), and negative strength (sum of absolute negative weights). These features, along with a bias term, are stacked into a node feature matrix x. Finally, the function encapsulates the node features (x), the graph structure (edge index, edge weights), and the participant's age (y) into a PyTorch Geometric Data object, creating a single graph representation for that individual suitable for GNN analysis.

Graphical Convolutional Neural Network



(a) GCN Code Graphical Illustration (b) GCN Structure Overview

Figure 1: Simple Illustration of How GCN Works

This network takes a whole graph (say, one subject's brain-connectivity graph) and predicts a single value (age) in four steps. First, each node starts with five raw features. Second, the two stacked GCNConv layers let every node mix information with its neighbours twice, so local patterns propagate a few hops away and are distilled into a 64-dimensional "hidden" signature per node. Third, a global-mean pool simply averages all these node signatures to create one compact graph-level fingerprint that captures the subject's overall connectivity profile. Finally, a single fully-connected (Linear) layer reads that fingerprint and outputs one number; squeezing it drops the extra dimension so we get a clean scalar age prediction. In short, the network learns neighbour-aware node representations, condenses them into a graph summary, and maps that to the target value.

Graph Attention Network

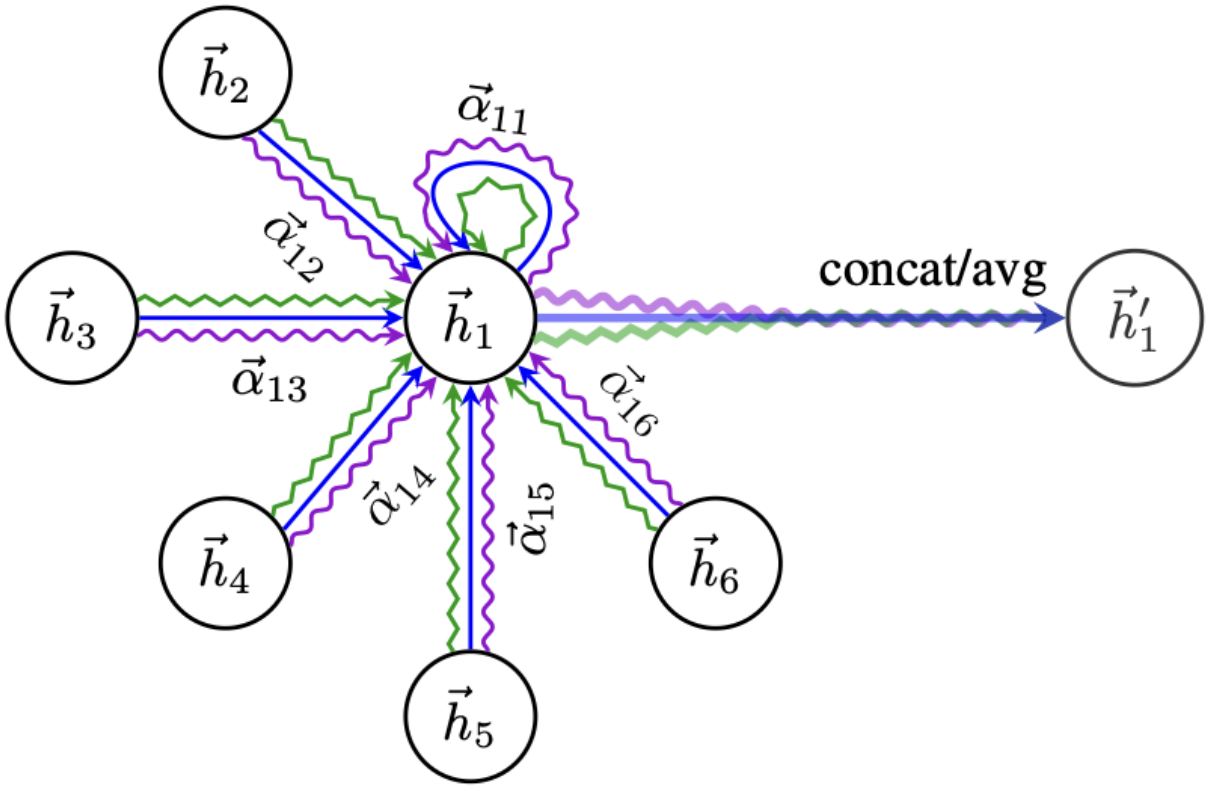


Figure 2: GAT Graphical Illustration

Our graph attention network (GAT) predicts the age of each subject directly from their graph-structured data in four intuitive stages. First, every node enters the model with a five-dimensional feature vector that captures its raw attributes. Second, the graph passes through two successive attention convolution layers. In the first layer four independent "attention heads" learn to weigh each neighbour's contribution differently and then concatenate their findings, giving every node a rich, multi-perspective 32-dimensional representation. The second layer distils those concatenated signals through one more head, producing a streamlined 32-dimensional embedding per node. Third, we use a global-mean pool to compress all node embeddings within a graph into a single graph-level fingerprint that summarises the subject's overall connectivity pattern. Finally, a simple linear re-

gression head maps that fingerprint to one scalar—the predicted age. In essence, as shown in figure 2, each neighbour is presented to h_1 in three different 'perspectives,' and h_1 learns how much to listen to each perspective before blending them into its new embedding. After the new embedding is created, they were aggregated to make prediction.

How to Prepare Data for Autoencoder

Since the original dataset got 19900 columns for each participant, we convert those 19900 correlation values back to the 200×200 matrix, and now the training dataset should become a $1104 \times 200 \times 200$ matrix. After that, we can fit this matrix into autoencoder to reduce the dimension of the dataset.

Autoencoder

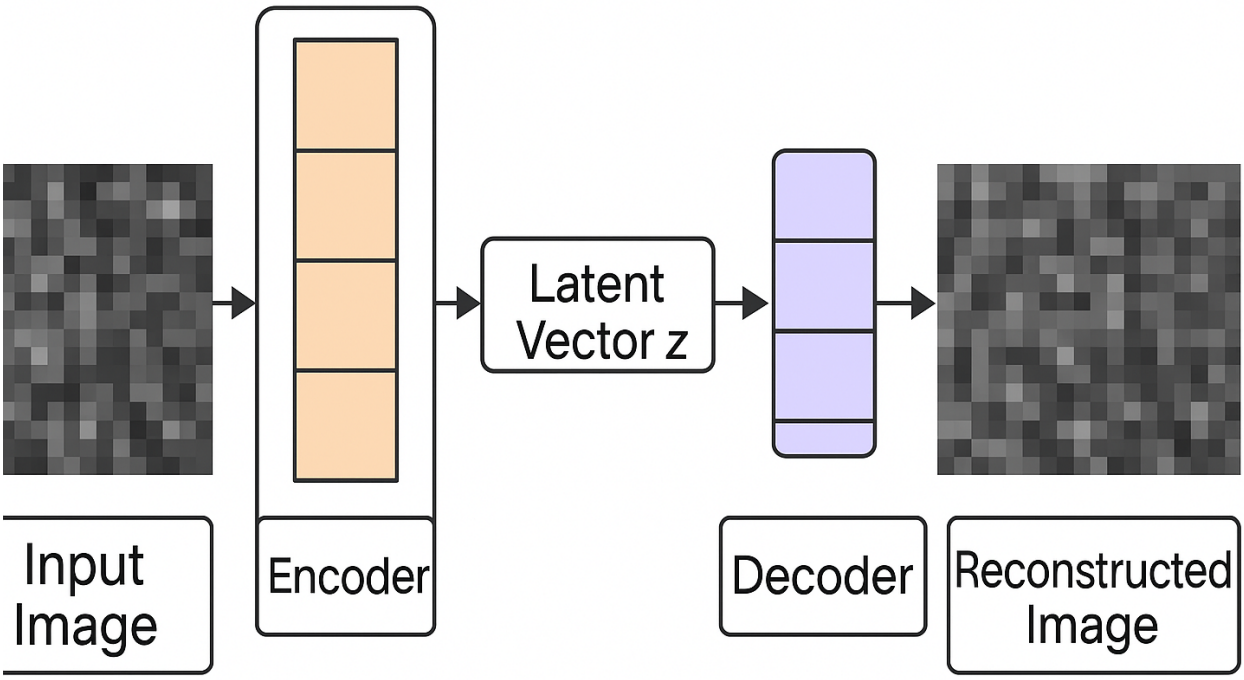


Figure 3: Autoencoder Graphical Illustration

Encoder — squeezing the picture

In our case, single-channel input image (e.g., a 200×200 correlation map) passes through four convolution-plus-ReLU blocks, each halving the spatial size and expanding the channel depth. After the last conv, the data will be converted to a stack of 256 tiny feature maps, 13×13 each. These maps are flattened and fed into a fully connected layer that distils everything down to a 64-number latent vector z (the network's compact snapshot of the image).

Decoder — rebuilding from the snapshot

Another fully connected layer inflates the 64 numbers back to $256 \times 13 \times 13$, reshaping them into a feature stack. Four layers of transposed convolution (a.k.a. deconvolution) then up-sample step by step, reversing the encoder reductions until we recover an image of the original size. The final layer outputs one channel, giving a reconstruction that should look as close as possible to the input. During training, the model tweaks its weights so the reconstructed image minimises a loss (typically mean-squared error) against the original. When it succeeds, the latent vector z becomes a tidy, information-dense representation that can plug into downstream models like Lasso, XGBoost or MLP, while the decoder proves that z really does capture the essentials. For this project, we tried two different autoencoder approaches for the metadata:

1. The first way is to add them as extra dimension when using autoencoder (making it become conditional), then the autoencoder will learn the meta information when reducing the dimension

2. The second way is to use autoencoder only to reduce the dimension of correlation matrix, then after getting the latent vector z, add the extra meta information onto z

MLP

Based on the reduced output of the autoencoder, we can put the latent vector z in simple MLP. In our code, multilayer perceptron (MLP) predicts age from a

flat feature vector in three quick stages. First, the input passes through a fully connected layer that expands it to 256 hidden units; batch-normalisation steadies the activations and a ReLU adds non-linearity so the network can capture complex patterns. A light 20% dropout then randomly zeros some units, which helps the model generalise instead of memorising the training set. The second hidden layer repeats the process—another 256-unit linear transform followed by ReLU and dropout—allowing the network to refine higher-level feature interactions. Finally, a single-neuron output layer maps the refined 256-dimensional signal to one number, and function *squeeze* removes the surplus dimension so we return a clean scalar age prediction. In short: two stacked $Linear \rightarrow BatchNorm \rightarrow ReLU \rightarrow Dropout$ blocks learn rich, noise-robust representations, and the final linear head translates that representation into the target age.

Results Summary

Table 1: Comparison of Different Deep Learning Model Performance on Test Data

Model Name	Testing R^2	Testing MSE
GCN	-0.003	10.665
Autoencoder + Lasso	**0.540**	**4.611**
Autoencoder + ElasticNet	0.539	4.625
Conditional Autoencoder + Lasso	0.509	4.920
Conditional Autoencoder + ElasticNet	0.514	4.625
MLP with Hypertuning	0.421	6.049

Based on the result in table 1, fitting Lasso on the reduced dimension using Autoencoder got the best result, which is consistent with our findings in previous machine learning section where Lasso regression on PCA reduced dataset gives the best results.