Detecting Alzheimer on Morphological Connectivity Strength with Two New Graph Attention Methods: GrlS-Gat and TGCSA

Ahmet Zafer SAĞLIK[1,1] and Gökhan KIYĞA[1,2]

[1] Faculty of Computer and Informatics, Istanbul Technical University, Istanbul, Turkey

Abstract. Nowadays, graph theory has a wide variety of usage domains such as world-wide-web, neuroscience, and social networks. Graph mining is based on insights of graph data. Link prediction, graph classification and community detection are made via graph mining in order to maximize influence on target people. On the other hand, real-world data can be a challenging issue to represent with the graph due to the complexity and noise of its nature [1]. Graph neural network has given better performance as an important graph representation method based on machine learning and has drawn significant interest in study. However while calculating such features for graph classification, most of the approaches like GNN deal with the entire graph. Recently the attention which is inspiration of brain functions and based on the relevant parts of the data have been implemented to address this issue, is the most thrilling developments in machine learning, whose huge potential was well shown in invariable fields. In this study, we propose Graph-Level Similarity-Based GAT (GrlS-GAT) and Transformed Graph Classification using Structural Attention (TGCSA) to detect the illness of the brain. GrlS-GAT is a novel architecture that strengthen the raw GAT. GrlS-GAT combines an ensemble learning technique Extra-Tree Classifier (use to learn importance of the feature to outcome), a graph attention network and graph convolution network. According to Veličković et al. GATs is a new implementation of the neural networks operating on graph-structured data, utilizing concealed self-attention layers to resolve the deficient prior methods based on approximating graph convolutions [2]. By loading layers in which nodes (entities) are able to participate over the features of their neighborhoods, they allow assigning different sizes to various nodes in the neighborhood, without needing any kind of expensive matrix process or understanding the layout of the graph beforehand. Throughout this way, they tackle many main problems of spectral-based neural graph networks collectively and render our concept conveniently accessible to both inductive and transductive challenging issues. Moreover, the use of attention mechanism allows us to collect information and give our attention on small parts of the graph. With the help of this approach, we can avoid noise in the rest of the graph.We present a combination of GVT (Graph View Transformer) and RNN model, called the Graph Classification using Structural Attention (TGCSA), that processes only a part of the graph by selecting a sequence of important nodes with an attention mechanism. And since we use GVT on the data set, we combine several graph views into one and create better results. The attention mechanism can be used to solve the real-word data challenging issues on the graph representation. With this study, we expect our combination model can detect brain illness at the high accuracy.

1 Introduction

Nowadays, graph theory has wide variety of usage domains such as world-wide web, neuroscience and social networks. Data taken from chemoinformatics [3], social network analysis [4], urban-computing [5] and cyber-security [6] can easily represent with labeled graphs. Graph mining is based on insights of graph data. Link prediction, graph classification and community detection are made via graph mining in order to maximize influence on target people. On the other hand, real world data can be challenging issue to represent with the graph due to complexity and noisy of its nature. Attention mechanism which is inspiration of brain functions and based on the relevant parts of the data have been proposed to address this issue.

CNNs have been useful in answering problems including computer vision, structural optimization or translation software. The data depiction inherent this design seems to have a grid-like mechanism. Such algorithms can quickly regenerate localized filtering by applying it to all source locations of learning parameters. Nevertheless, several fascinating activities include details that cannot be described in a grid-like layout and instead reside in an irregular environment. we are observing this result on 3D grid, social networks, computer networks, biological networks or neural links. Such data is mostly representing on graphs.

Throughout research, numerous attempts were made to apply neural networks to randomly ordered graphs. Originally recurrent neural networks were used for processing data represented by acyclic graphs in graphs. GNNs consist of an iterative process that transmits the node states to a balance point, followed by a network that produces a response, depending on the location, for each node. But the generalization of the transformations to the graphic realm is gradually drawn. Improvement is often referred to as spectral and non-spectral approaches in this area. On the one side, spectral methods work with a spectral depiction of graphs. The description of the node was effectively utilized. Such problems have been dealt with in subsequent work. Henaff et al. implemented a spectral filter parameterization of smooth coefficients to find the filters spatially [7]. Previously, Defferrard et al. proposed calculating filters by means of an Laplacian line extension, eliminating the need for Laplacian own vectors to be estimated and space-dispensable filter generations [8]. Kipf and Welling eventually streamlined the previous method by allowing filters to operate around each node in a 1-stage neighborhood [9].

Nevertheless, the filters analyzed are centered on Laplacian's own framework, depending on the graph form, in every of the spectral methods alluded to above. A model educated in a particular structure cannot therefore be extended to a diagram with a different structure directly. On the other side, we have non-spectral approaches to explain convolutions that work directly at clusters of neighborhoods. One of the complexities of these methods is to identify an operator that deals for different sizes of neighborhoods and preserves the weight distribution properties of CNNs.

In some situations, it requires a particular weight matrix for each node point to analyze weight by using the power of the transformation matrix to define the population for each entry and neighborhood step or removing and standardizing quarters with something like a fixed number of nodes. Attention systems have almost become a de facto standard for many sequence-based activities. One of the advantages of care systems is

that specific inputs can be used to make decisions based on the most relevant part of the results. When a form of focus is used to calculate the speech of a particular set, it is usually referred to it as self-attention. Self-attention has worked effectively in behavior such as machine literacy, and word representation, in combination with recurrent neural networks (RNNs). Inspired by these latest works, we create an emphasis architecture to characterize network-structured data at node levels.

The goal is to calculate any node in the graph's hidden representations, discuss its neighbors and follow a self-interest approach. The focus design has several fascinating characteristics: process is efficient since it can be paralleled through neighbor nodes, by setting arbitrary weights for neighbors, to graph nodes at different degrees, and the concept is relevant directly to inductive learning problems like tasks where the model must be extended to completely unknown graphs. Many similar methods include local linear embedding (LLE) and memory networks LLE selects a specific number of neighbors over each data set and trains to reconstruct each point as the average sum of its neighbors. The second optimization stage removes the embedding function of the level. Memory networks often share such relationships with our research because we find the position of a node to be a memory, that is used to calculate node features by looking through their values, and then updated to the same location by saving the new features.

1.1 Transformed Graph Classification using Structural Attention (TGCSA)

Graph Classification is a highly important problem which have numerous using field in computer science. Our new method TGCSA combining two different approaches and making a mixed contribution to problem of Graph Classification.

In Transformed Graph Classification using Structural Attention (TGCSA), we use two approaches in order to get more accurate and optimum solutions. First, we use Graph View Transformer (GVT) which led our data set to contribute more efficiently. Secondly, we use Graph Classification by Reinforcement Learning which gives nice results on the data that created by GVT.

In GVT our main goal is combining several graph views which are adjacency matrices, into one meaningful matrix. Classification part is using this output as input with labels. In this part of our approach is the reducing cost and finding more accurate results. We are using unsupervised learning and meta-path [10] creation to reach our goal.

In Graph Classifications part we use Recurrent neural network (RNN). Since we are working with graphs making the complicated methods will detract us from our goal. So we have bring a nice perspective for the problem which is Partially Observable Markov Decision Process (POMDP) [11]. In this approach the agent which we use with Reinforcement Learning only travel a part of graph. When it is ready it makes prediction. This help again decrease our cost and with the help of that we can work with more complicate graphs. In Graph Classification we used Reinforcement Learning [12] and Supervised Learning together.

1.2 Graph-Level Similarity-Based GAT (GrlS-GAT)

Graph-Level Similarity-Based GAT (GrlS-GAT) is the novel learning architecture which strengthen the raw Graph Attention Network provided by Veličković et al in 2017 [2]. GrlS-GAT aims to work on graph-level effectively. GrlS-GAT includes two networks that works together sequential manner. First network is the attention networks which is feed forward neural networks that includes one layer and the second network is a Graph Convolution Network which predicts class at last layer. Graph Convolution Network performs forward and backward propagation. As it mentioned before two networks work sequential manner; the attention network computes high level feature vectors for each graph and send them as input to Graph Convolution Network. The term high level feature means aggregated feature from the instance's neighbours, GrlS-GAT cares not only it sole feature but also the neighbours' features. We use dropout process in the training stage to solve the model over-fitting.

GrlS-GAT uses Extremely Random Tree Classifier (Extra-Tree Classifier) which is an ensemble learning technique to learn the importance of the features to the outcome. We update Euclidean distance according to the importance of the features, because the Euclidean distance can be illusive when it ignores the importance of the feature to the outcome. We consider to construct similarity on Graph-level, so each subject is a graph represents connectivity of 35 points in two brain region. GrlS-GAT learns importance of each connection between these points, so it interprets the similarity of the subjects in disease space. It means that GrlS-GAT looks distances to class from the subjects. We construct a mid-model that use zero weight matrix $\mathbf{W}$ as initial point in the attention layer. This model strengthen the raw GAT in many basis but it gives less accurate in some dataset also. Therefore we derived the weight matrix $\mathbf{W}$ from the features matrix to build more stable and accurate model. In the construction stage we deal with cardinality differences between layer and features matrix by multiplication derived weight matrix $\mathbf{W}$ with two matrix of ones which construct according to layer size on both left and right side of derived weight matrix. At the end we reach a model that combined an ensemble learning technique, attention layer and graph convolution network. GrlS-GAT increase accuracy of the raw GAT, we are going to evaluate this result in the result and discussion part.

2 Proposed Method

2.1 Transformed Graph Classification using Structural Attention (TGCSA)

The goal of our framework, Transformed Graph Classification using Structural Attention (TGCSA), is to generate new graph structures and do graph classification with an agent collect enough information that will allow it to make a correct prediction on the label of converted the graph.

TGCSA seek for new graph structures using multiple candidate adjacency matrices to perform more effective graph convolutions and learn more powerful node representations.

We formulate the problem of using attention on graphs as a decision process of a goal-directed agent traversing along an input attributed graph. The agent starts at a random node on the graph and,at each time step, moves to a neighboring node. The agent

only has the information it currently visit. Since we don't have access to all information of graph consist, the agent needs to combine information over time to help it determine the parts of the graph to explore further. The input for our framework is multiple graph views as connectivity matrices.

Graph View Transformer (GVT). The inputs for our framework are multiple graph views in the format of connectivity matrix. Our main goal in here is combining graph views. In first part of TGCSA, we learn meta-paths [13,14] for given data and apply graph convolution on the learned meta-path graphs. With the help of that we are learning useful meta-paths and create one optimum graph view using multiple meta-paths by unsupervised learning. The meta-path generation which is our Model's (TGCSA's) first part, is Graph View Transformer (GVT) Layer. GVT has two components. First, GVT Layer chooses two graphs Q_1 and Q_2 from candidate adjacency matrices A. Then, it learns a new graph structure by the matrix multiplication of this two graphs, Q_1Q_2. It calculates the combination of adjacency matrices like $\sum_{t_l \in T^e} \alpha_{t_l}^{(l)} A_{t_l}$ in (2) by 1x1 convolution, by using the weights that we calculated in softmax function as $Q = F(A; W\phi) = \theta(A; softmax(W_\phi))$,(1) where ϕ is the convolution layer and $W_\phi \in R^{11K}$ is the parameter of ϕ. With using two adjacency matrices Q_1 and Q_2, we are calculating the meta-path adjacency matrix with matrix multiplication, $Q1Q2$. After that the matrix is normalizing by its degree matrix For numerical stability, as in $A^{(l)} = D^1 Q_1 Q_2$. Now our model is ready to learn meta-paths from edge types (Views) and path lengths. The meta-paths which has the arbitrary chosen lengths l can be calculated with $A_P = \left(\sum_{t_1 \in T^e} \alpha_{t_1}^{(1)} A_{t_1} \sum_{t_2 \in T^e} \alpha_{t_2}^{(2)} A_{t_2} ... \sum_{t_l \in T^e} \alpha_{t_l}^{(l)} A_{t_l} \right)$ (2) where A_P is the adjacency matrix of meta-paths,T^e is a set of edge types and $\alpha_{tl}^{(l)}$ denotes the weight for edge type t_l at the lth GVT layer. A_P became the weighted sum of all length-l meta-path adjacency matrices.

Since, long meta-paths and short meta-paths can be both important we add identity matrix I in A, i.e., $A_0 = I$. This approach led us to learn all length of meta-paths. We used standard cross entropy as loss function.Second part of our model which is responsible from classification of graphs.

Graph Classification by Using Reinforcement Learning. In the second part of our model we have used RNN (Recurrent neural network). In every step our core network collects information and combine it with the information from previous steps. In every step our model predicts the graph label and prioritize for the further steps.

Our step module considers the first generation neighborhood of the current node c_{t-1} and select a neighbor c_t according to rank vector, r_{t1} by looking label and rank of node. After it chooses it makes the move for the next step. The selected nodes attribute vector is then extracts and contributes together with r_{t1} to create the step representation $s_t = f_s(d_{ct}, r_{t-1}; \theta_s)$ (see Figure 1). The step representation s_t has the updated information for our model.

We have different kind of nodes and they helps us for the exploration of graph. The type of the node can chance according to experiment. The node types in our experiment are 5 main brain parts. So how this node types contribute our model? The node types is calculated based on the statistics which is encoded in the first-level neighborhood of each node. With the help of that we can differentiate between the same brain parts.

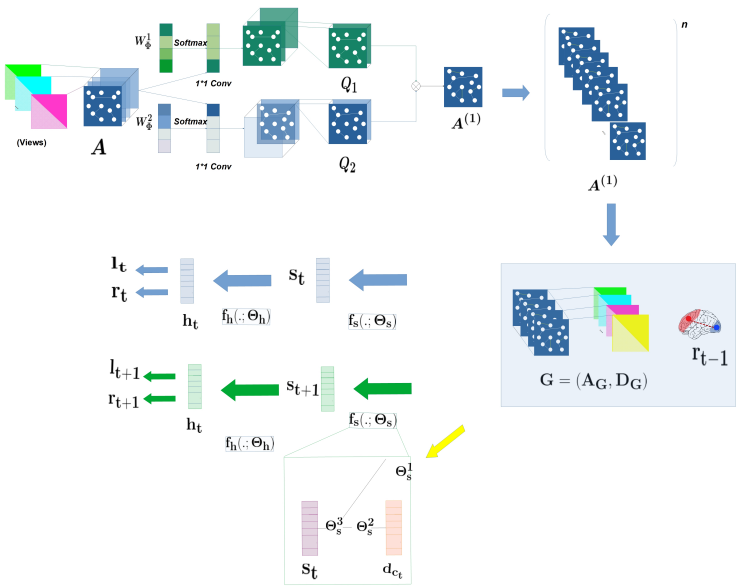

Fig. 1: The first part of TGCSA, GVT chooses Views adjacency matrices (adjacency matrices) from the set of Views A of graph G and learns a new meta-path graph denoted by $A^{(1)}$ Then, it learns a new graph structure by the matrix multiplication of this two graphs, $Q_1 Q_2$. The selection of views is a weighted sum of candidate views calculated by 11 convolution with non-negative weights from softmax (W_ϕ^1). After n times running we are creating a set of A^1's for the next step of TGCSA which is doing graph classification by using reinforce learning. As input we give a labeled graph G (set of the adjacency matrix A_G (A^1)'s and the attribute matrix D_G). Stochastic rank vector denotes r_{t1} and current node denotes c_{t1}. We use a step module for our agent to gather information from the nodes. The step module takes a step from the current node c_{t1} to one of its neighbors c_t,prioritizing those whose type(node labels which are brain parts) have higher importance in r_{t1}. By using the linear layer parameterized, θ_2 the attribute (feature) vector of c_t,d_{c_t},is extracted and matched to a hidden space. Similarly, r_{t1} matches with θ_1's. Then this two source combined and parameterized by θ_3 s and produce s_t, which is called step embedding vector. It denotes the information that we have collect from our current step. We use an RNN as the core component of the model; in particular, we apply the Long Short-Term Memory (LSTM) variant [15]. Our core network $f_h(.;\theta_h)$ is taking the current step information s_t and previous step information h_{t1} as an input and updates the history vector h_t. History vector h_t can be thought as the combined information that we have gather by our agent so far. Our rank network $fr(.;\theta_r)$ uses this uses this information for deciding which nodes are more important in next steps. And by using s_{h_t} our classification network $f_c(.;\theta_c)$ do predictions for graph label in every step.

Our models core based on history vector. It has the summary of all information which has been collected by the agent in each step. In each time step new information

taken by s_t , update the history vector by $h_t = f_h(s_t, h_{t1}; \theta_h)$. This helps model to integrate information over time.

With the information taken from history vector that collected so far our agent execute two missions. It predicts the label of the input graph $l_t = argmaxiP(y = i|f_c(h_t; \theta_c))$ by using the softmax output of the classification network learned from h_t. And it uses the rank network to update the rank vector $r_t = f_r(h_t; \theta_r)$ that will help further exploration according to importance of each node type.

As a summary, the rank vector's mission is to clarify the importance of different types of nodes. In most of the reinforcement learning settings, the agent gather new information x_{t+1} from the graph and create a reward signal r_{t+1} at each time t. In our model, $x_{t+1} = d_{c_{t+1}}$ and the reward is calculated only at the end, $r_T = 1$ if the model predict the graph label correctly, it is $r_T = 1$ if it can not it is $r_T = 1$. Because of that we can say $R = r_T$.

Therefore our model has Partially Observable Markov Decision Process (POMDP). In this process, we only obtain partial information from the graph with each time step. Our main goal is to learn the policy $\pi((r_t, l_t|s_{1:t}; \theta))$ by using parameters $\theta.\theta$ maps the set of our past interactions with the environment, $s_{1:t} = x_1, r_1, l_1, ..., x_{t1}, r_{t1}l_{t1}, x_t$ in order to a partition actions for the current time step t.

So, the summary of our past interactions is h_t, the classification network denotes as $f_c(.; \theta_c)$ and the rank network denotes as $f_r(.; \theta_r)$ or with other name our policy networks. And they learn to realize actions that maximizing reward.

Our core network, step network and rank network working together to create policy of our agent. Because of the fact that in each policy our agent leads over possible interaction sequences,$s_{1:T}$

We are trying to maximize the reward under this formulation $J() = E_{P(s_{1:T};\theta)}[R]$. It is not a easy task to maximize J exactly since we are working with numerous number of interaction sequences. But, since we specify our model's approach as a POMDP, we can get a sample approximation of the gradient of J by using the technique given by $\nabla_\theta J \frac{1}{M} \sum_{i=1}^{M} \sum_{t=1}^{T-1} \nabla_\theta log\pi(r_t^i[\tau(c_{t+1}^i)]|s_{1:t}^i; \theta)\gamma^{Tt}R^i$ (3) The $s^{i\prime}s$ are the interaction sequences from working agent under the policy for $i = 1, ..., M$ episodes $\gamma \in (0, 1]$ is a discount factor which led us to attribute more certain actions that performed closer to time T or when the prediction was made. $\tau(c_{t+1}^i)$ denotes a function which match a node with its type.

The logic also known as the REINFORCE rule. Our agent run with the current policy to get samples of interaction sequences. We increase the probability of choosing the true node which are frequently selected by correct predictions with the help of the parameters θ. By training the policy with this approach we increase the chance of agent to choose true steps to a certain type of node when it is in similar conditions. To compute: $\nabla_\theta log\pi(r_t^i[\tau(c_{t+1}^i)]|s_{1:t}^i; \theta)$,

In summary we calculate the gradient of our network in every time step. We are using standard back propagation [16]. The gradient estimate in Equation 3 can show result of high variance. We may choose to estimate $\nabla_\theta J \frac{1}{M} \sum_{i=1}^{M} \sum_{t=1}^{T-1} \nabla_\theta log\pi(r_t^i[\tau(c_{t+1}^i)]|s_{1:t}^i; \theta)(\gamma^{Tt}R^i - b_t^i)$ (4) instead. This will give us

same results with lower variances. Here $b_t^i = f_b(s_{1:t}^i; \theta_b) = f_b(h_t^i; \theta_b)$ this part captures the total reward we can get to take for a state h_t^i [17]. The term $\gamma^{Tt} R^i - b_t^i$, or we can named it as the advantage of choosing an action, allows us to increase the log-probability of actions. As a result we expect larger total reward and lower log-probability of actions that resulted reverse. We can train the parameter θ_b of f_b by decreasing the mean squared error of $R^i b_t^i$. At the end, we use cross entropy loss to train the classification network $f_c(.; \theta_c)$ by maximizing $log\pi(l_T|s_{1:T};_c)$,where l_T is the true predicted label of the input graph G. As in [17], we use this combined loss formulation where the rank network f_r is trained at each time step using REINFORCE Learning. We use supervised learning for the classification network f_c and the base line network f_b.

2.2 Graph-Level Similarity-Based GAT(GrlS-GAT)

In the following, we present the main steps of Graph-Level Similarity-Based GAT (GrlS-GAT) for predicting the presence of brain disease from brain connectcome. **Fig.** 2 provides a whole view of the proposed GrlS-GAT structure in four main steps, which will be outlined below.

Feature Importance Learning. Note that, the following importance learning structure run over training set and the test set remains untouched. To construct similarities matrix, the distance between two samples ignores the importance of the specific features to the outcome. We have to optimize the distance value according to the importance of the features for more robust model. Extra Trees Classifier is used to derive feature importance matrix. *Extremely Randomized Trees Classifier(Extra Trees Classifier)* is a form of ensemble learning model that combines the outcomes of many unrelated decision-making trees that are clustered in a "forest" to generate the outcomes of classification. To carry out importance learning, the following steps are used. First, the entropy of the sample has to be calculated. **Entropy(S)** $= \sum_{i=1}^{c} -p_i \log_2(p_i)$,where c is the number of unique label class and p_i is the proportion of rows with output label i. Then each decision tree compute the gain of the feature included by split. **Gain(S,A)** $= Entropy(S) - \sum_{v_i \in Values(A)} \frac{|S_{v_i}|}{|S|} Entropy(S_{v_i})$,where A is the feature whose gain will be computed, v_i is the unique value belongs to the A feature columns and $||$ denotes the number of sample in the specific set. Each decision trees computes the feature importance according to the above mathematical criteria. Then each specific feature gain values are aggregated from the decision trees.The gain values is stored feature importance vector which is denoted by $\vec{F}$.

Update Euclidean Distance Matrix according to derived feature importance vector. In this section, we present the construction of the similarity matrix which is input to the next stage that creates adjacency matrix according to the similarity values between instances. Firstly, the raw euclidean distance computed but this distance computation ignores the importance of the features to the outcome. So previously derived $\vec{F}$ will be used to optimize distances for more robust model. D'Agostino and Dardanoni stated that euclidean distance is often used as a loss function in statistics and decision theory;in psychology, related measures are often employed to evaluate the amount of "similarity" between two objects, each of which is decomposed into a

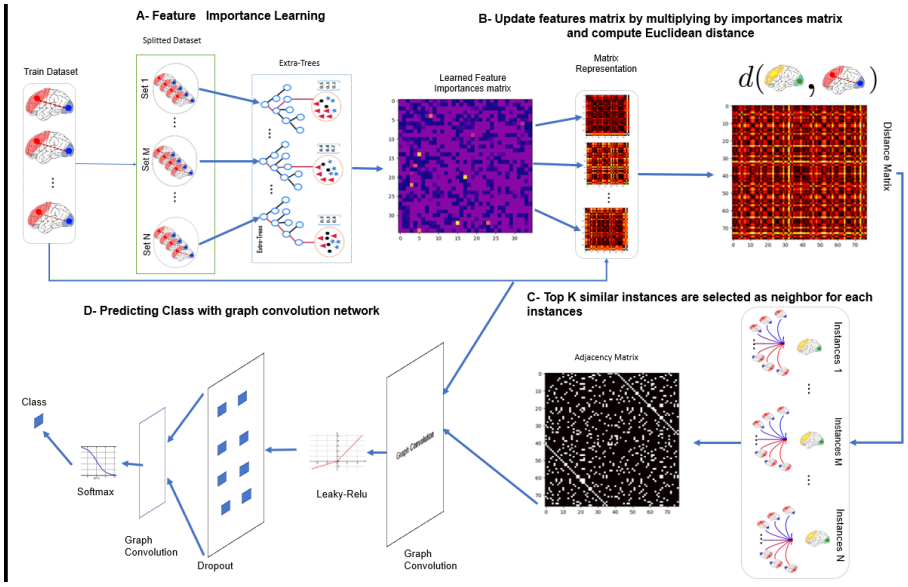

Fig. 2: *Graph-Level Similarity-Based GAT(GrlS-GAT) steps for predicting brain disease from the brain connectome.***(A)** *Feature Importance Learning*. Using Extra-Tree Classifier to derive the importances of features through the labels. These importances is stored in the feature importance vector which is denoted as $\vec{\boldsymbol{F}}$. **(B)** *Update Euclidean Distance Matrix according to derived feature importance vector*. The euclidean distances between each graph instance is computed and stored in the distance matrix which is denoted as $\boldsymbol{D}$ is updated row-wised manner via multiplying with $\vec{\boldsymbol{F}}$. The updated matrix is denoted as $\boldsymbol{D'}$. **(C)** *Top K similar instances are selected as neighbor for each instances.* The adjacency matrix is created from the $\boldsymbol{D'}$. $\boldsymbol{D'}_{ij}$ is the distance between i and j instances. For each instance the top K nearest distance values is selected as neighbor and according to this relationship the adjacency matrix which is denoted as $\boldsymbol{A}$ is set.**(D)** *Predict Class.* The weight matrix is set from the features at the initial stage and this matrix is denoted as $\boldsymbol{W}$. The $\boldsymbol{A}$ and $\boldsymbol{W}$ is provided as input. The prediction made according to this input and the loss value is computed, then the internal stages are updated according to the loss value via Adam optimizer.This procedure is actually is a graph convolution network which use leaky-Relu, dropout and softmax through predicting class.

fixed number of components, and (dis)similarity is then modelled as a suitable metric in the resulting feature space [18]. The euclidean distance between two instances p and q is $\mathbf{d(p,q)} = \sqrt{(p_1 - q_1)^2 + (p_2 - q_2)^2 + ... + (p_n - q_n)^2}$ equals to $\sqrt{\sum_{i=1}^{n}(p_i - q_i)^2}$, where i is the i^{th} feature and n is the number of the feature. As it mentioned before, the raw euclidean distance discards the importance of feature to outcome, so the distance values can be illusive. To handle with this issue, the multiplication of feature importance and euclidean distance is used as the optimization of distance value. This optimized distance can be denoted as d' and the optimized distance between instances

p and q is $d'(p,q) = \sqrt{\sum_{i=1}^{n} F_i(p_i - q_i)^2}$, where F_i is the importance of the i^{th} feature. The optimal distances is stored the updated Euclidean Distance Matrix denoted as D'.

Adjacency matrix construction. The adjacency matrix is constructed based on the K-value which is a hyper-parameter of this stage. Top K similar instances excluding itself for each instance is selected as neighbour to instance. The optimized euclidean distance matrix D' stored the distance value between instances. For instances D'_{ij} is the distance to j instance from i instance.

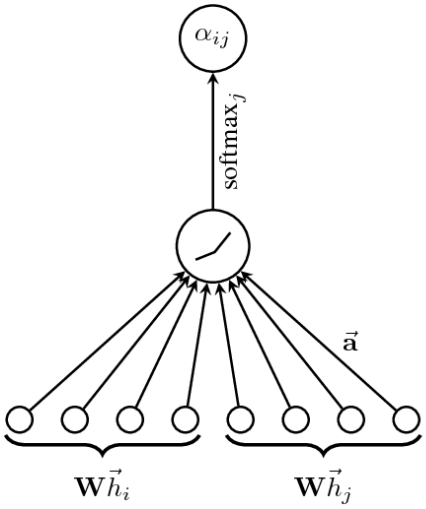

Fig. 3: *The attention mechanism a($\mathbf{W}\vec{h}_i, \mathbf{W}\vec{h}_j$) employed by our model, parametrized by a weight vector $\vec{\mathbf{a}} \in \mathbb{R}^{2G}$, applying a LeakyReLU activation.*

Graph-Level Similarity-Based GAT. In this section, we present the novel neural network architecture that strengthen the raw GAT architecture. **Fig.** 3 provides the scheme of attention mechanism which computes importance between two instances. Firstly, weight matrix **W** which is a parameter for attention mechanism derived from the set of graph features rather than zero matrix as initial point. In the attention mechanism the weight matrix $\mathbf{W} \in \mathbb{R}^{G' \times G}$, where G' is the cardinality of the set of graph features which is the output of the graph attention layer and G is the cardinality of the set of graph features which is the input to the graph attention layer. We used the set of graph features to derive initial weight matrix but the cardinality of the set of graph features is static value through the learning process and does not need to equals to the cardinalities of the graph attention layer which changes dynamically through the learning process. To solve this issue, matrix multiplications with the matrices contain only ones is applied to derived matrix from the set of graph feature on both left and right side. In this way, the inconsistency of cardinalities can be solved. Graph attention layer used attention mechanism to compute the importance between instances. The attention mechanism which is denoted as a used weight vector $\vec{\mathbf{a}} \in \mathbb{R}^{2G'}$ and LeakyRelu non-linearity as activa-

tion function. The attention mechanism a computes the importance between i and j instances e_{ij} can be expressed as mathematically: $e_{ij} = LeakyRelu(\vec{\mathbf{a}}^T[\mathbf{W}\vec{h}_i||\mathbf{W}\vec{h}_j])$, T denotes transpose and $||$ denotes concatenation. Veličković et al. stated the problem as the model allows every node to attend on every other node, dropping all structural information [2]. To address this issue, the importance values can be optimized as follows: $\alpha_{ij} = \frac{exp(e_{ij})}{\sum_{k \in N_i} exp(e_{ik})}$, where N_i is the neighbours of instance i and α_{ij} is the optimized importance instance j to instance i. After the attention mechanism a computed the optimized attention coefficients, the attention layer construct new feature vector which is higher level than initial point for each graph. The high level feature vector of i instance $\vec{h_i}' = LeakyRelu(\sum_{j \in N_i} \alpha_{ij}\mathbf{W}\vec{h}_j)$. Leaky Rectified Linear Unit(LeakyRelu) is a type of non-linear activation function. Non-linear activation function play a crucial role in neural network because linear activation function usage on network layers performs as a one layer network. The outcome of network is actually the linear combination of input even it use many layers in its architecture. As Nwankpa et al. stated that LeakyRelu add negative slope to Rectified Linear Unit(ReLu) activation function a solution to dead neuron problems such that the gradients will not be zero through the training [19]. The LeakyRelu can be computed as follows:

$$f(x) = \begin{cases} x, & \text{if } x > 0 \\ \beta x, & \text{if } x \leq 0 \end{cases} \tag{1}$$

β is the very small value typically equals to 0.01 at the equation (1). After computing the high level feature vectors for each graph. The model begins to perform forward and backward propagation like Graph convolution network. The Adam optimizer is used to update weight matrices at each layer. The Adam optimizer use two moments in its computation: mean of the gradient and decentralized variance of the gradient. As Kingma and Ba stated the Adam optimizer is most trending method nowadays because it is straightforward to implement, is computationally efficient, has little memory requirements, is invariant to diagonal re-scaling of the gradients, and is well suited for problems that are large in terms of data and/or parameters [20].

$$\hat{m}_t = \frac{m_t}{1 - {\gamma_1}^t} \tag{2}$$

$$\hat{v}_t = \frac{v_t}{1 - {\gamma_2}^t} \tag{3}$$

$$w_t = w_{t-1} - \eta \frac{\hat{m}_t}{\sqrt{\hat{v}_t} + \epsilon} \tag{4}$$

The equation (4) is the update rule for Adam optimizer, the model weight denoted by w, η represents the learning rate and ϵ is a small value to prevent the zero division. The equations (2) and (4) optimized mean and uncentered variance of the gradients. The γ_1 and γ_2 are hyper-parameters of the algorithm and their default values are 0.9 and 0.999 respectively.

GrlS-GAT has to handle model over-fitting like the other deep neural network do. Deep neural networks are very strong machine learning systems, but over-fitting is a

bottleneck for these systems. Srivastava et al.introduced Dropout [21] to solve this problem. Dropout solve this issue by turning off some units in training stage. A unit presents in training at the probability of p and the all units are present in test stage. For any layer l, $r^{(l)} = Bernoulli(p)$ is a vector of independent Bernoulli random variables each of which has probability p of being 1 [21]. The output vector of l layer $y^(l)$ is optimized as follows: $\hat{y}^(l) = y^(l) * r^(l)$, where * denotes element-wise product and $\hat{y}^(l)$ is optimized output vector of l layer.

3 Results and Discussion

Evaluation dataset. We used 5-fold cross validation to evaluate the proposed network evolution prediction framework on 77 subjects (41 patient and 36 non-patient). Each subject consist of 35 morphological connectivity strength between two brain region. We used same dataset for our two approaches. We evaluate dataset separately in order to make suitable for our methodologies. In GrlS-GAT, we evaluate each subject as graph that represent connectivity of the 35 nodes in two brain region. In GrlS-GAT attention layer used weight matrix that derived from the dataset also. The cardinality of this weight matrix is determined by input and output size and it can be change dynamically through the training and the cardinality of feature matrix does not need to suit with the weight matrix because such requirement limits the usage of these model over different dataset. We multiply the weight matrix that derived features on both left and right with two matrices of ones that cardinality selected according to the size of input and output to deal with these issue. GrlS-GAT learns importance of each feature via Extra-Tree Classifier and stored the value in the importance vector. GrlS-GAT compute the euclidean distances of each subject from the feature matrix and stored the values in distance matrix. After that the euclidean distances is updated by feature vector because raw euclidean distances can be illusive while looking the similarities of subjects. According to the hyper-parameter k, the top similar subjects are selected as neighbours to each subject. The 77×77 adjacency matrix is created from the neighbour relation. The top k similar objects to each object selecting as neighbours to each object and the value of these neighbours to the object in adjacency matrix set as 1 and to the others objects from the object set as 0.

Transformed Graph Classification using Structural Attention (TGCSA) has two parts which are GVT and Graph Classification. We are doing some arrangements for each part separately.

For GVT we give 77 sample connectivity matrices individually without doing any chancing. All of them has 35×35 float connectivity value. All samples has 4 views.Since the main goal of GVT is combining the views,it convert 35×35×4 matrices into 35×35 matrix $A^{(l)}$. We run this process 77 times in order to cover all samples.The output of GVT became 77 subject which are in the shape of 35×35 .

For Graph Classification part we are using the output of GVT.In here we are using supervised and reinforcement learning together. Since we use supervised learning we are giving graph labels (41 Patient 35 Non-Patient as input). Our code do prioritizing for nodes as well. So we give node labels as an input. While creating this labels we map 35 brain node with 5 actual brain parts which are Temporal Lobe, Limbic Lobe, Frontal

Lobe, Occibital Lobe and Parietal Lobe. Our code is not suitable for weighted graphs.So we do thresholding for making float values ones and zeros. We use 3 different threshold value which are mean,mean+standard division and mean-standard division. We test and train each of them separately.

Parameter setting. For fair comparison across all methods, we fixed the random seed as 5 at the beginning. The two approaches use 5-fold cross validation so we used some model for two approaches. The sklearn.model-selection.KFOLD ready model is used to perform 5-fold cross validation. K-Folds cross-validator Provides train/test indices to split data in train/test sets. Split dataset into k consecutive folds (without shuffling by default). Each fold is then used once as a validation while the k - 1 remaining folds form the training set. The parameters of model are set as follows: n-splits = 5, shuffle = True and random-state = 5. The upper bound of the epoch count is set to 200, the learning rate $lr = 0.01$, $\gamma_1 = 0.9$,$\gamma_2 = 0.999$ and weight decay is $e^{-0.5}$ for shared variable by two approaches. Note that γ_1 and γ_2 are hyper-parameters for Adam optimizer. GrlS-GAT use $k = 5$ as the number of neighbours to each object.

		LH(MEAN-STD)	
%	Accuracy	Sensitivity	Specificity
MEAN	42.86	47.62	21.43
TGCSA	53.24	54.39	50.00
		LH(MEAN)	
MEAN	49.35	51.56	38.46
TGCSA	55.84	62.07	52.08
		LH(MEAN+STD)	
MEAN	50.65	52.31	41.67
TGCSA	**61.03**	**66.67**	**56.82**
		RH(MEAN-STD)	
MEAN	48.05	50.72	25.00
TGCSA	53.24	55.56	50.00
		RH(MEAN)	
MEAN	51.95	52.86	42.86
TGCSA	57.14	61.11	53.66
		RH(MEAN+STD)	
MEAN	53.24	53.62	50.00
TGCSA	**59.74**	**64.71**	**55.81**

Fig. 4: shows the result of TGCSA. It consists comparison of 2 methods which are Mean (Mean of Views) and TGCSA. We use data set of 3 thresholding for 2 brain hemisphere (Left and Right). The results are given according to Accuracy,Specificity and Sensitivity.

Comparison methods and evaluation. Fig. 4 shows the results of Transformed Graph Classification using Structural Attention (TGCSA). We have two method on the table. The first one which is mean of the connectivity matrices (views) are calculated by the mean of 4 views for each sample graph. They are not giving to GVT as an input but directly used for graph classification.In the second method which is (TGCSA) we use GVT and graph classification together.We have three evaluation criteria which are

Left Hemisphere Combined View			
%	Accuracy	Sensitivity	Specificity
GAT	**53.584**	**71.43**	**50.79**
Mid-model	**53.584**	64.71	50
GrlS-GAT	**53.584**	69.23	50
Right Hemisphere Combined View			
%	Accuracy	Sensitivity	Specificity
GAT	53.584	54.10	50
Mid-model	53.584	54.39	47.62
GrlS-GAT	**55**	**54.69**	**53.85**

Fig. 5: is the result that taken from GrlS-GAT uses concatenation of the 4 views as a feature matrix. GrlS-GAT improves accuracy, sensitivity and specificity at little rate for right hemisphere subject. On the other hand it does not improves the accuracy and even decreases slightly sensitivity, specificity for left hemisphere subject. The mid-model is a first edition of GrlS-GAT that use zero weight matrix $\mathbf{W}$ rather than derived one from the feature matrix initially.

Right Hemisphere First View			
%	Accuracy	Sensitivity	Specificity
GAT	46.416	49.15	33.33
Mid-model	**57.166**	**57.14**	**57.14**
GrlS-GAT	53.584	53.85	50
Right Hemisphere Second View			
%	Accuracy	Sensitivity	Specificity
GAT	45.584	48.33	29.41
Mid-model	49.582	51.72	42.11
GrlS-GAT	**53.584**	**53.85**	**50**
Right Hemisphere Third View			
%	Accuracy	Sensitivity	Specificity
GAT	46.416	49.15	33.33
Mid-model	38.516	43.86	20
GrlS-GAT	**53.584**	**53.85**	**50**
Right Hemisphere Fourth View			
%	Accuracy	Sensitivity	Specificity
GAT	46.416	49.15	33.33
Mid-model	49.082	51.79	42.86
GrlS-GAT	**53.584**	**53.85**	**50**

Fig. 6: is the result that taken from GrlS-GAT uses each view seperately as a feature matrix. GrlS-GAT improves accuracy, sensitivity and specificity for right hemisphere subject compared to GAT. On the other hand it does not improves the accuracy, sensitivity and specificity for first view. The mid-model is a first edition of GrlS-GAT that use zero weight matrix $\mathbf{W}$ rather than derived one from the feature matrix initially.

accuracy, sensitivity (True Positive) and specificity (False Negative). Since our data set use thresholding, we have 3 type of measurement for 2 brain hemisphere.

Left Hemisphere First View			
%	Accuracy	Sensitivity	Specificity
GAT	46.416	50	45.90
Mid-model	49.584	56.25	47.54
GrlS-GAT	**52.252**	**64.29**	**49.21**
Left Hemisphere Second View			
%	Accuracy	Sensitivity	Specificity
GAT	46.416	50	45.90
Mid-model	50.498	60	48.39
GrlS-GAT	**53.584**	**66.67**	**50**
Left Hemisphere Third View			
%	Accuracy	Sensitivity	Specificity
GAT	53.084	**71.43**	50.79
Mid-model	47.666	53.85	46.88
GrlS-GAT	**53.584**	69.23	**50**
Left Hemisphere Fourth View			
%	Accuracy	Sensitivity	Specificity
GAT	46.416	50	45.90
Mid-model	40	33.33	41.94
GrlS-GAT	**53.584**	**69.23**	**50**

Fig. 7: is the result that taken from GrlS-GAT uses each view seperately as a feature matrix. GrlS-GAT improves accuracy, sensitivity and specificity for left hemisphere subject compared to GAT. On the other hand it decreases sensitivity for the third view. The mid-model is a first edition of GrlS-GAT that use zero weight matrix **W** rather than derived one from the feature matrix initially.

TGCSA increased all the results in every condition. It increased the accuracy of LH and RH approximately $\%6$ for every threshold. For left hemisphere, It gave the best value for the threshold of mean+standard deviation with $\%61.03$.When we increase the value higher than this, total the accuracy decreased. So we count it as the optimum threshold.For RH the result was the same with same threshold which is $\%59.74$. When we look at the sensitivity and specificity results, we again see threshold of mean+standard deviation gives the highest results.It increases the results for both hemisphere, for all thresholds approximately $\%10$. So we can understand that TGCSA not only raises the result for graph labels 1's but also label 0's. When we increase the threshold higher than mean+standard deviation,we observe the same decrease on the results of sensitivity and specificity.

Fig. 5, **Fig.** 6 and **Fig.** 7 are the result table of GrlS-GAT. The mid-model is first edition of GrlS-GAT that does not use derived weight matrix **W** from the feature matrix. **Fig.** 5 is the result that taken from GrlS-GAT uses concatenation of the 4 views as a feature matrix. As seen from **Fig.** 5 GrlS-GAT increases the accuracy, sensitivity and specificity for the right hemisphere subjects. On the other hand, there is no improvement in accuracy and even slightly decreasing in sensitivity and specificity for left hemisphere subjects. This shows that the concatenation of the 4 views is not a good feature to predict the class. Therefore the result for this dataset is not stable and we can not depend GrlS-GAT on concatenation of the 4 views. **Fig.** 6 shows result of the right hemisphere dataset that is evaluated each view separately.**Fig.** 7 shows result of the left hemisphere dataset that is evaluated each view separately. Each table shows that GrlS-GAT improve the

accuracy value by %7 approximately. GrlS-GAT improves the specificity and sensitivity values for most dataset also.

Comparison of Two Approaches. In this part we are comparing two different method which are GrlS-GAT and TGCSA.When we look at the combine view results GAT and Mean approximately same results for both hemispheres.If we look the results of sensitivity and specificity the results are again similar besides few exceptions. Because they are using similar attention mechanism.Results are as expected.

When we compare the method of GrlS-GAT and TGCSA we can say that TGCSA has %5 more accuracy. If we compare sensitivity and specificity results, they are slightly the same.

As a conclusion we can say that combining views with unsupervised learning (GVT) raise the accuracy of the results.However since we are working with similar attention mechanisms sensitivity and specificity results only slightly chanced.

4 Conclusion

In this work, we study the problem of class classification with 2 attention-based mechanisms. With Graph-level Similarity-Based GAT (GrlS-Gat) we create adjacency matrix via Graph Similarity Learning. We combine Graph-level learning with node-level learning. We take initial weights from features rather than zero weight matrix and we use dropout to solve model over-fitting.In Transformed Graph Classification using Structural Attention (TGCSA) we convert a supervised Graph Transformation method to an unsupervised approach (GVT). We combine two approaches (GVT and Graph Classification using RNN) together. We use brain parts in Gam as Node Labels. We apply mean and standard division for optimum thresholding. We compare two methods and took higher results with our approach (Mean of views and TGCSA).

We used 5-fold cross-validation for both approaches (GrlS-Gat and TGCSA) with the same seeds in order to make our methods deterministic.

For future work, we would like to combine our approaches GVT and GrlS-Gat. In GrlS-Gat we prepared our combined adjacency matrix by concatenate operation. However as we can observe from the result of TGCSA, an approach that is using unsupervised learning for combining views raise the results significantly. So we aim to raise the result of GrlS-Gat with using our latest approaches as ground truth.

References

1. Rossi, R., Zhou, R., Ahmed, N.: Estimation of graphlet statistics. (2017)
2. Veličković, P., Cucurull, G., Casanova, A., Romero, A., Liò, P., Bengio, Y.: Graph attention networks (2017)
3. Duvenaud, D., Maclaurin, D., Aguilera-Iparraguirre, J., Gómez-Bombarelli, R., Hirzel, T., Aspuru-Guzik, A., Adams, R.: Convolutional networks on graphs for learning molecular fingerprints. Advances in Neural Information Processing Systems (NIPS) **13** (2015)
4. Backstrom, L., Leskovec, J.: Supervised random walks: Predicting and recommending links in social networks. (2011) 635–644
5. Bao, J., He, T., Ruan, S., Li, Y., Zheng, Y.: Planning bike lanes based on sharing-bikes' trajectories. (2017)

6. Chau, D.H., Nachenberg, C., Wilhelm, J., Wright, A., Faloutsos, C.: Polonium: Tera-scale graph mining and inference for malware detection. (2011) 131–142
7. Henaff, M., Bruna, J., LeCun, Y.: Deep convolutional networks on graph-structured data (2015)
8. Defferrard, M., Bresson, X., Vandergheynst, P.: Convolutional neural networks on graphs with fast localized spectral filtering (2016)
9. Kipf, T.N., Welling, M.: Semi-supervised classification with graph convolutional networks (2016)
10. Yun, S., Jeong, M., Kim, R., Kang, J., Kim, H.: Graph transformer networks (2019)
11. Williams, R.: Simple statistical gradient-following algorithms for connectionist reinforcement learning. Machine Learning **8** (1992) 229–256
12. Lee, J., Rossi, R., Kong, X.: Graph classification using structural attention. (2018) 1666–1674
13. Wang, X., Ji, H., Shi, C., Wang, B., Ye, Y., Cui, P., Yu, P.: Heterogeneous graph attention network. (2019) 2022–2032
14. Zhang, Y., Xiong, Y., Kong, X., Li, S., Mi, J., Zhu, Y.: Deep collective classification in heterogeneous information networks. (2018) 399–408
15. Weston, J., Chopra, S., Bordes, A.: Memory networks. (2014)
16. Wierstra, D., Förster, A., Peters, J., Schmidhuber, J.: Solving deep memory pomdps with recurrent policy gradients. Volume 4668. (2007) 697–706
17. Mnih, V., Heess, N., Graves, A., Kavukcuoglu, K.: Recurrent models of visual attention. Advances in Neural Information Processing Systems **3** (2014)
18. D'Agostino, M.: What's so special about euclidean distance? a characterization with applications to mobility and spatial voting. Social Choice and Welfare **33** (2009) 211–233
19. Nwankpa, C., Ijomah, W., Gachagan, A., Marshall, S.: Activation functions: Comparison of trends in practice and research for deep learning (2018)
20. Kingma, D., Ba, J.: Adam: A method for stochastic optimization. International Conference on Learning Representations (2014)
21. Srivastava, N., Hinton, G., Krizhevsky, A., Sutskever, I., Salakhutdinov, R.: Dropout: A simple way to prevent neural networks from overfitting. Journal of Machine Learning Research **15** (2014) 1929–1958