DECODING BRAINS BY PAYING ATTENTION: AN ATTENTION-BASED FMRI TASK STATE DECODING DEEP NETWORK ARCHITECTURE

BY

FRANCIS ROXAS

THESIS

Submitted in partial fulfillment of the requirements
for the degree of Master of Science in Electrical and Computer Engineering
in the Graduate College of the
University of Illinois Urbana-Champaign, 2021

Urbana, Illinois

Adviser:

Assistant Professor Oluwasanmi Koyejo

ABSTRACT

One of the core goals in the field of cognitive neuroscience is to decode task state fMRI data. Task decoding is the process of taking neuroimaging data and determining the task that was performed when that data was collected. A large volume of work used for task decoding is done in pursuit of creating a deep learning model for task prediction. Typically these models will include either handcrafted features or data driven approaches for downscaling the input features in successive layers. In this thesis, we explore and compare the effectiveness of linear, graph-based and attention-based methods for hierarchical classification. Furthermore, we propose a new attention-based network architecture which showcases superior performance to all of our baseline architectures without the use of handcrafted features on several neuroimaging datasets.

To my parents, for their love and support.

ACKNOWLEDGMENTS

I would like to acknowledge Professor Sanmi Koyejo for the opportunity to work in his group and for his mentorship. Over the past year, we have spoken multiple times about motivation for which I am very grateful.

I would also like to show my sincere appreciation to Dr. Marcel Nassar. I was consistently in awe of his intellect and interesting insights which motivated many of the directions taken in this work.

Furthermore, I would like to extend my thanks to Katherine Tsai for providing the code foundation of her prior work for the graphical models and data collection. I would also like express my gratitude for the advice she has given me over the past year.

Finally, I would like to thank all of the friends I have made here at UIUC over the years who have consistently pushed me to be better and supported me in my endeavors. I know I have said thanks a seemingly countless number of times, but I always mean it.

CONTENTS

Chapter 1

INTRODUCTION

The amount and complexity of data collected for neuroscientific imaging has seen remarkable growth in recent years [1], [2]. Much of this growth can be attributed to the popularity of functional magnetic resonance imaging (fMRI) which is a non-invasive, cost-effective and widely available brain imaging technique that allows neuroscientists to quantitatively measure signals throughout the brain corresponding to sensory stimuli [3]. This area of research is typically referred to as fMRI brain state decoding.

In recent years, there has been increasing interest to in decoding brain state information using a plethora of deep learning network structures. Past works in this area used standard deep learning models for medical analysis such as fully connected models and convolutional neural networks (CNNs) [4]. However, recent works suggest the need for powerful network-based architectures that can create and utilize capture complex interactions in neurobiological systems [2] such as graph neural networks (GNNs) [5] and transformers [6]. Furthermore, accompanying neuroscience literature suggests the importance of clustering [7] and hierarchical organization [8] in fMRI analysis.

These observations motivate our work. In this thesis, we go over and compare several hierarchical and end-to-end deep learning network structures for fMRI task state decoding. In particular, we leverage the insights of the fixed grouping layer [9], graph convolutional networks [5] and attention mechanisms [6] and propose an attention-based deep learning architecture that is well-suited to task state decoding. We will show that our architecture outperforms all of the aforementioned architectures on several common neuroimaging datasets. In the following chapters, we will explore related works in the field of fMRI task state decoding using GNNs and attention. Afterwards, we will lay out the baseline architectures and the compare the results of our experiments. In the final chapter, we will discuss our results and future directions for our work.

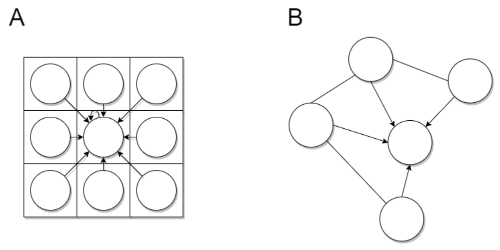

Figure 1.1: Comparison of CNNs and spatial GNNs: (A) In classical convolutional layers on images, the hidden representation of a central node (pixel) is based on a weighted combination of neighboring nodes which are defined by proximity to the center node. **(B)** In graphs, nodes are not arranged in a grid-like structure. So, neighbors are defined through edges rather than by proximity. Similarly to convolutional layers, the representation of the central node in a graph is updated based on an aggregation of its neighbors.

1.1 Graph Neural Networks

1.1.1 Graph Convolutional Networks

Standard deep neural network architectures have found great success in domains where the underlying representation of data can be modeled in an Euclidean or grid-like fashion [10]. However, in many domains, such as functional imaging analysis, we wish to find a non-Euclidean underlying structure representation for our data. An answer to this predicament comes about by modeling our data and the interactions amongst our data through graph structures and by extending standard deep learning frameworks to graph structures.

The task of creating a graph neural network has been tackled by a variety of different authors with some notable methods being [5], [11], and [12]. Typically, we can divide these approaches into spectral-based approaches and spatially-based approaches [11]. Spectral approaches utilize graph signal processing techniques to perform convolutions between an input graph signal and an input graph filter [13]. However, spectral approaches suffer when the number of input nodes in a graph is large due to the high computational cost of computing the graph Fourier transform [13]. On the other hand, spatial approaches borrow from traditional convolutional layers by updating the hidden representation of any given node through a function of that node itself and its neighbors (as seen in Figure 1.1). In recent years, spatial approaches have seen much more use than spectral-based methods due to the former's efficiency and generality [13].

In spatially-based approaches, we wish to find a function to update the features of a given graph G. The input to a graph convolutional model is a feature vector matrix $X = [x_0, x_1, ..., x_N] \in \mathbb{R}^{N \times D}$ where N is the number of input nodes to our model and D is the number of features of per node. These nodes are related by an adjacency matrix A which defines the set of edges E of the graph. In general, we can update the hidden representation $h_i^{(l)}$ of each node at layer l in the following manner:

$$h_i^{(l+1)} = \sigma \left(W \cdot h_i^{(l)} + \mathrm{AGGR}_{\mathbb{N}(i)}(h_q^{(l)}) \right) \tag{1.1}$$

where AGGR is an aggregation function of the neighbors of node i, $\mathbb{N}(i)$ is the neighborhood about node i, σ is a non-linear activation function, $h_i^{(0)} = x_i$, and W is a linear mapping from $\mathbb{R}^{D_i}$ to $\mathbb{R}^{D_o}$ for D_i input features and D_o output features. The aggregation function is usually taken to be a MAX or MEAN operation but can be any function of node i's neighbors, which are defined by A. One such aggregation function could be defined through the use of self-attention as found in graph attention models [14].

1.2 Linking GNNs to Transformers

Another deep learning architecture that has been gaining traction for use with non-Euclidean data is the transformer. At a glance, GNNs and transformers may not seem to have much in common. GNNs are typically used in tasks where one can infer links between input vectors such as tasks in social networks, neuroscience, and physics simulations [15]. These links are usually provided to the model through an input graph which relates the elements before updating the hidden state of each node. By contrast, in transformer models, the relationships between corresponding nodes are not passed as an input to a transformer layer, but rather learned through the use of an attention matrix. Despite this, in this section, we will motivate our use of transformers in functional brain analysis tasks by showing a connection

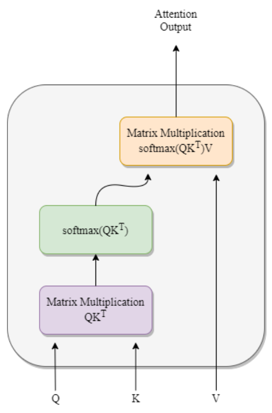
Attention Output
Matrix Multiplication softmax(QK^T)V
softmax(QK^T)
Matrix Multiplication QK^T
Q
K
V

Figure 1.2: Self-Attention Mechanism: A standard attentional mechanism which computes the attention output by applying a softmax function on the product of Q and K^T to interpolate the elements of V.

between GNN's and transformers.

Typically, transformer models are characterized by their use of a self-attention mechanism as depicted in Figure 1.2 [6]. These attention mechanisms take in three inputs: a query matrix Q, a key matrix K, and a value matrix V. Each of these inputs is formed by linearly projecting an input feature matrix X to the spaces of the query vectors, the key vectors, and the value vectors using $W_Q, W_K, W_V \in \mathbb{R}^{N \times d}$ respectively where N is the number of input nodes to the self-attention mechanism and d is the dimensionality of each input feature. So for input X, we can get Q, K and V as follows:

$$Q = W_Q \cdot X \tag{1.2}$$
$$K = W_K \cdot X \tag{1.3}$$
$$V = W_V \cdot X \tag{1.4}$$

Using these inputs we can compute the self-attention:

$$\text{Attention}(Q, K, V) = \text{softmax}\left(\frac{QK^T}{\sqrt{d}}\right) V \tag{1.5}$$

We see from Equation 1.5, that the each element of the output matrix is an interpolation of the value vector V through the use of softmax $\left(\frac{QK^T}{\sqrt{d}}\right)$. We can view the output of this softmax function as a matrix where the element at the ith column and jth row is a measure of similarity between input features x_i and x_j. So, for a given matrix of hidden representations $H^{(l)}$ at a given hidden layer l of our model we can write the update for the hidden representation of the nodes at the output of the transformer as follows:

$$h_i^{(l+1)} = \sum_{q \in I_X} \text{softmax}(W_Q^{(l)} h_i^{(l)} \cdot W_K^{(l)} h_q^{(l)}) \cdot \left(W_V^{(l)} h_q^{(l)} \right) \tag{1.6}$$

where $H^{(0)} = X$, $h_i^{(l)}$ is the ith column of $H^{(l)}$, and I_X is the index set of X.

We can now take a look back at a the propagation rule in Equation 1.1 for the hidden representations of nodes in a GNN to establish a connection between GNNs and transformers. We see that in a GNN, a given hidden node i is updated via an aggregation of nodes in the neighborhood of node i and then summed with a weighted version of node i. In a similar manner, in a transformer, the hidden representation of a given node $h^{(l)}$ at layer l is updated by an aggregation over all nodes as defined by the use of a similarity matrix. Due to this connection between GNNs and transformers, we decided to build transformer network architectures for fMRI task decoding and compare the results of those architectures with the results of GNN-based architectures.

Chapter 2

BACKGROUND

In recent years, fMRI analysis has been tackled using a variety of different deep learning approaches. A common type of approach is to create hierarchical models which attempt to exploit hierarchical organization of brain features through the use of handcrafted or learned clusters [8]. These models use these clusters to downsample their inputs by aggregating features together over multiple layers and obtain an output task prediction through the use of a fully connected layer [9]. Complementary to models using hand-engineered features are end-to-end models which take only raw fMRI signals as an input and return a task prediction as an output. Common examples of end-to-end models include convolutional and transformer-based architectures. In this chapter, we will continue this discussion by examining key insights of several network architectures which make use of handcrafted features and by examining key insights of several end-to-end network architectures that have been previously used in fMRI task state decoding. In addition, we will discuss our baseline architectures and discuss how prior works influence our baseline architectures.

2.1 Convolutional Neural Networks

Convolutional neural networks [4] tend to lend themselves well to problems where the input data contains strong spatial dependencies and, as a result, have been used by numerous authors for fMRI analysis [9]. In CNNs, the relations between input features are assumed to be contained in local windows defined by the size of the kernel of the convolutional layer. Within these local windows, weights are learned that allow the network to capture high-frequency information such as information held in textures and edges [16]. In recent years, multiple attempts have been made to determine the

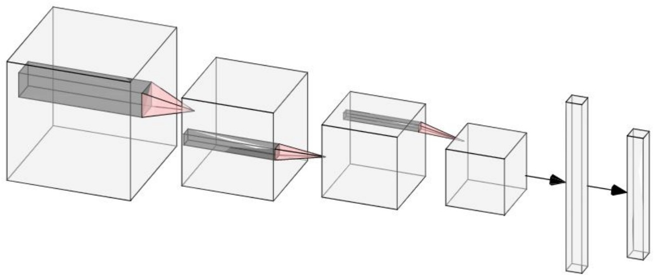

Figure 2.1: 3D CNN Diagram: An example of a 3D Convolutional Neural Network is shown here. The first four layers of the model downscale the 3D input volume. Afterwards, the output of the 3D convolutional layers is flattened and sent to two fully connected layers to obtain a final output prediction.

benefits of using CNNs for fMRI tasks. One such attempt is [17] in which 3D CNNs (depicted in Figure 2.1) have been shown to effectively utilize spatial relationships in the brain for task state inference. With this in mind, in several of our baseline architectures, we attempt to capture information typically found in spatially local regions of the brain and try to determine how beneficial that information is for task state prediction when paired with alternative network structures.

2.2 Fixed Grouping Layer

Another promising network architecture for fMRI analysis is based on the fixed grouping layer (FGL) which is a network layer proposed by Habeeb and Koyejo [9]. Given input $\mathbf{x}$ the output z to a given FGL l is as follows:

$$z = S[(\mathbf{x}v) \circ u] + b \tag{2.1}$$

where S is an assignment matrix formed through the use of handcrafted features, $v \in \mathbb{R}^{D_i \times D_o}$ is an input to output feature transform, $u \in \mathbb{R}^{N^{(l)} \times D_o}$ is node specific feature-wise weight parameter, $N^{(l)}$ is the number of input nodes to layer l, D_i is the number of input features to the layer, D_o is the number of output features of the layer and $b \in \mathbb{R}^{N^{(l+1)} \times D_o}$ is a bias term [9].

Upon stacking multiple FGLs together, Habeeb and Koyejo [9] have created a hierarchical network architecture as depicted in Figure 2.2. Through

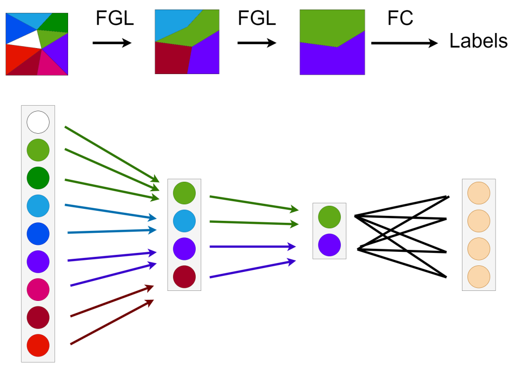

Figure 2.2: Fixed Grouping Layers: FGL Visualization recreated from [9]. The leftmost square on the top shows a segmentation with nine regions. Below are the input variables, each of which is corresponding to the regions of interest defined by that square before the first fixed grouping layer. After each layer of FGL, the segments in the top squares are grouped together and the number of nodes within each ROI is reduced in a hierarchical manner using an aggregation of the feature representation of nodes in the previous hidden layer of the model. After two fixed grouping layers, the output is sent to a fully connected layer to generate a prediction.

their results, they show that that their architecture outperforms common baseline architectures for brain state decoding [9]. The success of the FGL suggests that both the ability to simultaneously cluster feature vectors together through the use of a learned assignment matrix S and the use of a learnable parameter u to weight each feature vector before downsampling the hidden representation of the data are conducive to inference in hierarchical networks. Due to this success, we utilize FGL as a baseline for our work and incorporate the insights of FGL into our graphical baseline architectures.

2.3 Graph Neural Networks

Graph neural networks are perhaps the most influential network structure mentioned in this section thus far for fMRI analysis. This is due to the rising popularity, in recent years, of network-based deep learning models for analyzing neurobiological systems which typically contain complex interactions between a large number of elements [2]. As such, a large volume of research is dedicated to finding how to best determine and model these interactions in fMRI scans. Prior literature suggests that aggregation of information from several statistically connected brain regions in a hierarchical manner is conducive to fMRI task prediction [7], [8]. This is supported by works such as [18] and [19] which both propose hierarchical graph neural networks. These hierarchical graph networks take in fMRI images, form graphs between regions of interest through the use of statistical dependencies or temporal correlations between fMRI voxels, and demonstrate respectable performance against common baseline architectures for brain decoding [18], [19]. As a result, for our baseline hierarchical graph neural network, we attempted to leverage these insights by forming edges between nodes at each hidden layer through the use of statistical correlations.

2.4 Graph Attention Networks

Graph attention networks (GATs) are an extension of GNNs which utilize self-attention as proposed by Veličković et al. [14] to update node features. Specifically, GATs update the hidden representation $h_i^{(l)}$ of each node i at layer l using an aggregation of its neighbors based on an attention mechanism (visualized in Figure 2.3) as shown below [14]:

$$h_i^{(l+1)} = \sigma\left(\sum_{q \in \mathbb{N}_i} \alpha_{i,q} \mathbf{W} h_q^{(l)}\right) \tag{2.2}$$

where $h_i^{(0)} = x_i$, $\mathbf{W}$ is a linear transform that takes in an input in $\mathbb{R}^{D_i}$ and returns an output in $\mathbb{R}^{D_o}$, D_i is the number of input features per node, D_o is the number of output features per node, and $\alpha_{i,q}$ is an attention coefficient that determines the influence of nodes q's features on node i. The coefficient $\alpha_{i,q}$ does this by scoring the strength of the connection between two given nodes i and q as [14]:

$$\alpha_{i,q} = \text{softmax}(\text{LeakyReLU}(\mathbf{a}[\mathbf{W}h_i^{(l)} || \mathbf{W}h_q^{(l)}])) \tag{2.3}$$

where $||$ represents a concatenation operation along the feature dimension and $\mathbf{a}$ is a learnable parameter vector.

While their use in fMRI task decoding is seemingly limited, GATs have been shown to be beneficial to in a variety of graph classification tasks in

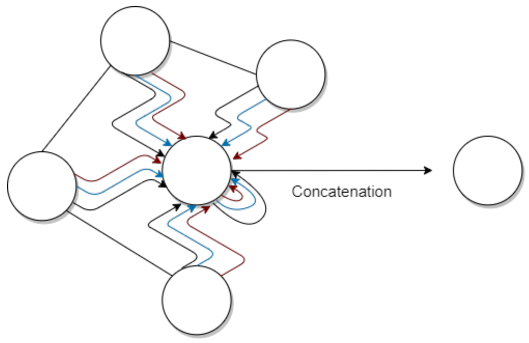

Figure 2.3: Graph Attention Mechanism: A diagram of multi-headed self-attention on graphs where the number of heads $K = 3$ based on multi-headed attention from [14] is shown here. The red, blue and black arrows denote differing attentional heads which each take a weighted representation of their respective node features. These attention heads are aggregated together using either an average or a concatenation (shown) to compute a new hidden representation for the center node.

recent literature. For instance, Gao and Ji [20] combine GCNs, GATs, and self-attention [6] for inductive graph prediction. Moreover, Filip et al. [21] show in another work promising results for the task of single-subject personality trait prediction through the use of a modified GAT architecture on fMRI images provided by the Human Connectome Project (HCP). These works suggest that the use of graph attention may prove fruitful for task state prediction.

2.5 Attention

Modern attention for neural networks was first introduced by Bahdanau et al. [22] as an extension of recurrent neural networks architectures for machine translation tasks. A limitation of this approach was poor parallelization within training examples [6]. Vaswani et al. [6] proposed a new attention-based architecture that overcame these limitations titled the transformer. Over the past few years, attention and transformer models have become increasingly popular in the machine learning community in particular for natural language processing and computer vision tasks.

More recently, the use of attention has extended into the domain of machine learning for neuroscience for use in task state decoding. Typically, in these works, attention is applied to a 3D fMRI input after it has been preprocessed by a convolutional deep network architecture such as ResNET-18 [23]. Other works, such as Qi et al. [24], incorporate a modified version of the transformer's encoder-decoder module directly into ResNET and InceptionNET architectures to either replace skip connections or to encode the hidden representations into a lower-dimensional latent space. Inspired by these works, we prepend convolutional layers to our modified transformer architecture and compare the results of that architecture with the rest of our baselines.

Furthermore, there have been various attempts at creating a hierarchical attention model. These works typically are completed for the study of sequence prediction in natural language processing tasks. In these approaches, the outputs of the transformer are pooled in a manner similar to that found in [25] and that input is then used as the input to the next transformer layer. This is repeated until the final output is passed through a fully connected network for a low-dimensional prediction. In both of these works, a single value is used as an input to the transformer layers and a downsampling operation occurs on the output of the transformer layers to create a hierarchical structure. In contrast to these prior attempts, we create a hierarchical structure by utilizing three distinct inputs to each transformer block. We let the values be the original input features to each layer, let the keys be learned feature representations of the inputs and let the queries be pooled representations of those inputs.

2.6 Baseline Architectures

In this section, we will describe the architectures of the baseline models that we will compare our channel attention architecture against.

2.6.1 Fixed Grouping Layer

For this baseline, we follow the general structure of the FGL architecture as found in [9]; however we do change the number of channels per layer to 32. Our baseline model has three FGLs with 13803, 4096, and 2048 nodes in each layer respectively. The final FGL reduces the number of nodes to 128

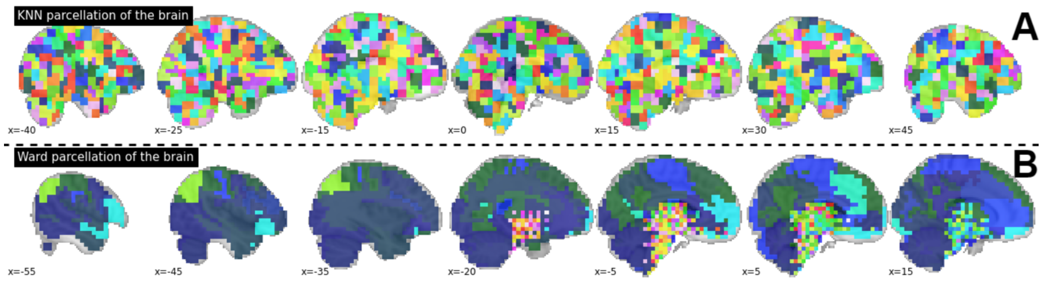

Figure 2.4: k-NN and Ward Parcellations: (A) Brain parcellations created using a k-nearest neighbors algorithm (where k = 10 neighbors) on the corresponding 3D coordinates of each fMRI voxel. To achieve a hierarchical parcellation, regions of interest of similar sizes are grouped together based on their proximity to one another. **(B)** Ward's algorithm to divide the brain. For small brain volumes, our implementation of Ward's parcellation resulted in few large ROI and several smaller ROI at the center of the brain.

and these nodes are sent to a fully connected layer for output predictions. We assign nodes to groups and perform a reduction of features in subsequent layers using the "max" variation of the FGL architecture. That is, we follow Equation 2.1 and pool the nodes at each layer by keeping the maximum $N^{(l+1)}$ nodes after applying assignment matrix S where $N^{(l+1)}$ is a model parameter for the number of input nodes to layer $l+1$. Afterwards, we discard the rest of the nodes. The handcrafted feature matrix S is formed by using either a Ward parcellation algorithm or a k-nearest neighbors algorithm (with k = 10) on the downsampled brain volumes. We compare the results of our brain parcellations on a downsampled brain image of size 37 x 45 x 37 voxels using both Ward parcellations and k-nearest neighbors (k-NN) parcellations in Figure 2.4.

2.6.2 Hierarchical GNN

For our second baseline, we utilize a hierarchical GNN (Figure 2.5) and, for each convolutional layer, we expand upon the layers of the graph convolutional network (GCN) introduced in [5]. Mathematically, we can write the representation of the nodes at each hidden layer of a GCN at layer l by the

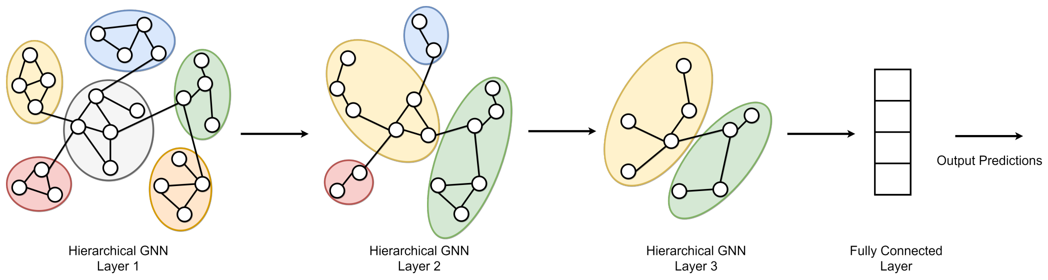

Figure 2.5: Hierarchical GNN Visualization: An illustration of a hierarchical GNN composed of three convolutional layers. After each layer receives input features, the nodes of that layer are grouped into six ROI. Within each convolutional layer, the network updates the hidden representation of each node based on their respective neighbors. In the second layer, the six ROI are grouped together into four ROI. Features within each ROI are combined to reduce the number of nodes. In the two subsequent layers, convolution is again used to update the representation of the nodes and ROI are combined. Downsampling occurs within each of the new ROI to compute a hierarchical representation. At the end, a final fully connected layer is reached and an output is found.

following formula:

$$H^{(l+1)} = \sigma\left(\tilde{D}^{-\frac{1}{2}}\tilde{A}\tilde{D}^{-\frac{1}{2}}X^{(l)}W^{(l)}\right) \tag{2.4}$$

where $X^{(l)}$ is the input feature vector, $H^{(l)}$ is the hidden representation of the nodes, $W^{(l)}$ is a linear transform that takes in an input in $\mathbb{R}^{D_i}$ and returns an output in $\mathbb{R}^{D_o}$, D_i is the number of input features per node, D_o is the number of output features per node, $\tilde{A} = A + I_N$ is the adjacency matrix of input graph G, I_N is an identity matrix with N diagonal elements, N is the number of nodes in the input graph, σ is an activation function such as ReLU and $\tilde{D}$ is a matrix that weights the connections between nodes in graph G [5].

To create our hierarchical graph convolutional architecture, we build upon Equation 2.4 by borrowing both the aggregation step and the learnable weight vector u from FGL. That is, we begin by taking learnable weight vector u and elementwise multiplying u with the output of Equation 2.4. Afterwards, to reduce the number of nodes in hierarchical fashion, we aggregate the nodes through the use of a linear transformation S that takes in an input of dimensionality $\mathbb{R}^{N^{(l)}}$ and returns an output of dimensionality $\mathbb{R}^{N^{(l+1)}}$. We then write the final expression for the output at layer l of our hierarchical GNN model as:

$$X^{(l+1)} = S\left(H^{(l+1)}\right) \circ u \tag{2.5}$$

where $\circ$ denotes the element-wise product.

Overall, we use four layers for our baseline architecture: three hierarchical graph convolutional layers followed by a fully connected layer. In our first hierarchical graph convolutional layer, we expand the number of input channels from 1 to 32. We maintain this number of channels for the two subsequent graph convolutional layers. The number of input nodes in each graphical layer is 13803, 4096, and 2048 respectively. To reduce the number of nodes per layer, we make use k-NN parcellations with k = 10. The output of the final graph convolutional layer has 128 nodes and 32 channels and is flattened before being sent to a fully connected layer for inference. We attempted to create edges between nodes through the use of Pearson's correlation coefficient, an identity matrix and through a learnable edge weight system inspired by [15]. In the case of defining an edge between regions of interest using correlation, an edge between regions was defined if the Pearson's correlation coefficient between the mean of two regions of interest was greater than 0.99. However, we found that edges formed with an identity matrix yielded the best performance.

2.6.3 Hierarchical GAT

For our third baseline, we implemented a hierarchical graph attention network. In this network, we replace the graph convolutional layers in our hierarchical GNN model with graph attentional layers. So, our model takes in an input fMRI volume, masks the input volume and separates the volume into input nodes. We then append three channels corresponding to the 3D coordinates of the input fMRI voxels to our input features. Next, we utilize a node weight vector u from FGL to learn which nodes are beneficial to hierarchical classification. Afterwards, we pass these features to three hierarchical graph attention layers. The final output is then used as an input to a fully connected layer for inference.

Within each graph attention layer, we update the hidden representation

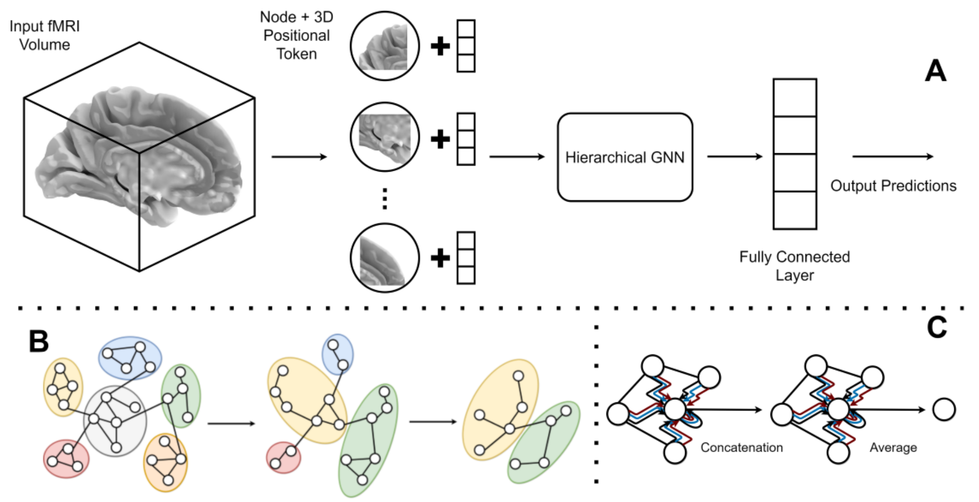

Figure 2.6: Hierarchical GAT Visualization: (A) An overview of the hierarchical graph attention model. Our hierarchical graph attention model takes in a 3D fMRI volume and separates the volume into several nodes. Each node is appended along the channel dimension with a 3×1 3D positional token based on the node's positional location in the input volume. The set of all nodes becomes our input to our hierarchical GAT model. **(B)** The hierarchical GAT model forms ROI and downstreams the input graph in a similar manner to our hierarchical GNN. **(C)** Each hierarchical GAT layer is composed of two sub-layers. The first uses self-attention as an aggregation scheme and concatenates the features to get an intermediate representation. Afterwards, this concatenated representation is passed to another attention layer where the learned representations are averaged to update the final hidden representations of the nodes.

$h_i^{(l)}$ for node i in layer l in the following manner:

$$y_i^{(l)} = \sigma\left(\| \sum_{j \in N_i} \alpha_{ij}^k W^k h_j^{(l)} \right) \tag{2.6}$$

$$h_i^{(l+1)} = \sigma\left(\frac{1}{K} \sum_{k=1}^{K} \sum_{j \in N_i} \alpha_{ij}^k W^k y_j^{(l)} \right) \cdot u_i \tag{2.7}$$

where K is the number of attention heads used, $\|$ is the concatenation operation, α_{ij}^k is the attention coefficient of node j on node i for head k, W^k is a linear transform that takes in an input in $\mathbb{R}^{D_i}$ and returns an output in $\mathbb{R}^{D_o}$ for head k, D_i is the number of input features per node, D_o is the number of output features per node, and u_i is the ith element of node weighting vector u.

After each update to the hidden node representation, we cluster the nodes in an aggregation step once again leveraged from FGL in order to lower the number of input feature vectors to the next set of graph attention layers. We set the number of nodes per layer to 13803, 4096, and 2048 respectively and the number of input channels to each layer except for the first to 32. The output of our final hierarchical graph attention layer has 128 nodes and 32 channels. This output is flattened and sent to a final fully connected layer for task predictions. Furthermore, we utilize an identity matrix to form our adjacency matrix for each graph layer since we once again found that edges formed with an identity matrix yielded the best in-sample and out-of-sample accuracy. A visualization of our final hierarchical graph attention network can be seen in Figure 2.6.

Chapter 3

CHANNEL ATTENTION MODEL

3.1 Hierarchical Transformer Network

In this chapter, we describe our hierarchical transformer architecture and describe several variants of this model.

3.1.1 Transformer Channel Attention Model

Our proposed hierarchical transformer architecture, as seen in Figure 3.1, is composed of three transformer channel attention layers and a fully connected layer. The first transformer channel attention layer expands the number of input channels from 1 to 32 and is made up of 13803 nodes. The next two layers maintain this number of channels and have 1024 nodes. The output of the final channel attention layer has 128 nodes and is flattened before it is sent to a fully connected layer to generate output predictions.

Each of the aforementioned transformer layers, as visualized in Figure 3.2, is modeled after the transformer encoder as defined in [6]. The input to a transformer layer is a matrix of node features $X \in \mathbb{R}^{N^{(l)}xD_i}$ where $N^{(l)}$ is the number of nodes at layer l and D_i is the number of features per node. We apply a linear transform v to the input X such that

$$Y = Xv \tag{3.1}$$

The transform v is a linear projection vector which takes an input in $\mathbb{R}^{D_i}$ and maps it to an output in $\mathbb{R}^{D_o}$ where D_o is the number of output features of the layer. Afterwards, the intermediate representation Y is used to compute

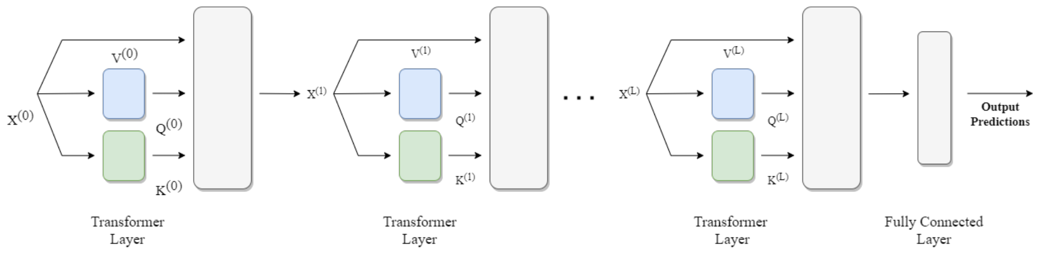

Figure 3.1: Hierarchical Transformer Network Visualization: An illustration of the overall transformer channel attention model composed of L channel attention layers. Each layer takes in three separate inputs: query Q, value V and key matrix K which are computed using the input feature matrix X. The output of each channel attention layer is used to compute the next set of Q, K, V matrices.

value V, key K, and query Q as follows:

$$V = Y^T \tag{3.2}$$

$$K = f_1(Y^T) \tag{3.3}$$

$$Q = f_2(Y^T) \circ \text{softmax}(f_3(Y^T)) \tag{3.4}$$

where f_1 is a neural network which takes an input in $\mathbb{R}^{N^{(l)}}$ and maps it to an output in $\mathbb{R}^{N^{(l)}}$ and f_2, and f_3 are neural networks which take inputs in in $\mathbb{R}^{N^{(l)}}$ and maps it to an output in $\mathbb{R}^{N^{(l+1)}}$.

The output of the multi-head attention is given by the following equation:

$$Z_1 = \text{softmax}\left(\frac{QK^T}{\sqrt{N^{(l)}}}\right) V \tag{3.5}$$

In order to get the input feature matrix $X^{(l+1)}$ to the next transformer layer from this output, we make use of a residual connection [26] and layer normalization [27] as follows:

$$Z_2 = \text{LayerNorm}(f_4(V) + Z_1) \tag{3.6}$$

$$X^{(l+1)} = (\text{LayerNorm}(f_5(Z_2) + Z_2))^T \tag{3.7}$$

where f_4 is a fully connected layer that projects inputs in $\mathbb{R}^{N^{(l+1)}}$ to outputs in $\mathbb{R}^{N^{(l+1)}}$. f_5 is a feed-forward network that consists of a ReLU transformation

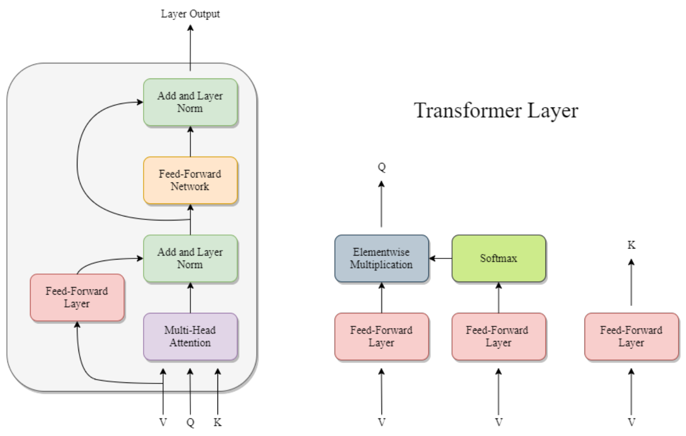

Figure 3.2: Transformer Channel Attention Layer: A diagram of a single transformer channel attention layer. In our transformer layer, we compute the queries, values and keys separately. We take features as value matrix V. K is learned through a neural network. Q is also learned through a neural network and then weighted by a learned probability score before being used in the transformer layer. This transformer layer is similar to the transformer encoder stack found in [6]; however, we replace the first skip connection with a feedforward network that lowers the dimensionality of matrix V.

and two linear transformations as shown below:

$$f_5(Z_2) = \max\left(0, W_1(Z_2) + b_1\right) W_2 + b_2 \tag{3.8}$$

where W_1 and W_2 are linear transforms that take in inputs in $\mathbb{R}^{d_{ff}}$ and $\mathbb{R}^{d_{model}}$ and return outputs in $\mathbb{R}^{d_{model}}$ and $\mathbb{R}^{d_{ff}}$ respectively. We set d_{ff} and d_{model} as 2048 and $N^{(l+1)}$ respectively and let b_1 and b_2 be learnable biases.

3.1.2 Transformer Node Attention Model

A simple variant of our architecture can be constructed in order to attend over nodes rather than over channels. In this variant of our transformer layer, we once again apply a linear mapping V to the input X to get intermediate

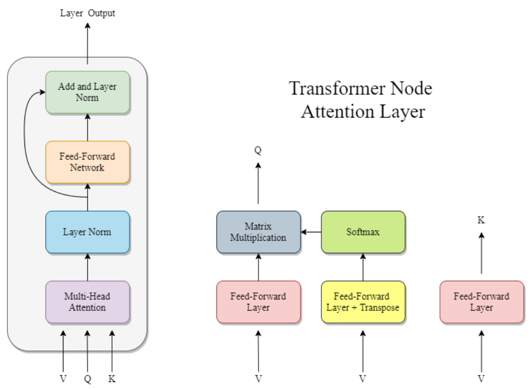

Figure 3.3: Transformer Node Attention Layer: A diagram of a single transformer node attention layer. This layer is similar to the channel attention layer. However, we remove the feed forward connection on V and lower the dimensionality of Q through the use of a learned similarity matrix as opposed to weighting the elements of Q in the channel attention formulation.

representation Y. We compute V, K, and Q as seen below:

$$V = Y \tag{3.9}$$

$$K = f_1(Y) \tag{3.10}$$

$$Q = \text{softmax}(f_2(Y))^T f_3(K) \tag{3.11}$$

where f_1, f_2, and f_3 are neural networks such that f_1 and f_3 maps from inputs in $\mathbb{R}^{D_o}$ to $\mathbb{R}^{D_o}$ and f_2 maps from inputs in $\mathbb{R}^{D_o}$ to $\mathbb{R}^{N^{(l+1)}}$. Like before, these vectors are used as inputs to transformer node attention layer l.

The output of the multi-headed attention block Z_1 can then be found as seen in Equation 2.1. That is,

$$Z_1 = \text{softmax}\left(\frac{QK^T}{\sqrt{D_o}}\right)V \tag{3.12}$$

where $Z_1 \in \mathbb{R}^{N^{(l+1)} \times D_o}$. Afterwards, Z_1 is used to get the output of a single transformer layer as seen below:

$$Z_2 = \text{LayerNorm}(Z_1) \tag{3.13}$$

$$X^{(l+1)} = \text{LayerNorm}(f_4(Z_2) + Z_2) \tag{3.14}$$

where f_4 is a feed-forward network as described in Equation 3.8 for $d_{ff} = 16$

and $d_{model} = D_o$. A complete overview of a node attention layer is provided in Figure 3.3.

Our final node-based attention model has three transformer layers stacked together in a hierarchical fashion with 13803, 4096, and 2048 nodes in each layer respectively. In the first layer of our model, we expand the number of input features from 1 to 32 and we maintain this number of features for the latter two transformer layers. The output of the final transformer layer contains 128 nodes, is once again flattened, and sent to a final fully connected network before a predictive score is found.

Finding Queries: Data Driven and Spatial Approaches

The query vector Q for layer l can be learned in two ways. The first is a similar manner to how it was constructed in Section 2.1.2 in an approach which we will refer to this approach as the data driven pooling approach. However, an interesting variant for constructing the query vector comes about by crafting the query vector through the use of an aggregation function to reduce the dimensionality of the input features $X^{(l)}$. In this spatial pooling approach, we construct the query vectors as follows:

$$Q = S^T X^{(l)} \tag{3.15}$$

where S is a linear transformation that maps from inputs in $\mathbb{R}^{N^{(l)}}$ to outputs in $\mathbb{R}^{N^{(l+1)}}$. The matrix S is crafted using regions of interest found through the Ward parcellation algorithm on resting state fMRI scans [9] or through the use of a k-nearest neighbors algorithm applied to the 3D coordinates of each fMRI volume.

3.1.3 Channel Attention Transformer Preprocessed with 3D Coordinate Convolution

A second variant of our transformer architecture can be constructed by combining 3D convolutional layers with channel attention layers as seen in Figure 3.4. Under this construction, we first preprocess the input 37 x 45 x 37 fMRI image using 3D convolutional layers. The first convolutional layer expands the input from one channel to four channels. The second layer takes in an

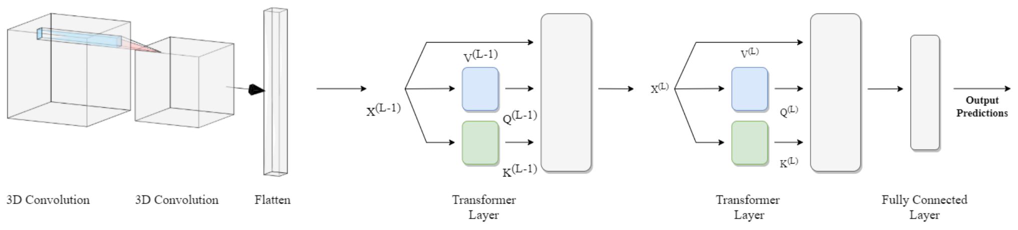

Figure 3.4: Convolution with Channel Attention: A diagram of our transformer model with 3D convolutional layers. 3D coordinate convolutional layers, as found in [9], are used to create a low dimensional embedding of the input volume. The output of the convolutional layers is then reshaped and used as an input to two channel attention layers as described in Section 3.1.1.

input volume of size 18 x 22 x 18 with four channels and outputs a hidden representation with 32 channels. In order to incorporate spatial information, in each layer, we append the 3D positions of each voxel of the input volume to the input as three separate channels. The output of the convolutional layers is reshaped and passed to two transformer channel attention layers stacked together in a hierarchical fashion. The input to the first transformer channel attention layer has 7128 nodes and 32 channels. The input to the second transformer channel attention layer has 2048 input nodes and 32 input channels. The final output contains 128 nodes, is flattened and is sent to a fully connected layer to get an output prediction.

3.1.4 Channel Attention Variant

A final variant for our channel attention model can be found if we adopt the node attention architecture for channel attention. An illustration of each transformer layer of this variant is given in Figure 3.5. In this network, we keep the same general structure of reading an input, processing that input through three channel attention layers and passing the attention output to a fully connected layer for a low-dimensional prediction. However, we construct

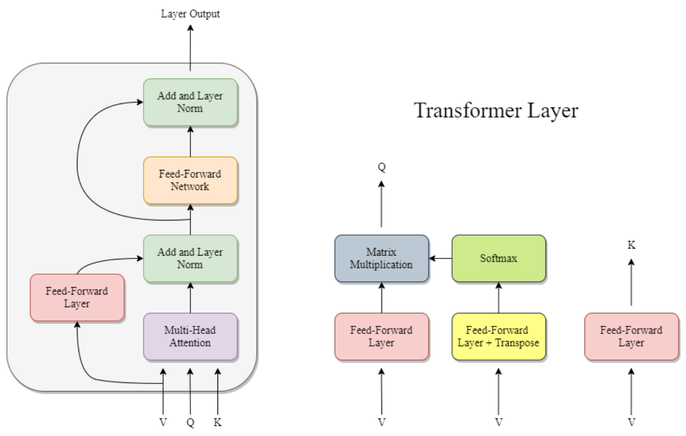

Figure 3.5: Variant of Channel Attention: A diagram of our variant of a single transformer channel attention layer. In this model, we compute the query matrix through the use of an learned weight matrix just as we did in the node attention model. We also leverage the feed forward connection in the channel attention from Section 3.1.1.

K, Q, and V in the following manner:

$$V = Y^T \tag{3.16}$$

$$K = f_1(Y^T) \tag{3.17}$$

$$Q = \text{softmax}(f_2(Y^T))^T f_3(K) \tag{3.18}$$

Again f_1, f_2, and f_3 are neural networks such that f_1 and f_3 maps from inputs in $\mathbb{R}^{N^{(l)}}$ to $\mathbb{R}^{N^{(l)}}$ and f_2 maps from inputs in $\mathbb{R}^{N^{(l)}}$ to $\mathbb{R}^{N^{(l+1)}}$. The remainder of the network architecture remains the same as the channel attention architecture described in Section 3.1.1.

Chapter 4

EXPERIMENTS

In this chapter, we compare the results of our channel attention deep network architecture against the aforementioned baseline models on five fMRI datasets which are publicly available on Neurovault. Moreover, we summarize the setup of our experiment and our results.

4.1 Datasets

For this thesis, we use five fMRI datasets which are publicly available on Neurovault [28]. Neurovault is a public web-based archive of human brain statistical maps [28]. The datasets leveraged from Neurovault for use in this thesis are described in the following list:

- **ARCHI** [29]: This dataset is comprised of 78 healthy subjects who were between the ages of 19 and 28 years old and in total contains 2340 fMRI images. The images collected in this dataset correspond to how subjects reacted to visual and auditory stimuli such as reading and comprehension of short sentences, subtractions and motor instructions. The images are broken up into three tasks based on either how the data was collected or what general category those tasks represented [30]. In total, there are 30 labels for the tasks in this dataset.

- **Brainomics** [29]: This dataset is a subset of the Fuctional Localizer dataset [29] and is composed of 94 healthy subjects with 1786 fMRI images. The mean age of the subjects was 24.7 years old. The Brainomics dataset aims to create a rough cognitive profile of each subject based of each subject's responses to questions tackling education, developmental disorders, reading difficulties, basic numerical knowledge, arithmetic and visuospatial abilities and visuomotor abilities [29].

- **Cam-CAN** [31]: The Cam-CAN dataset has fMRI scans for 605 patients between 18 and 87 years of age with 3025 images in total. The dataset aimed to capture the changes in cognitive ability due to aging through the use of two different types of tasks: AV-frequency tasks and audio video tasks. AV-frequency tasks concern how subjects reach to audio and visual stimuli at various frequencies. For the audio video tasks the subjects performed objectives to measure their motor coordination after audio or visual stimuli. All in all, these tasks are subdivided into five labels.

- **WU-Minn HCP 1200 Subjects (HCP)** [32]: This dataset is made up of 787 subjects with 18070 fMRI images where the subjects were healthy adults between the ages of 22 and 35 years old. The dataset can be broken up into resting state fMRI data and task state fMRI data. The tasks used to collect data from the subjects were broken into the following categories: working memory, gambling, motor, language, social cognition, relational processing and emotional processing [32]. In total, there are 23 labels for all of the tasks.

- **LA5c** [33]: This dataset was collected on participants between 21 and 50 years. In this dataset, the participants were divided into healthy and patient groups where members of the patient groups had previously been diagnosed with ADHD, bipolar disorder, or schizophrenia. Collectively, this dataset has 191 subjects with 5756 images. The tasks used to collect this dataset focused on understanding the dimensional structure of memory and cognitive control in subjects [33]. These tasks are divided into 24 labels.

As a preprocessing step, we downsampled each image in the previously mentioned datasets using nilearn which is a Python package based on scikit-learn designed to help out researchers with neuroimaging for machine learning [34]. Each 4D fMRI scan was decomposed into several 3D fMRI volumes of size 37 x 41 x 37.

4.2 Training Procedure

In this section we go over our the model configurations for the baseline architectures and our results. For each of the models, we follow a similar training setup to that used in [9]. That is, we begin by taking each dataset and splitting it into train and test sets such that there is no overlap between both sets of data. We hold 30% out of each dataset and use that for our testing (out-of-sample) dataset and the remaining 70% of each dataset is used for our train (in-sample) set. We repeat this process ten times and report the average of our best results over 35 epochs. For our optimizer, we used the Adam optimizer [35] with $\beta_1 = 0.5$ and $\beta_2 = 0.9$. For all models, we use a multi-step learning rate scheduler to decrease the learning rate by a factor of 10 after epoch 5 and epoch 25. In addition, we implement each model using Pytorch. We train each transformer model with a learning rate of 0.00005 and a batch size of 32. Whereas our graph-based models and FGL model are trained with a learning rate of 0.001 and a batch size of 32. Furthermore, we implement all models except for our FGL baseline on two Nvidia GeForce GTX Titan XP GPUs. For our FGL baseline, we utilize two K80 GPUs.

4.3 Results

In Tables 4.1, 4.2, 4.3, and 4.4, we compare the mean out-of-sample accuracy, the mean f1 score, the mean precision and the mean recall over ten random splits across all five neuroimaging datasets for the following network architectures: a hierarchical GNN, a hierarchical GAT model, our FGL baseline model using Ward and k-NN parcellations, a transformer model with 3D convolutional layers and our proposed transformer channel attention model. Furthermore, we provide figures showcasing the average accuracy and average loss at each epoch for our three best performing models which are our channel attention model, our transformer model prepended with 3D convolutional layers and our FGL baseline. The accuracy and loss curves for the ARCHI, the Brainomics, the Cam-CAN, the HCP, and the LA5c datasets can be found in Appendix A, Figures A.1, A.2, A.3, A.4, and A.5 respectively. On our figures, we denote the standard deviation of the accuracies and losses for each model at each epoch through the use of error bars.

Table 4.1: Out-of-sample accuracy on all datasets per model (in percent)

	ARCHI	**Brainomics**	**Cam-CAN**	**HCP**	**LA5c**
Hierarchical GNN Model	79.90	72.65	60.21	83.59	53.88
Hierarchical GAT Model	71.26	81.86	59.26	81.50	50.98
FGL (Ward Parcellation)	75.41	83.42	64.01	82.62	51.85
FGL (k-NN Parcellation)	85.01	89.76	**67.02**	87.46	60.46
Transformer 3D Convolution	84.52	93.16	64.73	89.61	63.31
Transformer Channel Attention	**88.12**	**94.45**	65.92	**91.26**	**64.16**

Table 4.2: F1 Score on all datasets per model (in percent)

	ARCHI	**Brainomics**	**Cam-CAN**	**HCP**	**LA5c**
Hierarchical GNN Model	80.08	72.41	59.69	83.43	52.35
Hierarchical GAT Model	71.50	81.98	58.95	81.44	49.54
FGL (Ward Parcellation)	75.51	83.22	63.75	82.54	50.29
FGL (k-NN Parcellation)	85.04	89.73	**66.60**	87.36	59.34
Transformer 3D Convolution	84.46	93.17	64.17	89.49	61.36
Transformer Channel Attention	**88.04**	**94.46**	65.47	**91.17**	**61.67**

Table 4.3: Mean precision on all datasets per model (in percent)

	ARCHI	**Brainomics**	**Cam-CAN**	**HCP**	**LA5c**
Hierarchical GNN Model	81.10	75.22	59.70	83.73	53.55
Hierarchical GAT Model	72.88	83.09	58.89	81.75	50.29
FGL (Ward Parcellation)	76.63	84.61	63.65	82.90	51.08
FGL (k-NN Parcellation)	85.67	90.44	**66.50**	87.55	61.06
Transformer 3D Convolution	85.00	93.46	64.10	89.62	61.95
Transformer Channel Attention	**88.48**	**94.70**	65.27	**91.29**	**62.83**

Table 4.4: Mean recall on all datasets per model (in percent)

	ARCHI	**Brainomics**	**Cam-CAN**	**HCP**	**LA5c**
Hierarchical GNN Model	79.90	72.65	60.21	83.55	52.13
Hierarchical GAT Model	71.26	81.86	59.26	81.46	49.46
FGL (Ward Parcellation)	75.41	83.42	64.01	82.57	50.29
FGL (k-NN Parcellation)	85.01	89.76	**67.02**	87.42	58.73
Transformer 3D Convolution	84.52	93.16	64.73	89.58	61.59
Transformer Channel Attention	**88.12**	**94.45**	65.92	**91.22**	**62.15**

From our results, we can see that our channel attention model surpasses all of our other benchmarks on all measurements on all datasets except for the Cam-CAN dataset. On the Cam-CAN dataset, we see that our FGL model with the use of k-NN parcellations performs best which shows a clear benefit to using spatial information to cluster nodes for inference in task state decoding. Moreover, we see that all of the models used in our experiments performed worst on the LA5c and Cam-CAN datasets. We speculate that this is related to the age and health conditions of the patients involved in those datasets. In general, the participants involved in the collection of the ARCHI, Brainomics, and HCP datasets were cognitively healthy adults between the ages of 20 and 30 [29], [32]. However, this was not the case for subjects of the Cam-CAN and LA5c datasets. While those involved in the collection of the Cam-CAN dataset were healthy, they varied in age by more than 60 years [31]. In addition, in the LA5c only 130 patients involved in the study were healthy individuals [33]. The remaining subjects had one of ADHD, bipolar disorder, or schizophrenia.

Chapter 5

CONCLUSION

In this thesis, we introduced an attention-based architecture for task state decoding. Our model takes entire 3D volumetric fMRI scans as inputs and outputs a prediction in an end-to-end fashion. By attending over channels instead of over nodes, we have demonstrated superior performance against multiple hierarchical end-to-end deep learning architectures across four of five neuroimaging datasets out-of-sample accuracy. We have also shown that pre-trained features, such as hierarchical clustering derived from Ward's algorithm, are not needed for good predictive performance.

With that said, where do we go from here? We envision multiple different directions that we could take to improve the results in this thesis. One such approach could be to combine the efforts of node-based attention and channel attention. In this thesis, we utilized nodal attention and channel attention separately in two different models. In future works, we could create a new attention layer that attends over channels and then attends over nodes in a hierarchical manner to downstream the data. Similar works have been attempted to attend over channels and nodes such as the work of [20]; however, to our knowledge there has not been a work that is solely based on transformers which utilizes node and channel attention for inference.

Another interesting direction to take this work could be found in making appropriate modifications to our model to incorporate larger input data sequences. In our current model, we downsample the input fMRI signals using nilearn due to having limited computational resources when computing both the query matrix and the attention matrix in the transformer layers. We can alleviate this issue by looking to related efforts in natural language processing tasks for computing transformer outputs for large input sequences. These efforts mainly extend the use of transformers to higher-dimensional data by splitting the computation of one large attention matrix into the several computations of multiple sparse attention matrices [36], [37]. These sparse attention matrices are summed together to get an approximation for the original attention matrix [36], [37]. In future works, we could build upon some of these insights to compute the attention matrix for larger fMRI signals within each of our transformer layers.

A final avenue to investigate for our work is related to determining the effect of locally connected spatial regions in the transformer. In prior works, it has been shown that positional information is beneficial to functional task state inference in transformer models [23]. Next steps for our transformer model could incorporate the use of a 3D positional embedding token to relate spatially connected regions in the input fMRI data.

Appendix A

ADDITIONAL MATERIALS

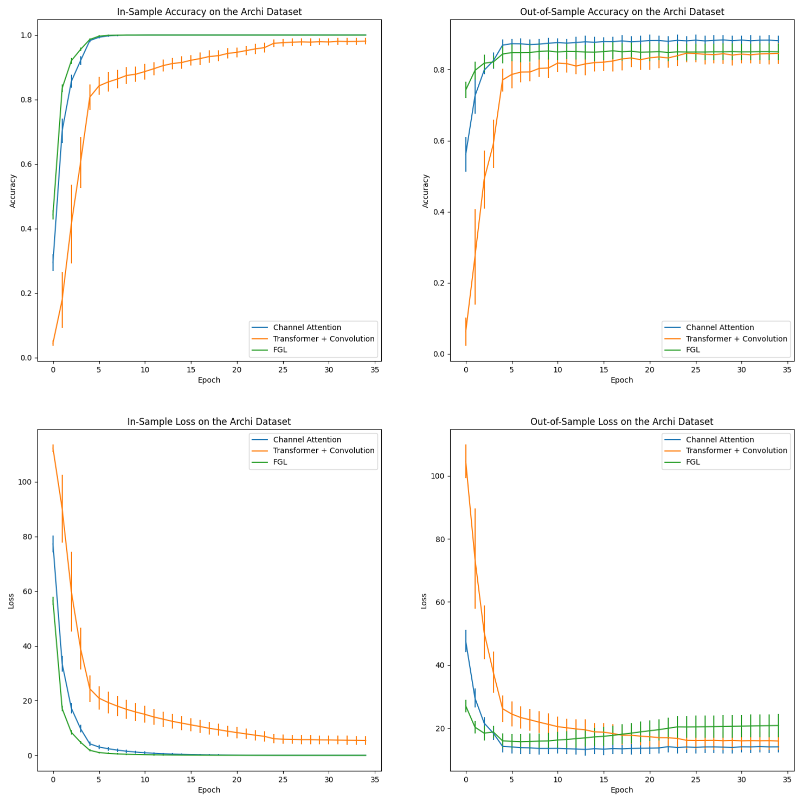

Figure A.1: In-Sample and Out-of-Sample Accuracy and Loss Curves for the ARCHI Dataset: Accuracy and loss curves per epoch for our channel attention model, our transformer preprocessed with convolutional layers model and our baseline FGL model over ten random splits of our data. We show the spread of the distribution of accuracies and losses by plotting the standard deviation of each model about the mean of each epoch through the use of error bars. We note that on the ARCHI dataset, both our transformer model prepended with convolutional layers and our FGL model converge to approximately the same out-of-sample accuracy after 35 epochs. Our channel attention model out-performs both of the aforementioned models and does not suffer from overfitting to the in-sample data.

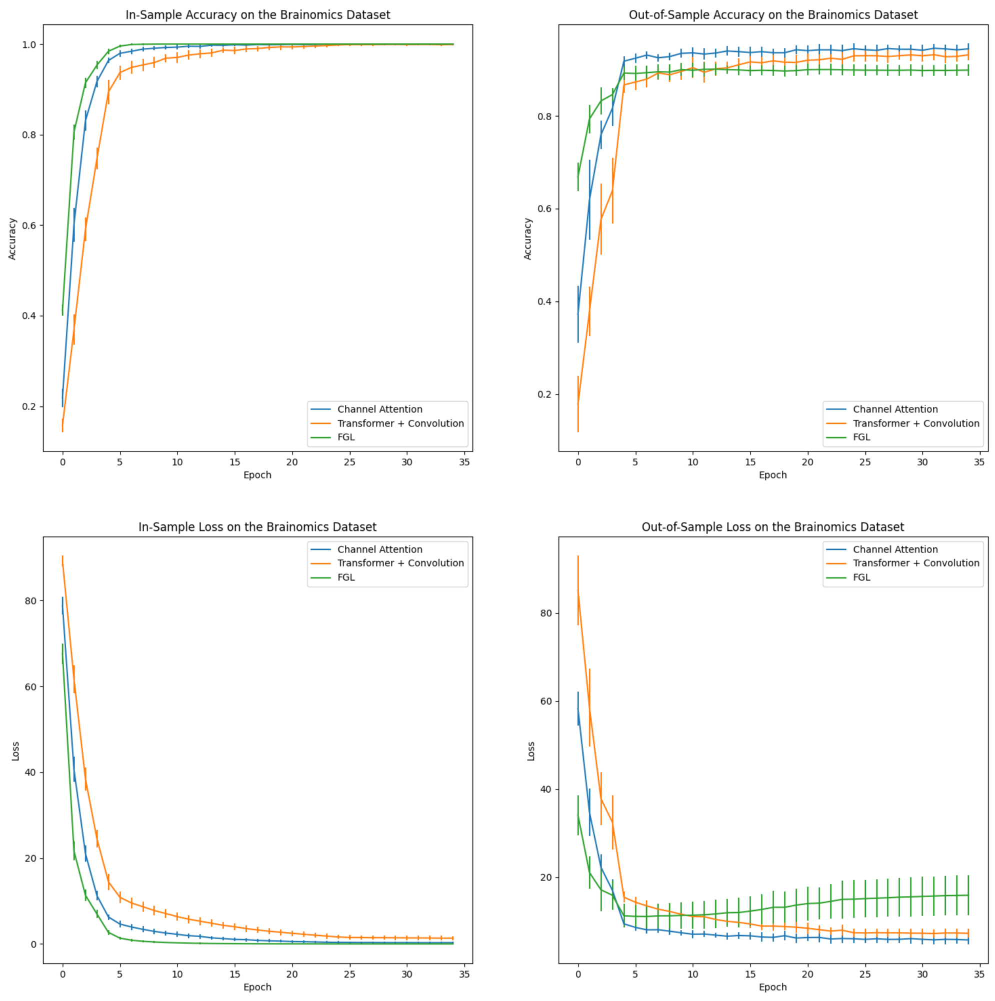

Figure A.2: In-Sample and Out-of-Sample Accuracy and Loss Curves for the Brainomics Dataset: Accuracy and loss curves for the three deep learning models with the highest out-of-sample accuracy on the Brainomics dataset averaged over ten random splits. On this dataset, we see that both transformer models have a higher out-of-sample accuracy than the FGL model. We also see that both transformer models suffer less from overfitting to the in-sample data than the FGL model does.

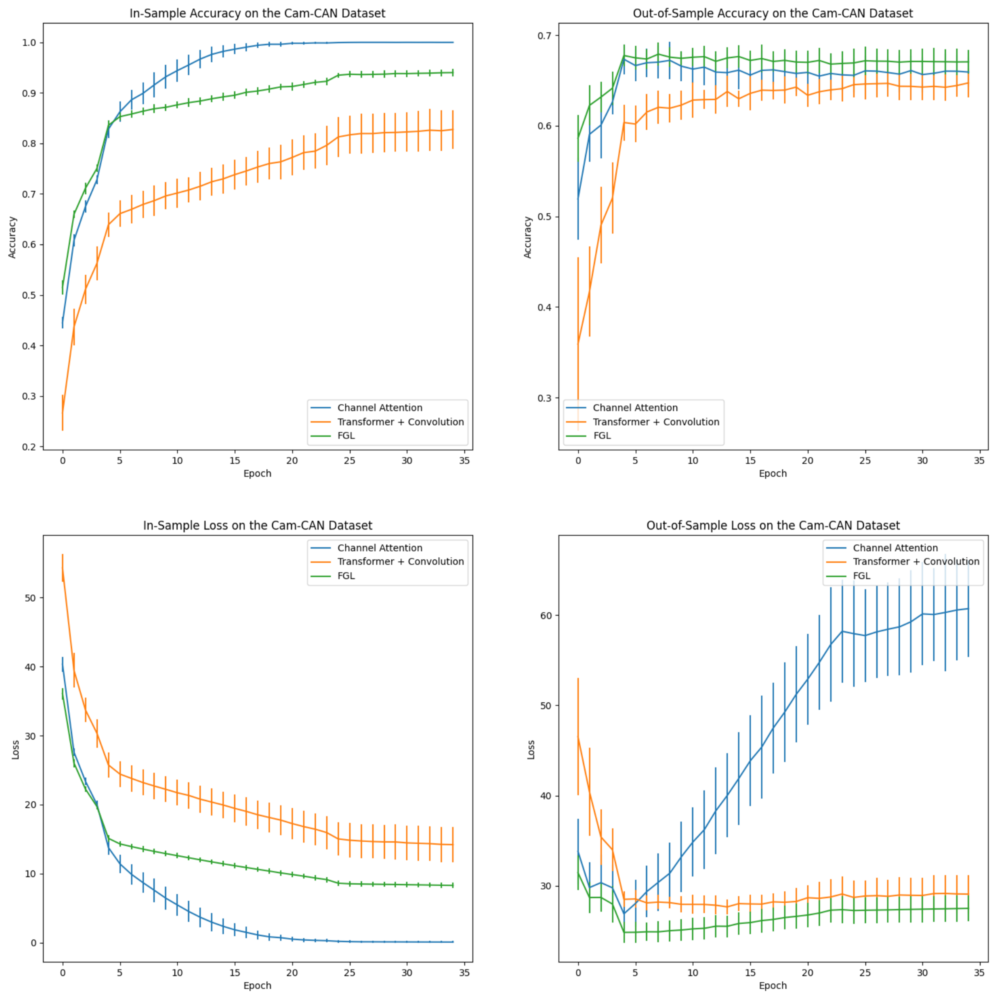

Figure A.3: In-Sample and Out-of-Sample Accuracy and Loss Curves for the Cam-CAN Dataset: Accuracy and loss curves over 35 epochs for the three deep learning models shown in our tabular results with the highest out-of-sample accuracy on the Cam-CAN dataset. On this dataset, the FGL baseline out-performs despite not overfitting to the in-sample data on this dataset. Our channel attention model suffers greatly from overfitting as seen in our out-of-sample loss curve and as seen by the decrease in performance in our out-of-sample accuracy curve.

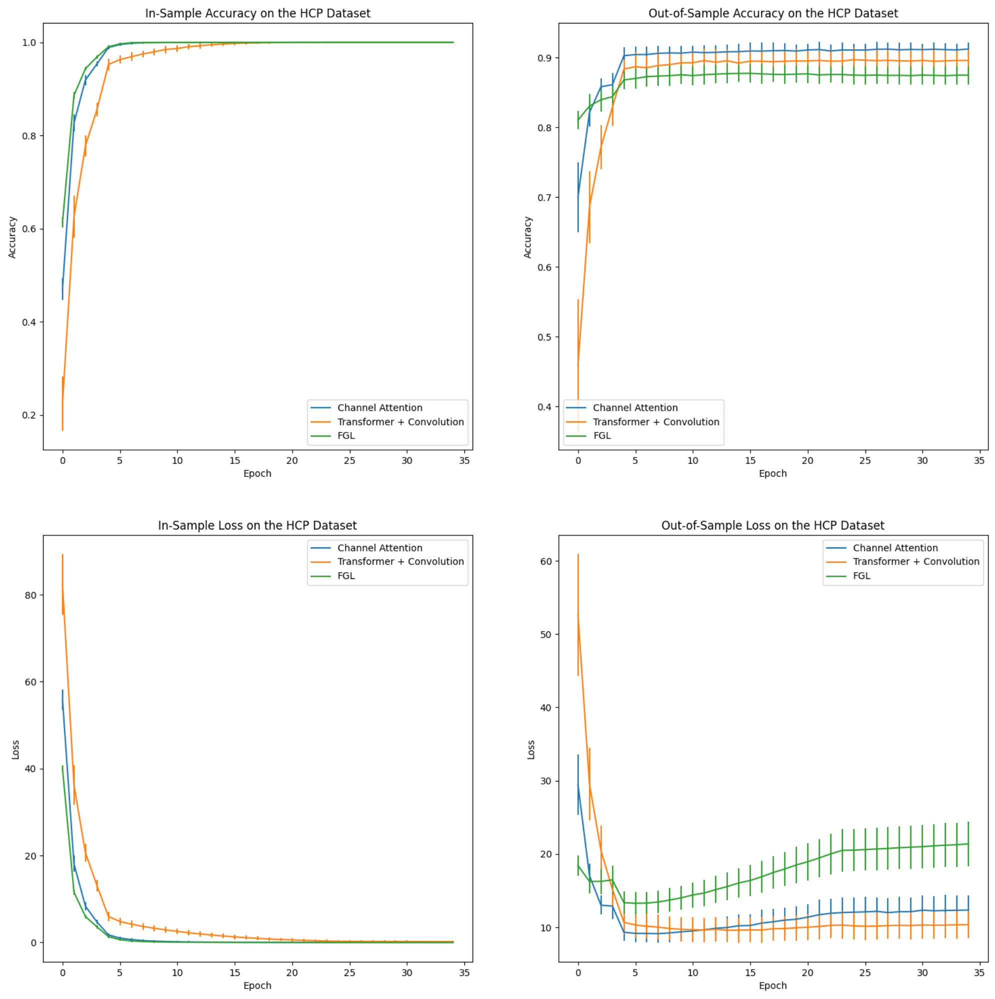

Figure A.4: In-Sample and Out-of-Sample Accuracy and Loss Curves for the HCP Dataset: Accuracy and loss curves for the three deep learning models with the highest out-of-sample accuracy averaged over 35 epochs on the HCP dataset over 10 random splits of our data. Once again, the transformer-based models out-perform the FGL baseline. We note that our channel attention model continues to have the best out-of-sample accuracy. Furthermore, we see that the out-of-sample accuracy curves of our models stay relatively flat which indicates that our models do not suffer much from overfitting to the HCP dataset.

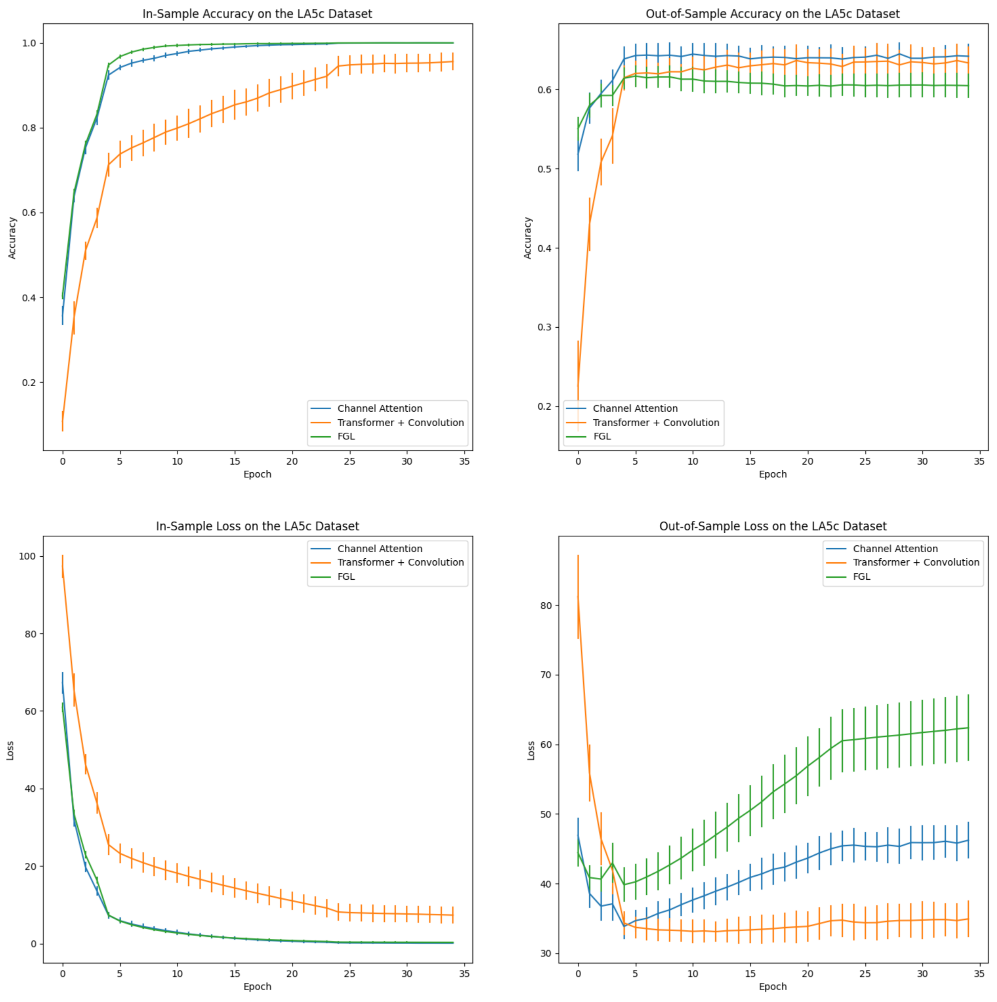

Figure A.5: In-Sample and Out-of-Sample Accuracy and Loss Curves for the LA5c Dataset: Accuracy and loss curves for our transformer models and our baseline FGL model over 10 random splits of our data. The transformer-based models continue to out-perform the FGL baseline. For this dataset, the channel attention model suffers from overfitting to the in-sample data whereas our transformer model prepended 3D convolutional layers has no such issue.

BIBLIOGRAPHY

[1] T. J. Sejnowski, P. S. Churchland, and J. A. Movshon, “Putting big data to good use in neuroscience,” *Nature Neuroscience*, vol. 17, no. 11, pp. 1440–1441, 2014.

[2] D. S. Bassett and O. Sporns, “Network neuroscience,” *Nature Neuroscience*, vol. 20, no. 3, pp. 353–364, 2017.

[3] T. Horikawa and Y. Kamitani, “Hierarchical neural representation of dreamed objects revealed by brain decoding with deep neural network features,” *Frontiers in Computational Neuroscience*, vol. 11, 2017.

[4] Y. Lecun and Y. Bengio, “Convolutional networks for images, speech and time series,” *The Handbook of Brain Theory and Neural Networks*, pp. 255–258, 1995.

[5] T. N. Kipf and M. Welling, “Semi-supervised classification with graph convolutional networks,” *International Conference on Learning Representations (ICLR)*, 2017.

[6] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. u. Kaiser, and I. Polosukhin, “Attention is all you need,” in *Advances in Neural Information Processing Systems*, I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, Eds., vol. 30. Curran Associates, Inc., 2017. [Online]. Available: https://proceedings.neurips.cc/paper/2017/file/3f5ee243547dee91fbd053c1c4a845aa-Paper.pdf

[7] E. Bullmore and O. Sporns, “Complex brain networks: Graph theoretical analysis of structural and functional systems,” *Nature Reviews Neuroscience*, vol. 10, no. 3, pp. 186–198, 2009.

[8] R. Chaudhuri, K. Knoblauch, M.-A. Gariel, H. Kennedy, and X.-J. Wang, “A large-scale circuit mechanism for hierarchical dynamical processing in the primate cortex,” *Neuron*, vol. 88, no. 2, pp. 419–431, 2015.

[9] H. Habeeb and O. Koyejo, "Towards a deep network architecture for structured smoothness," in *International Conference on Learning Representations*, 2020. [Online]. Available: https://openreview.net/forum?id=Hklr204Fvr

[10] M. M. Bronstein, J. Bruna, Y. Lecun, A. Szlam, and P. Vandergheynst, "Geometric deep learning: Going beyond Euclidean data," *IEEE Signal Processing Magazine*, vol. 34, no. 4, pp. 18–42, 2017.

[11] J. Bruna, W. Zaremba, A. Szlam, and Y. LeCun, "Spectral networks and locally connected networks on graphs," *arXiv preprint arXiv:1312.6203*, 2013.

[12] M. Henaff, J. Bruna, and Y. LeCun, "Deep convolutional networks on graph-structured data," *arXiv preprint arXiv:1506.05163*, 2015.

[13] Z. Wu, S. Pan, F. Chen, G. Long, C. Zhang, and P. S. Yu, "A comprehensive survey on graph neural networks," *IEEE Transactions on Neural Networks and Learning Systems*, vol. 32, no. 1, pp. 4–24, Jan 2021. [Online]. Available: http://dx.doi.org/10.1109/TNNLS.2020.2978386

[14] P. Veličković, G. Cucurull, A. Casanova, A. Romero, P. Liò, and Y. Bengio, "Graph attention networks," in *International Conference on Learning Representations (ICLR)*, 2018.

[15] T. Kipf, E. Fetaya, K.-C. Wang, M. Welling, and R. Zemel, "Neural relational inference for interacting systems," in *International Conference on Machine Learning*. PMLR, 2018, pp. 2688–2697.

[16] V. Badrinarayanan, A. Kendall, and R. Cipolla, "Segnet: A deep convolutional encoder-decoder architecture for image segmentation," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 39, no. 12, pp. 2481–2495, 2017.

[17] X. Wang, X. Liang, Z. Jiang, B. A. Nguchu, Y. Zhou, Y. Wang, H. Wang, Y. Li, Y. Zhu, F. Wu, and et al., "Decoding and mapping task states of the human brain via deep learning," *Human Brain Mapping*, vol. 41, no. 6, pp. 1505–1519, Apr 2020. [Online]. Available: http://dx.doi.org/10.1002/hbm.24891

[18] H. Yang, X. Li, Y. Wu, S. Li, S. Lu, J. S. Duncan, J. C. Gee, and S. Gu, "Interpretable multimodality embedding of cerebral cortex using attention graph network for identifying bipolar disorder," *bioRxiv*, 2019. [Online]. Available: https://www.biorxiv.org/content/early/2019/06/14/671339

[19] X. Li, Y. Zhou, N. Dvornek, M. Zhang, S. Gao, J. Zhuang, D. Scheinost, L. Staib, P. Ventola, and J. Duncan, "BrainGNN: Interpretable brain graph neural network for fMRI analysis," *bioRxiv*, 2021. [Online]. Available: https://www.biorxiv.org/content/early/2021/06/07/2020.05.16.100057

[20] H. Gao and S. Ji, "Graph representation learning via hard and channel-wise attention networks," in *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, 2019, pp. 741–749.

[21] A.-C. Filip, T. Azevedo, L. Passamonti, N. Toschi, and P. Lio, "A novel graph attention network architecture for modeling multimodal brain connectivity," in *2020 42nd Annual International Conference of the IEEE Engineering in Medicine Biology Society (EMBC)*, 2020, pp. 1071–1074.

[22] D. Bahdanau, K. Cho, and Y. Bengio, "Neural machine translation by jointly learning to align and translate," *arXiv preprint arXiv:1409.0473*, 2014.

[23] S. Nguyen, B. Ng, A. D. Kaplan, and P. Ray, "Attend and decode: 4D fMRI task state decoding using attention models," in *Machine Learning for Health.* PMLR, 2020, pp. 267–279.

[24] Y. Qi, H. Lin, Y. Li, and J. Chen, "Parameter-free attention in fMRI decoding," *IEEE Access*, vol. 9, pp. 48 704–48 712, 2021.

[25] R. Ying, J. You, C. Morris, X. Ren, W. L. Hamilton, and J. Leskovec, "Hierarchical graph representation learning with differentiable pooling," *arXiv preprint arXiv:1806.08804*, 2018.

[26] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," *arXiv preprint arXiv:1512.03385*, 2015.

[27] J. L. Ba, J. R. Kiros, and G. E. Hinton, "Layer normalization," *arXiv preprint arXiv:1607.06450*, 2016.

[28] K. J. Gorgolewski, G. Varoquaux, G. Rivera, Y. Schwarz, S. S. Ghosh, C. Maumet, V. V. Sochat, T. E. Nichols, R. A. Poldrack, J.-B. Poline, and et al., "Neurovault.org: A web-based repository for collecting and sharing unthresholded statistical maps of the human brain," *Frontiers in Neuroinformatics*, vol. 9, 2015.

[29] P. Pinel, B. Thirion, S. Meriaux, A. Jobert, J. Serres, D. L. Bihan, J.-B. Poline, and S. Dehaene, "Fast reproducible identification and large-scale databasing of individual functional cognitive networks," *BMC Neuroscience*, vol. 8, no. 1, 2007.

[30] D. T. Willingham and E. W. Dunn, "What neuroimaging and brain localization can do, cannot do and should not do for social psychology," *Journal of Personality and Social Psychology*, vol. 85, no. 4, pp. 662–671, 2003.

[31] M. A. Shafto, L. K. Tyler, M. Dixon, J. R. Taylor, J. B. Rowe, R. Cusack, A. J. Calder, W. D. Marslen-Wilson, J. Duncan, T. Dalgleish, and et al., "The cambridge centre for ageing and neuroscience (Cam-CAN) study protocol: A cross-sectional, lifespan, multidisciplinary examination of healthy cognitive ageing," *BMC Neurology*, vol. 14, no. 1, 2014.

[32] D. V. Essen, K. Ugurbil, E. Auerbach, D. Barch, T. Behrens, R. Bucholz, A. Chang, L. Chen, M. Corbetta, S. Curtiss, and et al., "The human connectome project: A data acquisition perspective," *NeuroImage*, vol. 62, no. 4, pp. 2222–2231, 2012.

[33] R. Poldrack, E. Congdon, W. Triplett, K. Gorgolewski, K. Karlsgodt, J. Mumford, F. Sabb, N. Freimer, E. London, T. Cannon, and et al., "A phenome-wide examination of neural and cognitive function," *Scientific Data*, vol. 3, no. 1, 2016.

[34] A. Abraham, F. Pedregosa, M. Eickenberg, P. Gervais, A. Mueller, J. Kossaifi, A. Gramfort, B. Thirion, and G. Varoquaux, "Machine learning for neuroimaging with scikit-learn," *Frontiers in Neuroinformatics*, vol. 8, 2014.

[35] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," *arXiv preprint arXiv:1412.6980*, 2017.

[36] R. Child, S. Gray, A. Radford, and I. Sutskever, "Generating long sequences with sparse transformers," *arXiv preprint arXiv:1904.10509*, 2019.

[37] M. Zaheer, G. Guruganesh, A. Dubey, J. Ainslie, C. Alberti, S. Ontanon, P. Pham, A. Ravula, Q. Wang, L. Yang, and A. Ahmed, "Big bird: Transformers for longer sequences," *arXiv preprint arXiv:2007.14062*, 2021.