A Dynamic Model for Early Prediction of Alzheimer's Disease by Leveraging Graph Convolutional Networks and Tensor Algebra

Cagri Ozdemir[1,3,4,†],Mohammad Al Olaimat[1,3,4], Serdar Bozdag[1,2,3,4,†], and Alzheimer's Disease Neuroimaging Initiative*

[1]*Department of Computer Science and Engineering, University of North Texas, TX, USA*

[2]*Department of Mathematics, University of North Texas, TX, USA*

[3]*BioDiscovery Institute, University of North Texas, TX, USA*

[4] *Center for Computational Life Sciences, University of North Texas, TX, USA*
[†]*E-mail: cagri.ozdemir@unt.edu, serdar.bozdag@unt.edu*

Data used in preparation of this article were obtained from the ADNI database (adni.loni.usc.edu). As such, the investigators within the ADNI contributed to the design and implementation of ADNI and/or provided data but did not participate in analysis or writing of this report. A complete listing of ADNI investigators can be found at: `https://adni.loni.usc.edu/wp-content/uploads/how_to_apply/ADNI_Acknowledgement_List.pdf`

Alzheimer's disease (AD) is a neurocognitive disorder that deteriorates memory and impairs cognitive functions. Mild Cognitive Impairment (MCI) is generally considered as an intermediate phase between normal cognitive aging and more severe conditions such as AD. Although not all individuals with MCI will develop AD, they are at an increased risk of developing AD. Diagnosing AD once strong symptoms are already present is of limited value, as AD leads to irreversible cognitive decline and brain damage. Thus, it is crucial to develop methods for the early prediction of AD in individuals with MCI. Recurrent Neural Networks (RNN)-based methods have been effectively used to predict the progression from MCI to AD by analyzing electronic health records (EHR). However, despite their widespread use, existing RNN-based tools may introduce increased model complexity and often face difficulties in capturing long-term dependencies. In this study, we introduced a novel **Dy**namic deep learning model for **E**arly **P**rediction of **AD** (DyEPAD)* to predict MCI subjects' progression to AD utilizing EHR data. In the first phase of DyEPAD, embeddings for each time step or visit are captured through Graph Convolutional Networks (GCN) and aggregation functions. In the final phase, DyEPAD employs tensor algebraic operations for frequency domain analysis of these embeddings, capturing the full scope of evolutionary patterns across all time steps. Our experiments on the Alzheimer's Disease Neuroimaging Initiative (ADNI) and National Alzheimer's Coordinating Center (NACC) datasets demonstrate that our proposed model outperforms or is in par with the state-of-the-art and baseline methods.

Keywords: Alzheimer's disease, early prediction, dynamic graphs, tensor algebra.

*The source code is available at `https://github.com/bozdaglab/DyEPAD`

1. Introduction

Throughout the past few decades Alzheimer's disease (AD), once thought to be a rare disorder, has gained recognition as a major public health concern.[1,2] According to the survey data,[3] AD affected more than 30 million people in 2015, and it is estimated that this number could surpass 114 million by 2050. AD causes an irreversible decline in memory, mood, and behavior, along with difficulties with everyday tasks and other cognitive challenges. Mild Cognitive Impairment (MCI) is a condition characterized by observable cognitive decline, which is generally considered as not sufficient to effect patients' daily functioning. More importantly, MCI serves as a critical stage for identifying individuals at risk of developing AD. Individuals with MCI are at a higher risk of progressing to AD, with an annual progression rate between 10% and 20%.[4] Although, to date, there is no complete cure for AD, there are treatments to slow AD-related symptoms at their early stages. Therefore, to slow down AD progression and avoid its worst effects, it is crucial to develop methods for the early prediction of AD in individuals with MCI.

Many early prediction tools for AD mainly rely on image data, making use of advanced imaging technologies like MRI, PET scans, and CT scans.[2,5–8] However, even though image-based approaches provide valuable insights, imaging is an expensive method and is not easily accessible particularly for people in developing countries. Electronic health records (EHR) consist of temporal sequences of clinical features. The longitudinal nature of EHR enables the examination of patients' medical history trajectories. These records have been utilized to train machine learning (ML) models for classifying and clustering patient data, enhancing clinical decision-making.[9,10] However, traditional ML methods (e.g., Random Forest and SVM) fail to account for the temporal dependencies in the data sequences.[11] An effective method for capturing the temporal patterns in sequential data is Recurrent Neural Networks (RNN). However, irregular time intervals between consecutive inputs (i.e., clinical visits of patients), a common occurrence in EHR, pose a challenge for RNN models.[12] When intervals vary, it disrupts the model's ability to effectively capture temporal dependencies and may lead to suboptimal performance. To address this challenge, Time-aware long short-term memory (T-LSTM)[13] has been introduced. T-LSTM modifies LSTM architecture to address challenges arising from irregular time intervals between clinical visits. Another computational tool, named Predicting Progression of Alzheimer's Disease (PPAD),[14] utilizes an RNN component where patients' ages at the time of clinical visits were utilized to handle varying time intervals between clinical visits. More recently, time-aware RNN (TA-RNN) has been presented for early prediction of AD.[15] TA-RNN utilizes a time embedding layer that incorporates elapsed time between consecutive visits to address lack of consideration of irregular time intervals between consecutive inputs by RNN models.

To enhance graph analysis, Graph Convolutional Networks (GCN)[16] has been introduced as a more efficient variant of Graph Neural Networks (GNN). GCN utilizes a convolution operation that aggregates information from multiple hops of neighbors. While GCN treats all neighboring nodes equally during aggregation, Graph Attention Networks (GAT)[17] employs an attention mechanism to learn the importance of each neighboring node. In the context of early diagnosis of AD, a GNN-based method has been introduced that constructs patient-patient

graphs using image features from both MRI and PET scans.[18] In addition, an interpretable dynamic graph convolutional networks (IDGCN) integrates dynamic graph leaning into a GCN architecture to improve the performance of personalized diagnosis for AD and provide interpretable results.[19]

Even though these existing tools offer RNN-based solutions for the early diagnosis of AD using EHR, they have some limitations. In all these approaches, RNN parameters are updated using the entire sequence of time steps. While this methodology can capture sequential patterns, it often leads to increased complexity and potential issues with learning long-term dependencies due to RNN's structural design. As sequences get longer, they tend to forget earlier information, making it hard to capture patterns over extended periods. Additionally, longitudinal data may have hierarchical temporal structures, such as monthly and yearly patterns. RNN units often struggle to effectively capture these hierarchies. Furthermore, for long sequences, RNN layers often encounter vanishing or exploding gradient problems, which complicate the training process.

To address these limitations, in this study, we introduced a novel **Dy**namic deep learning model for **E**arly **P**rediction of **AD** (DyEPAD). DyEPAD consists of a two-phase training process. In the first phase, DyEPAD learns latent representations (i.e., embeddings) of patients at each clinical visit. For this, for each visit a patient similarity network is constructed. In each graph, the nodes represent patients with attributes derived from the corresponding clinical visit, and the edges capture the similarities between patients based on these visit attributes. At each time step, node embeddings of the corresponding graph are learned using a GCN layer and an aggregation function, which incorporates the node embeddings of the graph in the previous visit. In the second phase of DyEPAD, spatiotemporal tensor is built by stacking the embeddings learned in the first phase. Then, tensorial functions are employed to capture full scope of evolutionary pattern in the data by mapping it into a non-linear feature space and utilizing frequency domain representations.

We presented our experimental results on the Alzheimer's Disease Neuroimaging Initiative (ADNI)[20] and National Alzheimer's Coordinating Center (NACC)[21] datasets to predict AD diagnosis at the next visit and multiple visits ahead. Our experimental results show that our proposed model outperforms or is in par with the state-of-the-art and baseline methods.

2. Methods

2.1. *Preliminaries: Overview of Tensor Algebra*

Multidimensional data is defined as arrays of numbers organized in more than two dimensions, commonly known as *tensors*.[22] Unlike traditional data structures such as vectors (1D) or matrices (2D), tensors extend to higher dimensions, allowing for more complex data representations. The dimensions of a tensor are called ways or modes. The number of modes determines the order of a tensor. If, for example, $\mathcal{A} \in \mathbb{R}^{m \times n \times \ell}$, then $\mathcal{A}$ is a third-order tensor. Here, m could represent time, n could represent different patients, and ℓ could represent various clinical measurements. The definitions presented in this section are fundamentally based on recent advancements in Fourier theory and the algebra of circulants,[23–28] which provide powerful tools for analyzing multidimensional data. By using tensor-based methods, we can

better capture and model the intricate relationships across multiple modes, which are often lost in simpler, lower-dimensional representations like vectors or matrices.

It will be useful to divide a third-order tensor $\mathcal{A}$ into different slices and tubal elements as shown in Fig. 1. In Python notation, $\mathcal{A}^{(i)} \equiv \mathcal{A}[i,:,:]$ refers to the ith frontal slice; $\mathcal{A}_{(i)} \equiv \mathcal{A}[:,i,:]$ refers to the $i$the horizontal slice; and $\vec{\mathcal{A}}_{(i)} \equiv \mathcal{A}[:,:,i]$ refers to the ith lateral slice.

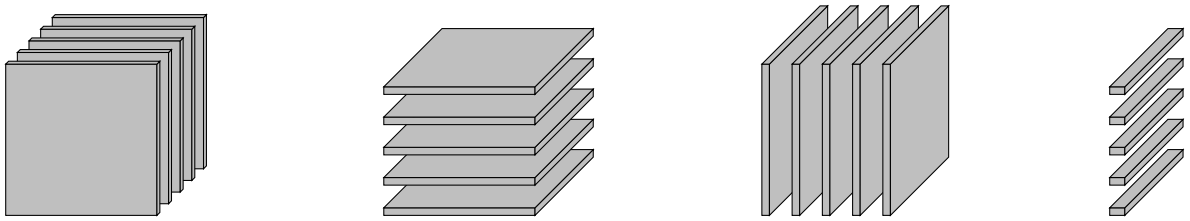

Fig. 1. (a) Frontal, (b) horizontal, (c) lateral slices, and (d) tubal scalars of a third-order tensor.

Definition 1. An element $\vec{\mathbf{a}}_i^j \in \mathbb{R}^{n\times 1\times 1}$ is called a **tubal scalar** of length n. $\vec{\mathbf{a}}_i^j \equiv A[:,j,i]$ refers to the jth tubal scalar of ith lateral slice.

Definition 2. Let $\mathcal{A} \in \mathbb{R}^{m_1\times m_2\times m_3}$ be a third-order tensor. Then $\texttt{unfold}(\mathcal{A})$ maps the tensor $\mathcal{A}$ into a $m_1 \times (m_2 \cdot m_3)$ matrix by stacking all the tubal scalars as the columns of the resultant matrix. The operation that takes $\texttt{unfold}(\mathcal{A})$ back to tensor form is the `fold` command:

$$\mathcal{A} = \texttt{fold}\big(\texttt{unfold}(\mathcal{A})\big).$$

Definition 3. Let $\mathcal{A} \in \mathbb{R}^{m_1\times m_2\times m_3}$ be a third-order tensor, and let $\mathbf{B} \in \mathbb{R}^{m_1\times m_1}$ be a matrix. The mode-1 product of $\mathcal{A}$ and $\mathbf{B}$, denoted $\mathcal{A} \times_1 \mathbf{B}$, is defined as:

$$\mathcal{A} \times_1 \mathbf{B} = \texttt{fold}\big(\mathbf{B} * \texttt{unfold}(\mathcal{A})\big),$$

where $*$ denotes matrix multiplication.

Definition 4. The face-wise product multiplies each of the frontal slices of two tensors. Let $\mathcal{A} \in \mathbb{R}^{m\times n\times k}$ and $\mathcal{B} \in \mathbb{R}^{m\times k\times \ell}$ be third-order tensors. Then the face-wise product $\mathcal{C} = \mathcal{A}\Delta\mathcal{B} \in \mathbb{R}^{m\times n\times \ell}$ is defined by performing matrix multiplication between the corresponding frontal slices of $\mathcal{A}$ and $\mathcal{B}$ as follows:

$$\begin{aligned}&\mathcal{C} = \mathcal{A}\Delta\mathcal{B},\\ &\mathcal{C}^{(i)} = \mathcal{A}^{(i)} * \mathcal{B}^{(i)} \ \text{ for } i = 1,\ldots,m.\end{aligned}$$

2.2. *Fundamental Tensor Operations*

From a theoretical perspective, it is well known that block circulant matrices can be block diagonalized by using the Fourier transform.[29] Therefore, the multiplication, transpose, and inverse operations on tensors were defined based on block circulant matrices and the Fourier transform.[23–26,30] Most recently, it was shown that these tensor operators can be effectively

defined by performing an invertible linear transform along all tubal scalars of tensors, conducting pair-wise matrix multiplications for all frontal slices of the tensors in the transform domain.[31] To this end, we will define tensor operators in the so-called transform domain. The "$_L$" subscript is used to represent any invertible linear transformation.

Definition 5. Let $\mathcal{A} \in \mathbb{R}^{m\times n\times k}$ and $\mathcal{B} \in \mathbb{R}^{m\times k\times \ell}$ be third-order tensors. The tensor-tensor product based on L transform, denoted $\mathcal{A} \circ_L \mathcal{B} \in \mathbb{R}^{m\times n\times \ell}$, is defined as:

$$\begin{aligned}
&\tilde{\mathcal{A}} = \mathcal{A} \times_1 \mathbf{L},\\
&\tilde{\mathcal{B}} = \mathcal{B} \times_1 \mathbf{L},\\
&\mathcal{A} \circ_L \mathcal{B} = (\tilde{\mathcal{A}} \Delta \tilde{\mathcal{B}}) \times_1 \mathbf{L}^{-1},
\end{aligned}$$

where $\mathbf{L}$ is an $m \times m$ invertible transformation matrix. $\mathbf{L}^{-1}$ is the inverse of the transformation matrix. "$\times_1$" is the mode-1 product and "Δ" is the face-wise product given in **Definition 3** and **Definition 4**, respectively.

Definition 6. If $\mathcal{A} \in \mathbb{R}^{m\times n\times k}$, then the tensor transpose, denoted $transpose_L(\mathcal{A}) \in \mathbb{R}^{m\times k\times n}$, is defined by taking matrix transpose of each frontal slice of $\mathcal{A}$ in the transform domain as following:

$$\begin{aligned}
&\mathcal{B} = transpose_L(\mathcal{A}),\\
&\tilde{\mathcal{A}} = \mathcal{A} \times_1 \mathbf{L},\\
&\tilde{\mathcal{B}}^{(i)} = (\tilde{\mathcal{A}}^{(i)})^T \text{ for } i = 1, \ldots, m,\\
&\mathcal{B} = \tilde{\mathcal{B}} \times_1 \mathbf{L}^{-1},
\end{aligned}$$

where "T" denotes matrix transpose.

2.3. *The Proposed Method*

Our proposed method, DyEPAD (Fig. 2), employs GCN layers to extract node (patient) embeddings from graph-structured EHR data. Each time step (visit) is trained in static network manner, meaning that the GNN parameters are updated independently based on a specific loss function for each time step. By doing so, the complexities of training an RNN unit are avoided. Furthermore, our model incorporates advanced designs, such as dropout, batch normalization, and mini- batch, which are present in static GNN-based learning methods.[32–34] To capture the evolutionary patterns of the patient embeddings across all time points in EHR, these embeddings are subsequently subjected to tensor algebraic operations for frequency domain analysis.

2.4. *Graph Convolutional Networks and Embedding Aggregation*

In DyEPAD, we construct a patient similarity network for each clinical visit. In this network, nodes represent patients, and edges encode the similarities between patients based on their EHR for that specific visit. Additionally, their EHR are also assigned as node features. For

each clinical visit at time t, DyEPAD utilizes a GCN module to learn node embeddings as follows:

$$H_t = \sigma(D_t^{-1/2} A_t D_t^{-1/2} X_t W_t), \tag{1}$$

for $t = 1, 2, ..., n$, where n is the total number of graphs (visits), and $X_t \in \mathbb{R}^{m \times d}$ is the feature matrix of nodes (m is the number of nodes and d is the feature size). $A_t \in \mathbb{R}^{m \times m}$ and $D_t \in \mathbb{R}^{m \times m}$ are the adjacency and the node degree matrices, respectively. $W_t \in \mathbb{R}^{d \times \ell}$ is the learnable weight matrix, σ is the activation function, and $H_t \in \mathbb{R}^{m \times \ell}$ is the node embeddings matrix. It is important to note that the hidden layer size ℓ is determined by the column size of W_t. The adjacency matrix $A_t \in \mathbb{R}^{m \times m}$ was constructed using k-nearest neighbors (k was set to 5) based on cosine similarities between patients' EHR.

GCN layers of DyEPAD capture embeddings in a given graph-structured EHR at time step (visit) t. To update the embeddings learned by the GCN layer based on the embeddings

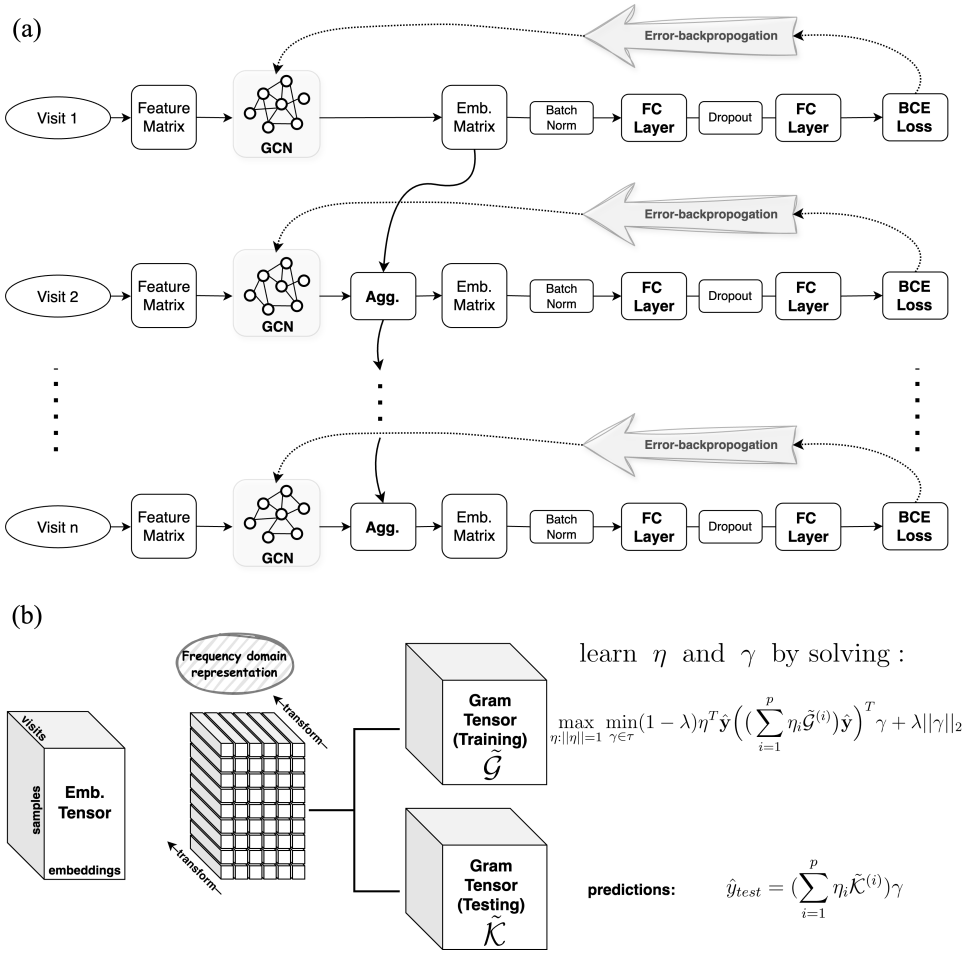

Fig. 2. Graphical illustration of DyEPAD. (a) DyEPAD utilizes GCN layers to derive node embeddings from graph-structured EHR data, and subsequently employs aggregation layers (**Agg.**) to aggregate the current embedding with those from the most recent previous visit. (b) The derived embeddings are then processed through tensor algebraic operations for frequency domain analysis, capturing the complete range of evolutionary patterns.

of the most recent previous visit, DyEPAD employs an aggregation function. This function makes the model dynamic by allowing it to adapt and update based on the most recent visit, thus effectively capturing and reflecting the evolving nature of the data over time. In traditional RNN architectures, Gated Recurrent Units (GRU) is used to process sequential data by updating hidden states across all time steps. In our work, GRU was used in a non-traditional way. Instead of processing sequences as part of an RNN, we applied GRU to aggregate the current embedding with those from the most recent previous visit as follows:

$$\begin{aligned} R_t &= sigmoid(H_t W_{ir} + \bar{H}_{t-1} W_{hr}), \\ U_t &= sigmoid(H_t W_{iz} + \bar{H}_{t-1} W_{hz}), \\ N_t &= tanh\big(H_t W_{in} + R_t \otimes (\bar{H}_{t-1} W_{hn})\big), \\ \bar{H}_t &= \big((1 - U_t) \otimes N_t\big) + \big(U_t \otimes \bar{H}_{t-1}\big), \end{aligned} \tag{2}$$

where $\otimes$ denotes element-wise multiplication, $R_t \in \mathbb{R}^{m \times \ell}$ is the reset gate, $U_t \in \mathbb{R}^{m \times \ell}$ is the update gate, and $N_t \in \mathbb{R}^{m \times \ell}$ is the new state matrices for a given time step t. W_{ir} and W_{hr} are the parameters for the reset gate. W_{iz} and W_{hz} are the parameters for the update gate. W_{in} and W_{hn} are the parameters for the new state. $\bar{H}_t \in \mathbb{R}^{m \times \ell}$ is the updated embeddings for given input embeddings $H_t \in \mathbb{R}^{m \times \ell}$ (current state embeddings) and $\bar{H}_{t-1} \in \mathbb{R}^{m \times \ell}$ (previous updated state embeddings).

We then employ two fully connected layers as follows:

$$p = sig\big(\sigma(\bar{H}_t \bar{W}_1)\bar{W}_2\big), \tag{3}$$

where σ is the activation function for the first layer, and sig denotes the sigmoid activation function.

To learn model parameters, we use binary cross entropy (BCE) loss function for each time point. The loss function for a single prediction can be defined as:

$$Loss = -\big(ylog(p) + (1 - y)log(1 - p)\big), \tag{4}$$

where y is the ground truth binary label (0 denotes MCI, and 1 represents AD labels), and p is the predicted probability. Adam optimization[35] is used as the state-of-the-art for stochastic gradient descent algorithm.

2.5. *Spatiotemporal Tensor Representation of Embeddings*

The patient embeddings learned by GCN and GRU in the previous step can be structured as a spatiotemporal tensor, where the dimensions correspond to time, patients, and patient embeddings as shown in Fig. 2(b). This spatiotemporal tensor preserves the intrinsic correlations present in the data while enabling to capture complex patterns across multiple dimensions.

$$\mathcal{H}^{(t)} = \bar{H}_t \;\text{ for }\; t = 1, 2, \ldots, n. \tag{5}$$

$\mathcal{H} \in \mathbb{R}^{n \times m \times \ell}$ is a spatiotemporal tensor where each frontal slice ($\mathcal{H}^{(t)}$) is the patient embedding matrix at time t (see Eq. (2)). Since there is no aggregation unit at time step 1 (Fig. 2(a)),

$\bar{H}_1$ is equal to H_1 (see Eq. (1)). To process the spatiotemporal tensor, we utilize gram tensors as described in **Theorem 1** as follow:

Theorem 1. *Let $\mathcal{H}_{(i)} \in \mathbb{R}^{n\times 1\times \ell}$ and $\mathcal{H}_{(j)} \in \mathbb{R}^{n\times 1\times \ell}$ be horizontal slices of the spatiotemporal tensor $\mathcal{H} \in \mathbb{R}^{n\times m\times \ell}$ (Eq. (5)). The gram tensor $\mathcal{G} \in \mathbb{R}^{n\times m\times m}$ for third-order tensors is constructed as follows:*

$$\mathbf{k}\big(\mathcal{H}_{(i)}, \mathcal{H}_{(j)}\big) = \big(\mathcal{H}_{(i)} \circ_L transpose_L(\mathcal{H}_{(j)})\big)^d,$$
$$\vec{\mathbf{g}}_i^j = \mathbf{k}\big(\mathcal{H}_{(i)}, \mathcal{H}_{(j)}\big).$$

*The kernel function applied to $\mathcal{H}_{(i)}$ and $\mathcal{H}_{(j)}$, denoted as $\mathbf{k}\big(\mathcal{H}_{(i)}, \mathcal{H}_{(j)}\big)$, results in $\vec{\mathbf{g}}_i^j$ which is a tubal scalar of $\mathcal{G}$ defined in **Definition 1**. This tensorial polynomial function was built upon on the inner product of two horizontal slices. We used the tensor-tensor multiplication (**Definition 5**) and the tensor transpose operation (**Definition 6**).*

Proof. As each frontal slice of the gram tensor is a kernel matrix in the transform domain.

$$\tilde{\mathcal{G}} = \mathcal{G} \times_1 \mathbf{L} \text{ (transform domain representation)},$$

It has been demonstrated that the quadratic form $\alpha^T \tilde{\mathcal{G}}^{(t)} \alpha$ is non-negative for all vectors $\alpha \in \mathbb{R}^m$.[36,37] As this implies that the frontal slices $(\tilde{\mathcal{G}}^{(t)})$ are positive semi-definite in the transform domain, $\tilde{\mathcal{G}}$ is a collection of positive-definite gram matrices. □

Each horizontal slice of a spatiotemporal tensor represents a patient. Each tubal scalars of a spatiotemporal tensor provides a sequence of embeddings over time. In matrix algebra, inner product of two vector gives a scalar. Each sample (horizontal slice) of the spatiotemporal tensor can be vectorized and a downstream task can be applied using matrix algebra. However, vectorizing samples destroys the spatial and temporal correlation within each sample. In our case, inner product of two horizontal slices provides a tubal scalar, as shown in **Theorem 1**. By doing so, we keep all the spatial and temporal correlation within each sample while computing the inner product between samples.

To learn non-linear patterns by implicitly mapping data into a higher-dimensional space, we need to construct gram tensors. Let $\mathcal{G}_{(1)}, \mathcal{G}_{(2)}, \cdots, \mathcal{G}_{(q)}$ be selected q horizontal slices of the spatiotemporal tensor $\mathcal{H}$ (Eq. (5)) for training. Similarly, let $\mathcal{K}_{(1)}, \mathcal{K}_{(2)}, \cdots, \mathcal{K}_{(w)}$ be selected w horizontal slices of the spatiotemporal tensor $\mathcal{H}$ for testing. Gram tensors for training, $\mathcal{G} \in \mathbb{R}^{n\times q\times q}$, and for testing, $\mathcal{K} \in \mathbb{R}^{n\times w\times q}$, can be constructed as follows:

$$\mathcal{G} = \begin{bmatrix} \mathbf{k}(\mathcal{G}_{(1)}, \mathcal{G}_{(1)}) & \mathbf{k}(\mathcal{G}_{(2)}, \mathcal{G}_{(1)}) & \cdots \mathbf{k}(\mathcal{G}_{(q)}, \mathcal{G}_{(1)}) \\ \mathbf{k}(\mathcal{G}_{(1)}, \mathcal{G}_{(2)}) & \mathbf{k}(\mathcal{G}_{(2)}, \mathcal{G}_{(2)}) & \cdots \mathbf{k}(\mathcal{G}_{(q)}, \mathcal{G}_{(2)}) \\ \vdots & \vdots & \vdots \\ \mathbf{k}(\mathcal{G}_{(1)}, \mathcal{G}_{(q)}) & \mathbf{k}(\mathcal{G}_{(2)}, \mathcal{G}_{(q)}) & \cdots \mathbf{k}(\mathcal{G}_{(q)}, \mathcal{G}_{(q)}) \end{bmatrix},$$

$$\mathcal{K} = \begin{bmatrix} \mathbf{k}(\mathcal{G}_{(1)}, \mathcal{K}_{(1)}) & \mathbf{k}(\mathcal{G}_{(2)}, \mathcal{K}_{(1)}) & \cdots \mathbf{k}(\mathcal{G}_{(q)}, \mathcal{K}_{(1)}) \\ \mathbf{k}(\mathcal{G}_{(1)}, \mathcal{K}_{(2)}) & \mathbf{k}(\mathcal{G}_{(2)}, \mathcal{K}_{(2)}) & \cdots \mathbf{k}(\mathcal{G}_{(q)}, \mathcal{K}_{(2)}) \\ \vdots & \vdots & \vdots \\ \mathbf{k}(\mathcal{G}_{(1)}, \mathcal{K}_{(w)}) & \mathbf{k}(\mathcal{G}_{(2)}, \mathcal{K}_{(w)}) & \cdots \mathbf{k}(\mathcal{G}_{(q)}, \mathcal{K}_{(w)}) \end{bmatrix}.$$

As outlined in **Theorem 1**, each frontal slice of a gram tensor is a kernel matrix in the transform domain. Therefore, $\tilde{\mathcal{G}}$ and $\tilde{\mathcal{K}}$ represent the transform domain representation of the gram tensors $\mathcal{G}$ and $\mathcal{K}$, respectively. To perform a classification task, we first need to integrate the multiple kernel matrices (frontal slices) in the gram tensor. To this end, we utilized Easy Multiple Kernel (EasyMKL) learning algorithm.[38] EasyMKL employs a minimization–maximization learning criterion to determine the optimal weighting of multiple kernel matrices, thereby maximizing the margin between two classes and improving classification performance. We want to find the best parameter combinations for the frontal slices of the gram tensor $\tilde{\mathcal{G}}$ by solving:

$$\max_{\eta:||\eta||=1} \min_{\gamma\in\tau} (1-\lambda)\eta^T y\Big((\sum_{i=1}^{n} \eta_i \tilde{\mathcal{G}}^{(i)})y\Big)^T \gamma + \lambda||\gamma||_2, \tag{6}$$

where y represents the target labels corresponding to visit $n+1$. We note that the spatiotemporal and gram tensors need to be constructed based on the first n visits. To learn the learnable parameters in Eq. (6), the target labels of the training set at visit $n+1$ were utilized along with the gram tensor $\tilde{\mathcal{G}}$ constructed for training. η is a learnable vector used to weight the linear combination of the kernel matrices (frontal slices $\tilde{\mathcal{G}}^{(i)}$). η is constrained to lie on the unit sphere. Mathematically, this is expressed as $||\eta|| = 1$. γ is another learnable parameter that is adjusted to minimize the objective function for a given η. The values of γ need to be optimized within a constrain set τ. λ $(0 \leq \lambda \leq 1)$ represents a regularization term that penalizes the magnitude of γ.

As shown in **Algorithm 1**, to solve the min-max problem, we decomposed the problem into two stages: first, we addressed the inner minimization problem over γ, and and then we solved the outer maximization problem over η. We employed convex optimization techniques to solve the inner maximization problem, utilizing the solvers available in the CVXPY library for Python. The outer maximization problem was solved using gradient ascent algorithms. We updated η iteratively (**Algorithm 1 line:8**) while ensuring it satisfies the unit norm constraint (**Algorithm 1 line:9**).

After learning the learnable parameters η and γ, the predictions can be computed for the test set $\tilde{\mathcal{K}}$ as following:

$$\hat{y} = (\sum_{i=1}^{n} \eta_i \tilde{\mathcal{K}}^{(i)})\gamma,$$
$$p_j = \frac{1}{1 + exp(\hat{y}_j)},$$

where $\hat{y}$ is the vector of raw prediction scores, and p_j is the probability of the raw score $\hat{y}_j$.

Algorithm 1 Min-Max Optimization Algorithm

Input: Initial vector η, regularization parameter λ, vector y, matrices $\{\tilde{\mathcal{G}}^{(i)}\}_{i=1}^{n}$, constraint set τ, step size α, tolerance ϵ
Output: Optimized vectors η and γ
Normalize $\eta \leftarrow \frac{\eta}{\|\eta\|}$
while true **do**
  $\eta_{prev} = \eta$
  Compute $\gamma(\eta)$ by solving

$$\gamma(\eta) = \arg\min_{\gamma \in \tau} \left[(1-\lambda)\eta^T y \left(\left(\sum_{i=1}^{n} \eta_i \tilde{\mathcal{G}}^{(i)} \right) y \right)^T \gamma + \lambda \|\gamma\|_2 \right]$$

  Calculate gradient:

$$\nabla_\eta J(\eta) = (1-\lambda)\nabla_\eta \left[\eta^T y \left(\left(\sum_{i=1}^{n} \eta_i \tilde{\mathcal{G}}^{(i)} \right) y \right)^T \gamma(\eta) \right]$$

  Update η:

$$\eta \leftarrow \eta + \alpha \nabla_\eta J(\eta)$$

  Project onto unit sphere:

$$\eta \leftarrow \frac{\eta}{\|\eta\|}$$

  if $\|\eta - \eta_{prev}\| < \epsilon$ **then**
    Break
  end if
end while
Return: η and γ

3. Results

3.1. *Datasets*

In this study, we utilized longitudinal data from two large AD databases: the Alzheimer's Disease Neuroimaging Initiative (ADNI) and the National Alzheimer's Coordinating Center (NACC) database.

The ADNI (adni.loni.usc.edu) was launched in 2003 as a public–private partnership, led by Principal Investigator Michael W. Weiner, MD. The primary goal of ADNI has been to test whether serial MRI, PET, other biological markers, and clinical and neuropsychological assessment can be combined to measure the progression of MCI and early AD. Since it has been launched, the public–private cooperation has contributed to significant achievements in AD research by sharing data to researchers from all around the world. The NACC, a comprehensive repository of data from several research sites across the United States, was specifically designed to aid research focused on understanding, diagnosing, and treating AD.

ADNI and NACC databases include data from cognitive performance tests, MRI scans, CSF analysis, demographic information, and diagnostic labels. However, only longitudinal

data were considered in this study, including cognitive performance tests, MRI scans, CSF analysis, and diagnostic labels that are AD and MCI. We preprocessed the data following the steps in PPAD method .[14] Missing values were imputed using k-NN algorithm, where we used average values from the nearest k neighbors with the same diagnosis (i.e., MCI or AD), employing the Euclidean as the distance metric and setting k to 5. To maintain data quality, we removed visits and features with $\geq 40\%$ and $\geq 60\%$ missing rate, respectively.

For the ADNI dataset, the original dataset comprised 15,087 records for 2,288 distinct patients, with each record representing a patient visit with 115 features. After preprocessing, the dataset had 20 longitudinal features for 1,169 patients across 5,759 visits, derived from cognitive performance tests and MRI scans. CSF analysis features were excluded due to significant missing data. For the NACC dataset, the original dataset comprised 172,026 records (i.e., visits) with 1024 features for 46,513 distinct patients. After preprocessing, the dataset had 5 longitudinal features for 8,121 patients across 35,423 visits, derived from cognitive performance tests. MRI scans and CSF analysis features were excluded due to significant missing data. Finally, we focused on patients with at least seven visits for training and model evaluation. This resulted in a final ADNI dataset of 20 longitudinal features for 250 patients and a final NACC dataset with 5 longitudinal features for 1,414 patients.

3.2. *Next Visit Prediction*

We trained DyEPAD on longitudinal EHR data to predict conversion of MCI patients to AD at the next visit. The experiments were performed on ADNI and NACC datasets separately and the results were compared with the state-of-the-art methods (i.e., GCN, GAT, T-LSTM, PPAD, and TA-RNN) as well as baseline methods, namely Random Forest (RF) and Support Vector Machine (SVM). For both datasets, the first six visits were considered to train the models. We measured the performance of all methods based on the visit number 7. Since RF and SVM cannot handle longitudinal data, we stacked all six visits' feature matrices to train these models. For GNN-based methods, the adjacency matrices were constructed using k-NN method (see **Section 2.4**). We evaluated all the methods on ten different randomly generated training and test splits. The number of patients selected for training was 80% of the total number of patients, while 20% were used for testing.

Table 1 demonstrates that our proposed approach DyEPAD outperformed all the state-of-the-art and baseline methods for both datasets for all three evaluation metrics. As outlined in **Section 2.1** and **2.2**, any invertible linear transformation can be used for our tensor operations. In our experiments, we used the discrete Fourier transform (DFT)[39] and discrete Hartley transform (DHT).[40] Although, both transformations resulted similar predictive performance, compared to the DFT, the DHT has the advantage of converting real functions into real functions, without the need for complex numbers. Therefore, running DyEPAD with DHT is more computationally efficient than running DyEPAD with DFT.[40]

3.3. *Multiple Visits Ahead Prediction*

In this subsection, we evaluated how well our proposed DyEPAD model performed at predicting conversion to AD in multiple visits ahead. We compared our results to TA-RNN and

Table 1. The reported values on ADNI and NACC datasets represent the averages along with standard deviations, based on ten runs, for three performance measures, namely: Accuracy, Macro F1 and Area Under the ROC Curve (AUCROC). Best values are shown in bold.

Dataset	Method	Accuracy	Macro F1	AUROC
ADNI	SVM	0.594 ± 0.040	0.570 ± 0.045	0.572 ± 0.045
	RF	0.566 ± 0.048	0.550 ± 0.047	0.550 ± 0.046
	GCN	0.661 ± 0.032	0.642 ± 0.033	0.642 ± 0.032
	GAT	0.685 ± 0.044	0.659 ± 0.035	0.660 ± 0.037
	PPAD	0.896 ± 0.035	0.893 ± 0.036	0.895 ± 0.034
	TA-RNN	0.883 ± 0.043	0.880 ± 0.043	0.885 ± 0.040
	T-LSTM	0.819 ± 0.145	0.778 ± 0.212	0.806 ± 0.157
	DyEPAD (DFT)	$\mathbf{0.900 \pm 0.035}$	$\mathbf{0.895 \pm 0.035}$	$\mathbf{0.896 \pm 0.035}$
	DyEPAD (DHT)	0.898 ± 0.038	0.893 ± 0.035	0.894 ± 0.038
NACC	SVM	0.773 ± 0.032	0.710 ± 0.032	0.690 ± 0.037
	RF	0.754 ± 0.030	0.688 ± 0.033	0.674 ± 0.036
	GCN	0.797 ± 0.032	0.742 ± 0.036	0.723 ± 0.044
	GAT	0.789 ± 0.035	0.735 ± 0.037	0.717 ± 0.042
	PPAD	0.950 ± 0.010	0.892 ± 0.023	0.878 ± 0.033
	TA-RNN	0.944 ± 0.011	0.880 ± 0.024	0.867 ± 0.031
	T-LSTM	0.935 ± 0.033	0.824 ± 0.153	0.812 ± 0.135
	DyEPAD (DFT)	0.950 ± 0.009	0.901 ± 0.014	0.900 ± 0.020
	DyEPAD (DHT)	$\mathbf{0.952 \pm 0.007}$	$\mathbf{0.902 \pm 0.014}$	$\mathbf{0.905 \pm 0.018}$

PPAD only, as the other methods are not designed to predict multiple visits ahead. All these methods were trained using the first three, four, and five visits of ADNI and NACC datasets and evaluated the performance on the seventh visit. The results in Figure 3 illustrate that DyEPAD performs comparably to the top state-of-the-art methods. In DyEPAD, tensorial functions operate via a linear transform to capture evolutionary characteristic in the data. However, applying a transform to a very short discrete signal may fail to capture periodic components, trends, and other evolutionary characteristics in the data. Therefore, not surprisingly, we observed that the performance of DyEPAD increased as the interval between the visits decreased. While this finding highlights the potential of DyEPAD, it also underscores a limitation that the model's effectiveness may be constrained by the granularity of the input data. In cases where patient visits are infrequent, the model may struggle to capture the dynamic nature of the underlying processes, potentially affecting its predictive accuracy.

3.4. *Ablation Study*

To assess the impact of deactivating various components of the proposed architecture on the model's performance, we conducted an ablation study. Specifically, to examine the impact of the tensorial functions and the aggregation layers, we compared the performance of the proposed DyEPAD architecture with two variants of DyEPAD: 1) we used an identity transformation, instead of the DHT; 2) we disabled the GRU layers and the embeddings were computed based on the GCN layer at that time point. We conducted the experiments on both ADNI and NACC datasets according to the experimental settings outlined in **Section 3.2**. The results given in Table 2 show that both the frequency domain representation and aggregation functions were crucial for capturing full scope of evolutionary patterns, as demonstrated by

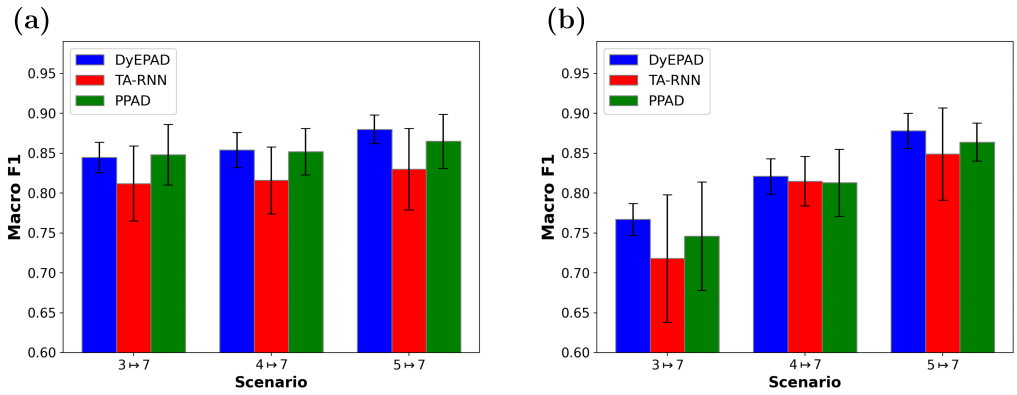

Fig. 3. Comparison of the performance of DyEPAD for different scenarios on (a) ADNI and (b) NACC datasets. Scenario $a \mapsto b$ denotes that the first a consecutive visits were utilized to train the model and bth visit was used for testing.

the superior performance of the original DyEPAD architecture for both datasets. We observed that for both ADNI and NACC datasets, the effect of the transform domain representation on the predictive performance was higher than the effect of the aggregation component. An identity transformation maps any element to itself. This means that we cannot capture any frequency components or patterns within the data because no change is applied to transform the data into the frequency domain or any other domain that might highlight such features. We can still used tensor operators, however the data remains in its original state, preserving its initial structure and values without revealing underlying periodicity or frequency information that a transform like the DHT would provide. This suggests that the considering a feasible invertible linear transformation is crucial in DyEPAD.

Table 2. The average Macro F1 and AUROC scores of different variants of DyEPAD on ADNI and NACC datasets over ten runs. Best values are shown in bold.

Variants of DyEPAD	**ADNI**		**NACC**	
	Macro F1	AUROC	Macro F1	AUROC
DyEPAD without any transformation	0.651 ± 0.071	0.654 ± 0.072	0.691 ± 0.019	0.718 ± 0.024
DyEPAD (DHT) without GRU	0.757 ± 0.074	0.754 ± 0.073	0.872 ± 0.028	0.868 ± 0.033
Proposed DyEPAD (DHT) architecture	$\mathbf{0.893 \pm 0.035}$	$\mathbf{0.894 \pm 0.038}$	$\mathbf{0.902 \pm 0.014}$	$\mathbf{0.905 \pm 0.018}$

4. Conclusions and Future Work

This paper presents a novel approach for predicting the progression of MCI subjects to AD using longitudinal EHR. Our proposed method, DyEPAD, captures latent space representations of EHR at each time step by utilizing GCN and GRU layers. We also use tensor algebraic operations for frequency domain analysis of these embeddings, capturing the complete range of evolutionary patterns across all time steps. The experimental outcomes reveal a notable superiority of DyEPAD over both state-of-the-art and baseline methods for most cases. Future work will aim to assess DyEPAD's performance on additional longitudinal biomedical datasets and examine the impact of different transformations and aggregation functions on its performance.

References

1. H. Hampel, D. Prvulovic, S. Teipel, F. Jessen, C. Luckhaus, L. Frölich, M. W. Riepe, R. Dodel, T. Leyhe, L. Bertram *et al.*, The future of alzheimer's disease: the next 10 years, *Progress in neurobiology* **95**, 718 (2011).
2. R. Cui, M. Liu, A. D. N. Initiative *et al.*, Rnn-based longitudinal analysis for diagnosis of alzheimer's disease, *Computerized Medical Imaging and Graphics* **73**, 1 (2019).
3. X. Wang, J. Qi, Y. Yang and P. Yang, A survey of disease progression modeling techniques for alzheimer's diseases, in *2019 IEEE 17th International Conference on Industrial Informatics*, (vol.1, pp. 1237–1242).
4. K. M. Langa and D. A. Levine, The diagnosis and management of mild cognitive impairment: a clinical review, *Jama* **312**, 2551 (2014).
5. X. W. Gao, R. Hui and Z. Tian, Classification of ct brain images based on deep learning networks, *Computer methods and programs in biomedicine* **138**, 49 (2017).
6. H. Li, M. Habes, D. A. Wolk, Y. Fan, A. D. N. Initiative *et al.*, A deep learning model for early prediction of alzheimer's disease dementia based on hippocampal magnetic resonance imaging data, *Alzheimer's & Dementia* **15**, 1059 (2019).
7. J. Guo, W. Qiu, X. Li, X. Zhao, N. Guo and Q. Li, Predicting alzheimer's disease by hierarchical graph convolution from positron emission tomography imaging, in *2019 IEEE international conference on big data (big data)*, (pp. 5359–5363, 2019).
8. M. Nguyen, T. He, L. An, D. C. Alexander, J. Feng, B. T. Yeo, A. D. N. Initiative *et al.*, Predicting alzheimer's disease progression using deep recurrent neural networks, *NeuroImage* **222**, p. 117203 (2020).
9. B. Shickel, P. J. Tighe, A. Bihorac and P. Rashidi, Deep ehr: a survey of recent advances in deep learning techniques for electronic health record (ehr) analysis, *IEEE journal of biomedical and health informatics* **22**, 1589 (2017).
10. S. S. Prabhu, J. A. Berkebile, N. Rajagopalan, R. Yao, W. Shi, F. Giuste, Y. Zhong, J. Sun and M. D. Wang, Multi-modal deep learning models for alzheimer's disease prediction using mri and ehr, in *2022 IEEE 22nd International Conference on Bioinformatics and Bioengineering (BIBE)*, (pp. 168–173, 2022).
11. M. Tanveer, B. Richhariya, R. U. Khan, A. H. Rashid, P. Khanna, M. Prasad and C.-T. Lin, Machine learning techniques for the diagnosis of alzheimer's disease: A review, *ACM Transactions on Multimedia Computing, Communications, and Applications (TOMM)* **16**, 1 (2020).
12. P. Yadav, M. Steinbach, V. Kumar and G. Simon, Mining electronic health records (ehrs) a survey, *ACM Computing Surveys (CSUR)* **50**, 1 (2018).
13. I. M. Baytas, C. Xiao, X. Zhang, F. Wang, A. K. Jain and J. Zhou, Patient subtyping via time-aware lstm networks, in *Proceedings of the 23rd ACM SIGKDD international conference on knowledge discovery and data mining*, (pp. 65–74, 2017).
14. M. Al Olaimat, J. Martinez, F. Saeed, S. Bozdag and A. D. N. Initiative, Ppad: A deep learning architecture to predict progression of alzheimer's disease, *Bioinformatics* **39**, i149 (2023).
15. M. Al Olaimat, S. Bozdag and A. D. N. Initiative, Ta-rnn: an attention-based time-aware recurrent neural network architecture for electronic health records, *Bioinformatics* **40**, i169 (2024).
16. T. N. Kipf and M. Welling, Semi-supervised classification with graph convolutional networks, *arXiv preprint arXiv:1609.02907* (2016).
17. P. Veličković, G. Cucurull, A. Casanova, A. Romero, P. Lio and Y. Bengio, Graph attention networks, *arXiv preprint arXiv:1710.10903* (2017).
18. Y. Zhang, X. He, Y. H. Chan, Q. Teng and J. C. Rajapakse, Multi-modal graph neural network for early diagnosis of alzheimer's disease from smri and pet scans, *Computers in Biology and Medicine* **164**, p. 107328 (2023).

19. Y. Zhu, J. Ma, C. Yuan and X. Zhu, Interpretable learning based dynamic graph convolutional networks for alzheimer's disease analysis, *Information Fusion* **77**, 53 (2022).
20. L. Jiang, H. Lin, Y. Chen, A. D. N. Initiative *et al.*, Sex difference in the association of apoe4 with cerebral glucose metabolism in older adults reporting significant memory concern, *Neuroscience Letters* **722**, p. 134824 (2020).
21. L. Besser, W. Kukull, D. S. Knopman, H. Chui, D. Galasko, S. Weintraub, G. Jicha, C. Carlsson, J. Burns, J. Quinn *et al.*, Version 3 of the national alzheimer's coordinating center's uniform data set, *Alzheimer Disease & Associated Disorders* **32**, 351 (2018).
22. T. G. Kolda and B. W. Bader, Tensor decompositions and applications, *SIAM review* **51**, 455 (2009).
23. K. Braman, Third-order tensors as linear operators on a space of matrices, *Linear Algebra and its Applications* **433**, 1241 (2010).
24. M. E. Kilmer, K. S. Braman, N. Hao and R. C. Hoover, Third order tensors as operators on matrices: A theoretical and computational framework with applications in imaging, *SIAM Journal on Matrix Analysis and Applications (SIMAX)* **34**, 148 (Feb. 2013).
25. C. Ozdemir, R. C. Hoover and K. Caudle, 2DTPCA: A new framework for multilinear principal component analysis, in *2021 IEEE International Conference on Image Processing (ICIP)*, (pp. 344-348, 2021).
26. C. Ozdemir, R. C. Hoover and K. Caudle, Fast tensor singular value decomposition using the low-resolution features of tensors, in *2021 20th IEEE International Conference on Machine Learning and Applications (ICMLA)*, (pp. 527–533, 2021).
27. C. Ozdemir, R. C. Hoover, K. Caudle and K. Braman, Kernelization of tensor discriminant analysis with application to image recognition, in *2022 21st IEEE International Conference on Machine Learning and Applications (ICMLA)*, (pp. 183–189, 2022).
28. C. Ozdemir, R. C. Hoover, K. Caudle and K. Braman, Tensor discriminant analysis on grassmann manifold with application to video based human action recognition, *International Journal of Machine Learning and Cybernetics* , 1 (2024).
29. R. N. Bracewell, The fourier transform, *Scientific American* **260**, 86 (1989).
30. N. Hao, M. E. Kilmer, K. Braman and R. C. Hoover, Facial recognition using tensor-tensor decompositions, *SIAM Journal on Imaging Sciences* **6**, 437 (2013).
31. E. Kernfeld, M. Kilmer and S. Aeron, Tensor–tensor products with invertible linear transforms, *Linear Algebra and its Applications* **485**, 545 (2015).
32. G. Li, M. Müller and G. Qian, Itzel carolina delgadillo perez, abdulellah abualshour, ali kassem thabet, and bernard ghanem. deepgcns: Making gcns go as deep as cnns, *IEEE transactions on pattern analysis and machine intelligence* **6**, p. 10 (2021).
33. J. You, Z. Ying and J. Leskovec, Design space for graph neural networks, *Advances in Neural Information Processing Systems* **33**, 17009 (2020).
34. C. Ozdemir, M. A. Olaimat, Y. Vashishath, S. Bozdag and A. D. N. Initiative, IGCN: Integrative graph convolutional networks for multi-modal data, *arXiv preprint arXiv:2401.17612* (2024).
35. D. P. Kingma and J. Ba, Adam: A method for stochastic optimization, *arXiv preprint arXiv:1412.6980* (2014).
36. V. N. Vapnik, V. Vapnik *et al.*, Statistical learning theory, *wiley New York* (1998).
37. T. Hofmann, B. Schölkopf and A. J. Smola, Kernel methods in machine learning, *The Annals of Statistics* (2008).
38. F. Aiolli and M. Donini, Easymkl: a scalable multiple kernel learning algorithm, *Neurocomputing* **169**, 215 (2015).
39. P. Duhamel and M. Vetterli, Fast fourier transforms: a tutorial review and a state of the art, *Signal processing* **19**, 259 (1990).
40. R. N. Bracewell, Discrete hartley transform, *JOSA* **73**, 1832 (1983).