Robust Spatial Filtering with Graph Convolutional Neural Networks

Felipe Petroski Such*, *Student Member, IEEE,* Shagan Sah*, *Student Member, IEEE,* Miguel Dominguez, *Student Member, IEEE,* Suhas Pillai, Chao Zhang, Andrew Michael, Nathan D. Cahill, *Senior Member, IEEE*, and Raymond Ptucha, *Senior Member, IEEE*
*equal contributions

***Abstract*—Convolutional Neural Networks (CNNs) have recently led to incredible breakthroughs on a variety of pattern recognition problems. Banks of finite impulse response filters are learned on a hierarchy of layers, each contributing more abstract information than the previous layer. The simplicity and elegance of the convolutional filtering process makes them perfect for structured problems such as image, video, or voice, where vertices are homogeneous in the sense of number, location, and strength of neighbors. The vast majority of classification problems, for example in the pharmaceutical, homeland security, and financial domains are unstructured. As these problems are formulated into unstructured graphs, the heterogeneity of these problems, such as number of vertices, number of connections per vertex, and edge strength, cannot be tackled with standard convolutional techniques. We propose a novel neural learning framework that is capable of handling both homogeneous and heterogeneous data, while retaining the benefits of traditional CNN successes.**

Recently, researchers have proposed variations of CNNs that can handle graph data. In an effort to create learnable filter banks of graphs, these methods either induce constraints on the data or require preprocessing. As opposed to spectral methods, our framework, which we term Graph-CNNs, defines filters as polynomials of functions of the graph adjacency matrix. Graph-CNNs can handle both heterogeneous and homogeneous graph data, including graphs having entirely different vertex or edge sets. We perform experiments to validate the applicability of Graph-CNNs to a variety of structured and unstructured classification problems and demonstrate state-of-the-art results on document and molecule classification problems.

***Index Terms*—graph signal processing, convolutional neural networks, deep learning.**

I. Introduction

Most naturally occurring problems can be described with an underlying graph structure. Functional MRIs, molecules, document databases, social networks, and 3D meshes in computer graphics can all be described by vertices connected by edges. For example, friends are connected through relationships, atoms are connected through bonds, and documents are connected by citations. Making inferences about these graphs and their elements is an active area of research.

Convolutional Neural Networks (CNNs) have forever changed the pattern recognition landscape with breakthrough results on image classification [1], [2], [3], [4], object detection [5], [6], and speech recognition [7]. It is natural to want to apply CNN methods to graph data to learn useful features. Graph problems are challenging because graph data does not have the gridded array structure that image, video, and signal data has. Each vertex (e.g. pixel) in gridded structures has the same number of neighbors and the same relationships to a neighbor in a given direction. Non-gridded graphs do not have these limitations. A non-gridded graph can vary in the number of neighbors from vertex to vertex, and there is not necessarily a geometrical interpretation for any given connection between two vertices.

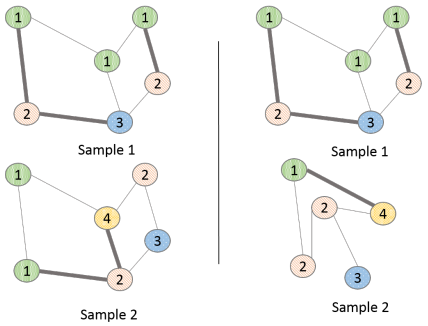


Figure 1: Two types of graph datasets. **Left:** Homogeneous datasets. All samples in a homogeneous graph data have identical graph structure, but different vertex values or "signals". **Right:** Heterogeneous graph samples. Heterogeneous graph samples can vary in number of vertices, structure of edge connections, and in the vertex values.

Filters are elegantly posed as spectral multipliers in a Fourier domain. Pooling can be modeled as spectral graph clustering operations as shown in [8]. Using these filter and pooling operations, numerous studies have applied CNNs on graphs [9], [8], [10], [11]. The downside of this approach is that the graphs that are processed by these models are required to be *homogeneous*. This means that each graph sample is required to have the same number of vertices and edge connections, as in Figure 1-Left. The samples can only differ in the "signal", that is, the vertex values in the graph. This is because the Fourier domain is unique for each graph. Spectral filters for one graph may not provide the same filtering behavior for another graph. *Heterogeneous* graphs (as in Figure 1-Right), which can vary in the number of vertices and the distribution of edge connections, cannot be processed with these models.

Defferrard et al. [12] introduced graph methods which are spectrally defined, but are implemented spatially with recursive polynomials on the graph Laplacian. This enables spectrally motivated approaches to handle heterogeneous graphs. For

example, [11] and [10] use [12] to measure the similarity between functional brain graphs and document classification respectively.

Sandryhaila, et al. [13] has shown that a shift-invariant convolution filter can be represented as a polynomial of adjacency matrices. We note the adjacency polynomial "translation" is an isotropic diffusion from the current vertex to vertices farther away. Like [12], weights are shared among a graph regardless of (any heterogeneous) structure. Each weight is used by all vertices of a given distance from the current vertex. We define filters as polynomials of functions of the graph adjacency matrix to define a useful spatial Graph-Convolutional Neural Network. Like [13], [12], our work allows filters to be applied to heterogeneous graphs without going into the spectral domain. We also exploit structure in certain graphs (such as images or 3D meshes) to provide anisotropic filtering that varies based on angle. Filters are learned directly from graph adjacency matrices and vertex features in the spatial domain. The code is downloadable from [14].

The contributions of this research are as follows:

- We introduce the concept of vertex filters on graphs. Vertex filters simultaneously learn properties from both graph vertices and edges. The technique also enables learning from multiple adjacency matrices with individual edge features or graphs.
- We provide a supervised graph embed pooling operation to learn a pooling transformation for heterogeneous graph data.
- We do extensive experiments with multiple graph datasets- brain fMRI, chemical compounds, 3D face and document citations, to show the applicability of our model to varying graph structures. We also show that our graph formulation performs similarly to a classical CNNs on CIFAR-10 and ImageNet datasets.

This work is a step towards creating a one-to-one mapping between deep convolutional neural networks for signals and images and one for graphs as represented in Figure 2. Further, the presented methods can be applied to graphs that are homogeneous or heterogeneous in nature.

The paper is organized as follows: Section II outlines related work. Section III describes the proposed Graph-CNN model in detail including different filtering techniques. Section IV details the numerical experiments on graph data- images, brain imaging, facial expression recognition, document classification, and chemical compounds. Section V discusses the computational complexity. Section VI contains concluding remarks.

II. Related work

In general graph data is encoded by the tuple $\boldsymbol{G} = (\boldsymbol{V}, \boldsymbol{A})$. $\boldsymbol{V} \in \mathbb{R}^{N \times C}$ is the vertex data or graph signal, where N vertices each contain C vertex features. $\boldsymbol{A} \in \mathbb{R}^{N \times N}$ is the adjacency matrix, which encodes the connections between vertices. The adjacency matrix entries can be defined as in (1).

$$a_{ij} = \begin{cases} w_{ij} & \text{if there is an edge between } i \text{ and } j \\ 0 & \text{otherwise} \end{cases} \tag{1}$$

The scalar w_{ij} is a weight that represents some measure of strength of the edge between vertex i and vertex j.

There are three general approaches we found to generalizing Deep Neural Networks for graph data: spectral, spatial, and geometric. There is some overlap as elements of spectral graph theory are frequently used throughout the literature.

A. Spectral Approaches

Spectral approaches exploit spectral graph theory. These works filter in a spectral domain by constructing an analogue to the Discrete Fourier Transform (DFT), which is based on the eigenvector decomposition of the Graph Laplacian. The Graph Laplacian is shown in (2) and the normalized Graph Laplacian is shown in (3). $\boldsymbol{A}$ is the adjacency matrix, $\boldsymbol{D}$ is the diagonal degree matrix, whose entries are the row-wise sums of $\boldsymbol{A}$, and $\boldsymbol{I}$ is the identity matrix.

$$\boldsymbol{L} = \boldsymbol{D} - \boldsymbol{A} \tag{2}$$

$$\boldsymbol{L} = \boldsymbol{I} - \boldsymbol{D}^{-1/2}\boldsymbol{A}\boldsymbol{D}^{-1/2} \tag{3}$$

$\boldsymbol{L}$ can be used to compute an eigenbasis $\boldsymbol{U}$ that represents an analogue to the DFT matrix. Then a graph signal $\boldsymbol{x}$, which contains the vertex values of the graph, can be filtered spectrally by transforming $\boldsymbol{x}$ into the spectral domain and multiplying each frequency by a filter $\boldsymbol{h}$, as in (4) ($\odot$ is the elementwise product).

$$\boldsymbol{x} * \boldsymbol{h} = \boldsymbol{U}^T \cdot (\boldsymbol{U}\boldsymbol{x} \odot \boldsymbol{h}) \tag{4}$$

Eigenvectors of the Graph Laplacian represent frequency components, similar to rows of the DFT matrix. By transforming graph signals to a spectral domain with the resulting eigenbasis, the graph signal can be multiplied by an array of filter coefficients to perform a filtering operation. Several works propose graph CNN models that are based on this method of filtering [9], [8], [12], [15], [10]. One of the advantages of this is that these works can leverage a mature body of literature on spectral clustering to propose effective graph-pooling mechanisms (an introduction to spectral clustering can be found in [16]). Some also use off-the-shelf software such as Graclus [17].

One initial concern about spectral approaches to graph convolutions is that these filters would not be localized in the spatial domain, meaning that learned weights would not be shared across different locations in the graph. Recent developments have counterintuitively shown that spectral filters can be localized in space. In [9], [8], [15], smooth spectral filters lead to localized filters in the spatial domain, leading to localized filters. In [18], [12], it is shown that a K-order polynomial formulation of the graph Laplacian perform a K-hop filtering operation, despite being a "spectral" operation. In addition [12] reveals an efficient recursive approximation of this spectral filtering using Chebyshev polynomials. This technique is used for semi-supervised document classification

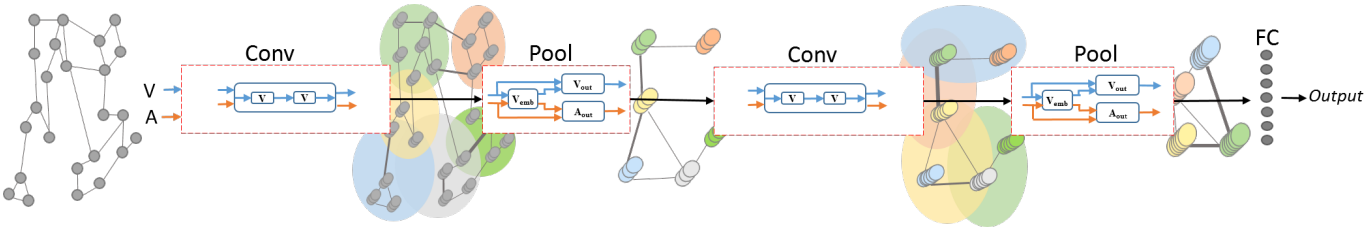


Figure 2: General vertex-edge domain Graph-CNN architecture. Convolution and pooling layers are cascaded into a deep network. FC are fully-connected layers for graph classification. V is vertex set and A is adjacency matrix that define a graph.

in [10]. Graph signals filtered with this technique are fed through an LSTM in [19] to model sequences.

One of the major practical limitations when learning filters in the spectral domain is that the eigenbasis that transforms a graph between spatial and spectral domains is unique for each graph. This requires input samples to be homogeneous. The Graph Laplacian eigenbasis needs to be solved separately for each unique graph structure. Most spectral works tend to focus on experiments where there is a single graph structure common across all samples, such as the MNIST image dataset [20]. More recent works based on polynomials of the Laplacian do not have this limitation [10], [11], [12]. By representing polynomial filters as linear combinations of Chebyshev polynomials, they are able to exploit the Chebyshev polynomial recurrence relation to rapidly apply these filters in the spatial domain (An interesting aspect of this approach is that it is motivated by spectral graph theory, but implemented spatially).

B. Spatial Approaches

Spatial approaches have an advantage over some spectral approaches in that they do not, by nature, require a homogeneous graph structure. However, they generally require sophisticated data preprocessing to enable learning. The challenge is in figuring out how to process neighborhoods that are different sizes and structures for each vertex. Diffusion-Convolutional Neural Networks (DCNNs) [21] model the graph by encoding layers of matrices that arrange vertex features based on a sequence of hops from different starting vertices. However, due to lack of a vertex pooling or clustering layers, it was not expanded to learn beyond the original level of abstraction. PATCHY-SAN attempts to linearize a graph based on the CNN concept of a receptive field [22]. [23] defines a hashing operation inspired by CNN properties and uses it to learn features on molecular fingerprints. Gated Graph Sequential Neural Networks [24] presented a graph Long Short-Term Memory (LSTM) Recurrent Neural Network model applied to program verification and basic logical reasoning tasks, extending a neural network model that learned a graph encoding from data [25]. DeepWalk [26] learns latent representations of graph vertices by using random walks to extract local information which encodes structural regularities in social networks.

Our work is also a spatial approach. Like the spectral approaches, our work is inspired by discrete graph signal processing theory. Specifically, we benefit from the spatial filtering theory proposed in [13]. Where we differ from current spatial approaches is that we require less preprocessing to format the graph structure. We learn convolutional filters directly on the adjacency matrices and vertex features that naturally represent graph data.

C. Geometric Approaches

Researchers from the 3D shape analysis community have also been working on the problem of signal processing on heterogeneous data structures. Several recent works have tried to develop filters and CNNs for manifolds, oftentimes for the purpose of point correspondence on 3D meshes. One popular approach is to map individual patches of these manifolds to an alternative representation that is more amenable to filtering. The idea is that filtered patches of similar objects (e.g. fingertips) should have similar feature representations between meshes that only differ in deformation. Some example representations are 2D polar coordinate representations that are filtered with a polar convolution[27], local windowed spectral representations [28], anisotropic variants of heat kernel diffusion filters and spectral filters [29], [30], and learned Gaussian Mixture-Model kernels [31]. A survey of this approach is provided in [32]. The Gaussian Mixture Model method was generalized to graph data and applied to MNIST classification and document classification [31].

III. The Graph-CNN Model

Our work uses the standard definition of graphs as described above. We place no constraints on $\boldsymbol{A}$ beyond what is described in (1). We allow the graph to be directed (a_{ij} does not necessarily equal a_{ji}), have reflexive connections (a_{ii} is a valid connection), and different graph samples to be heterogeneous: each sample may have different numbers of vertices and differing graph structures. This latter feature is an improvement over many spectral methods, where filters are learned on a particular "homogeneous" arrangement of vertices and edges.

For our work, we also define an adjacency tensor $\boldsymbol{\mathcal{A}} \in \mathbb{R}^{N \times N \times L}$, which is a stack of L adjacency matrices $(\boldsymbol{I}, \boldsymbol{A}_1, \boldsymbol{A}_2 \ldots \boldsymbol{A}_{L-1})$, where $\boldsymbol{I}$ is the identity matrix (only reflexive connections). This allows us to encode multiple edge features or to partition edges based on a hand-picked structure.

Throughout this work we utilize the notation defined in I.

A. Graph Filters

Sandryhaila, et al. [13] recently defined a spatial-domain convolution for graphs, shown in (5).

$$\boldsymbol{H} = h_0\boldsymbol{I} + h_1\boldsymbol{A} + h_2\boldsymbol{A}^2 + ... + h_k\boldsymbol{A}^k, \boldsymbol{H} \in \mathbb{R}^{N \times N} \quad (5)$$

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/JSTSP.2017.2726981, IEEE Journal of Selected Topics in Signal Processing

Table I: Graph-CNN notation.

	Dimensions	Description
N	$\mathbb{R}$	# of vertices
C	$\mathbb{R}$	# of vertex features
L	$\mathbb{R}$	# of edge features
F	$\mathbb{R}$	# of filters
$\boldsymbol{V}$	$\mathbb{R}^{N\times C}$	Vertex matrix
$\boldsymbol{A}$	$\mathbb{R}^{N\times N}$	Adjacency matrix
$\boldsymbol{\mathcal{A}}$	$\mathbb{R}^{N\times N\times L}$	Adjacency tensor (multiple $\boldsymbol{A}$s)
$\boldsymbol{H}$	$\mathbb{R}^{N\times N\times C\times F}$	Graph filter

The filter is defined as the kth-degree polynomial of the graph's adjacency matrix. Each exponent in the polynomial encodes the number of hops from a given vertex that are being multiplied by the given filter tap. $\boldsymbol{A}^1$ (or $\boldsymbol{A}$) represents the one-hop neighbors of the given vertex. $\boldsymbol{A}^2$, the square of the adjacency matrix, represents the two-hop neighbors, and so on. $\boldsymbol{I}$ represents the 0-hop or the vertex being processed. The scalar coefficients h_0, h_1,...,h_k control the contribution of the neighbors of a vertex during the convolution operation.

To convolve the vertices $\boldsymbol{V}$ with the filter $\boldsymbol{H}$ is a matrix multiplication, $\boldsymbol{V_{out}} = \boldsymbol{H}\boldsymbol{V_{in}}$ where $\boldsymbol{V_{in}}, \boldsymbol{V_{out}} \in \mathbb{R}^N$.

B. Graph-CNN

To adapt this filtering operation into a convolution operation fit for our Graph-CNN, we need to take into account the desire to process multiple filters and multiple adjacency matrices per sample. We also want to use the intuition from VGGNet [2] that learning cascades of small filters can effectively capture the receptive field of a single large filter. To that end we approximate (5) as a linear equation in (6).

$$\boldsymbol{H} \approx h_0\boldsymbol{I} + h_1\boldsymbol{A} \tag{6}$$

This equation is cascaded over multiple layers in a neural network, thereby achieving the receptive field of 5 but with nonlinearities at every step. A comparable linear approximation for a spectral approach was used by Kipf, et al. [10]. Their approach similarly cascaded these linear filters to approximate the K-hop filter formed by the polynomial of the Laplacian.

We want to scale this operation up to use our adjacency tensor $\boldsymbol{\mathcal{A}}$. This construct contains multiple adjacency matrices in a sample. The first slice of this tensor is $\boldsymbol{A}_1$, the second slice is $\boldsymbol{A}_2$ and so on. Each slice $\boldsymbol{A}_\ell$ encodes a particular edge feature for the graph in an adjacency matrix. We define a linear filter as a convex combination of each adjacency matrix as in (7).

$$\boldsymbol{H} \approx h_0\boldsymbol{I} + h_1\boldsymbol{A}_1 + h_2\boldsymbol{A}_2 + \ldots h_{L-1}\boldsymbol{A}_{L-1} \tag{7}$$

This can be written compactly as in (8).

$$\boldsymbol{H} \approx \sum_{\ell=1}^{L} h_\ell \boldsymbol{A}_\ell \tag{8}$$

One motivation for these multiple adjacency matrices is to encode multiple edge features, one feature in each $\boldsymbol{A}$. Another is to partition edges in a single $\boldsymbol{A}$ into multiple matrices to impart a sense of direction. Figure 3 illustrates this motivation. The figure on the left is an illustration of the default Graph-CNN linear filter in an image application. A given filter tap is applied to all vertices of a given distance, isotropically. In this case, h_0 is applied to the 0-hop vertex and h_1 is applied to all adjacent vertices. If another border of pixels surrounded this figure, each pixel in that border would be multiplied by a filter tap h_2.

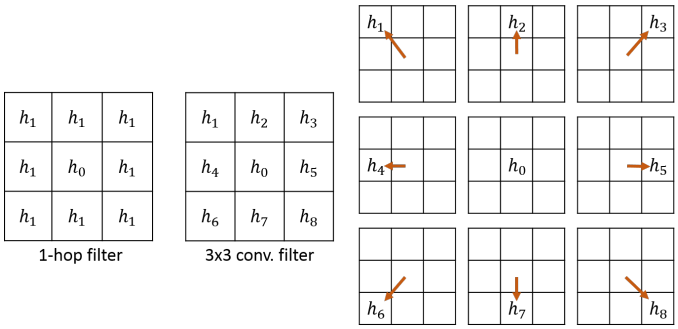


Figure 3: **Left:** Learnable parameters in 1-hop graph filters. **Center:** Classical 3×3 convolution filters. **Right:** Illustration of eight different edge connections combined to form a 3×3 filter.

If however, the adjacency matrix was partitioned into nine adjacency matrices, each one representing a different relative connection to a given vertex (upper-left, right, down, lower-right, and so on), then there would be a unique filter tap for each direction, including the 0-hop self-connection (Figure 3, right). This results in an anisotropic Graph-CNN filter. This is equivalent to a 3×3 FIR filter in conventional CNN applications (Figure 3, center). The Appendix furnishes a proof of this that explains the model in more detail. For graph datasets such as images and 3D meshes that have exploitable structure, this method can multiply parameters and increase the modeling capability of the network.

So far we have described filters for a single vertex feature. To have a set of filter coefficients for multiple vertex features, each h_ℓ needs to be in $\mathbb{R}^C$, leading $\boldsymbol{H}$ to be in $\mathbb{R}^{N\times N\times C}$. This means $\boldsymbol{H}$ is a stack of $N\times N$ filter matrices indexed by the vertex feature they filter. Equation (8) can be modified as in (9) to illustrate this new operation.

$$\boldsymbol{H}^{(c)} \approx \sum_{\ell=1}^{L} h_\ell^{(c)} \boldsymbol{A}_\ell \tag{9}$$

In (9), $\boldsymbol{H}^{(c)}$ is an $N\times N$ slice of $\boldsymbol{H}$ and $h_\ell^{(c)}$ is a scalar corresponding to a given input feature and a given slice of $\boldsymbol{A}_\ell$. To filter a vertex signal $\boldsymbol{V}_{in}$ using this filter tensor, we perform the operation described in (10).

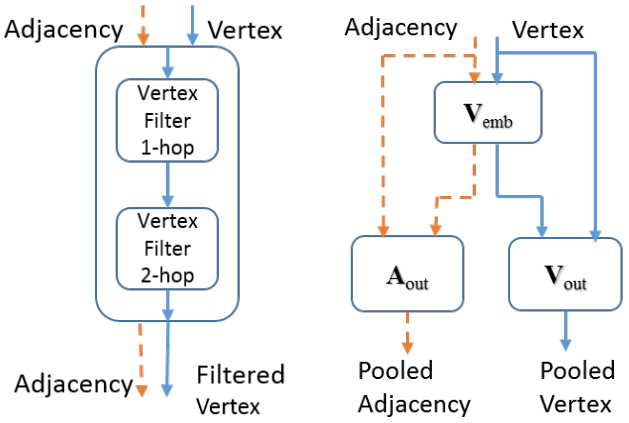
$$\boldsymbol{V}_{out} = \sum_{c=1}^{C} \boldsymbol{H}^{(c)} \boldsymbol{V}_{in}^{(c)} + b \tag{10}$$

$\boldsymbol{V}_{in}^{(c)}$ is the column of $\boldsymbol{V}_{in}$ that only contains vertex feature c. We also add a bias $b \in \mathbb{R}$. This results in $\boldsymbol{V}_{out} \in \mathbb{R}^N$. This is analogous to an image with multiple color channels being filtered down to a single grayscale channel in image-based

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/JSTSP.2017.2726981, IEEE Journal of Selected Topics in Signal Processing

CNNs. Multiple filters can be modeled by adding another dimension to $\boldsymbol{H}$ so that it can become part of $\mathbb{R}^{N\times N\times C\times F}$. Then (10) can be repeated, with each filter output representing a single column $\boldsymbol{V}_{out}^{(f)}$ in $\boldsymbol{V}_{out} \in \mathbb{R}^{N\times F}$. Each feature in the output vertices is the output of a single filtering operation across all features in the input vertices. In this case there are also F biases, one for each filter.

To be clear, these vertex filters only change the vertex data. The adjacency data is used to help filter the vertices, but remains unchanged by the operation. Figure 4a illustrates this.



(a) Graph Convolution (b) Graph Pooling

Figure 4: Graph convolution and pooling setting. The convolution operation obtains a filtered representation of the graph after a multi-hop vertex filter. Likewise, a compact representation of the graph after a pooling layer.

C. Initialization

Like other neural network models, Graph-CNNs require proper weight initialization. A common way of doing so is by initializing the weights with random numbers generated from a Gaussian distribution. This works well with small networks, but deep networks suffer from vanishing (or exploding) gradients. Xavier initialization [33] addresses this somewhat, by intelligently selecting the Gaussian distribution parameters to mitigate changes in the distribution of the input and output data. Xavier bases the parameters on the number of inputs and number of outputs of each filter. A more complex variant would be required for graph data. This is because the output distribution depends on the weights, the input vertices, and the input adjacency matrix. Batch Normalization [34] offers a more elegant solution. By normalizing each batch of data at each layer, vanishing and exploding gradients are explicitly prevented.

D. Graph Embed Pooling

An important building block of CNNs are pooling layers. Reducing dimensions of the input allows convolution filters to have a larger receptive field and also improves computation performance. One of the most common methods for pooling images is max-pooling. This method selects the top value over a defined region in a sliding window approach.

Pooling methods designed for images are tailored for gridded structures and cannot be applied to general graphs due to their often heterogeneous structure. As a solution, we introduce a method called *graph embed pooling*. Graph embed pooling learns a convolutional layer whose output can be treated as an embedding matrix that produces a fixed-size output. To produce a pooled graph reduced to a fixed N' vertices, the learned filter taps from this pooling layer produce an embedding matrix $\boldsymbol{V}_{emb} \in \mathbb{R}^{N\times N'}$. Similar to (10), a filter tensor $\boldsymbol{H}_{emb} \in \mathbb{R}^{N\times N\times C\times N'}$ is learned and multiplied by the vertices to produce a filtered output, as in (11). Equation (11) could be replaced with any layer that produces a fixed number of features for each vertex. We use the previously defined 1-hop Graph-CNN filter.

$$\boldsymbol{V}_{emb}^{(n')} = \sum_{c=1}^{C} \boldsymbol{H}_{emb}^{(c,n')} \boldsymbol{V}_{in}^{(c)} + b \tag{11}$$

Each output in the equation is a column $\boldsymbol{V}_{emb}^{(n')}$ in the output matrix $\boldsymbol{V}_{emb}$, indexed by row $n' \in 1, 2, \ldots N'$. Like the other filter tensors, $\boldsymbol{H}_{emb}$ is produced in an analogous fashion to (9). This means that a $O(N^2CN')$ variable-dimension embedding matrix $\boldsymbol{H}_{emb}$ (N is variable) can be learned with $O(LCN')$ parameters. Equations (13) and (14) show how $\boldsymbol{V}_{emb}$ produces the pooled graph data. The embedding values of $\boldsymbol{V}_{emb}$ are normalized using a softmax operation (σ). Note that in graph embed pooling, both the adjacency matrix and the vertices are transformed, as in Figure 4a.

$$\boldsymbol{V}_{emb}^{*} = \sigma(\boldsymbol{V}_{emb}) \tag{12}$$

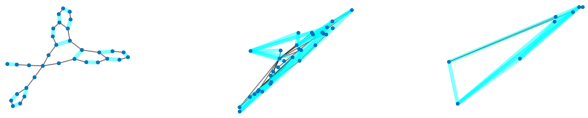
$$\boldsymbol{V}_{out} = \boldsymbol{V}_{emb}^{*T} \boldsymbol{V}_{in} \tag{13}$$

$$\boldsymbol{A}_{out} = \boldsymbol{V}_{emb}^{*T} \boldsymbol{A}_{in} \boldsymbol{V}_{emb}^{*} \tag{14}$$

There are two advantages to graph embed pooling. First, it flexibly takes input of any cardinality or structure and produces a fixed size output. This output can be of any cardinality N'. Second, this pooling is learned, so the output structure is the one that represents a reduced-dimension input structure in at least a locally optimal way, similar to other embedding methods. In our work, we use a special case of graph embed pooling where $N' = 1$ to produce a graph representation or Graph Fully-Connected (GFC) vector, an embedding we use in some of our experiments later in the paper.

One particular notion that must be realized is that visually this pooling does not resemble the intuition of average or max pooling in images, where the output signal would seem like a lower-resolution approximation of the input signal. In graph embed pooling, the output vertices are a convex combination of the input vertices, and they are fully connected. Further, this methodology induces self-connections. We use the simpler self-connections in $11 - 14$, because we notice it does not measurably change the resulting performance of the models. Figure 5 is an illustration of the pooling process, but the geometry of the figure should not be taken literally.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/JSTSP.2017.2726981, IEEE Journal of Selected Topics in Signal Processing



(a) Input graph. (b) Pool to 32 vertex. (c) Pool to 8 vertex.

Figure 5: Graph embed pooling representation as applied to a graph. (x, y) positions are synthesized for visual purposes. The graph embed pooling learned is applied resulting in (b) and then again in (c). Graphs show top 10% of edges and are not drawn to scale.

IV. Experiments

We apply our Graph-CNN model to five different problems. First, we compare Graph-CNN to traditional CNNs using the CIFAR-10 and ImageNet image classification datasets. Second, we perform gender classification based on Human Connectome Project (HCP) fMRI data. Third, we classify chemical compounds with the NCI1 and D&D datasets. Fourth, we classify facial expressions based on the Bosphorus 3D face dataset. Finally, we evaluate document classification with the Cora document datasets. These problems explore both homogeneous and heterogeneous datasets.

In each investigation, the Graph-CNNs are learned via stochastic gradient descent or Adam optimization [35] using back-propagation. For learning graph filters, the base learning rate is 0.01 and we use a momentum of 0.9 during updates. All architectures use ReLU activation function. Batch Normalization [34] was also used in all but the last layers of each architecture.

Most of the evaluations attempt graph classification, which attempts to apply a single label to an entire graph. The Document Classification task is a vertex classification task, which means the Graph-CNN attempts to apply a label to each individual test vertex based on the neighboring training vertices.

A. Image Classification

For a classification task, an image can be represented as a graph: a two-dimensional rectangular grid as shown in Figure 6. We run image classification experiments to give empirical evidence to the earlier claim that classical CNN can be modeled with Graph-CNN's with appropriate adjacency matrices. We use the CIFAR-10 [36] and the ImageNet dataset [37].

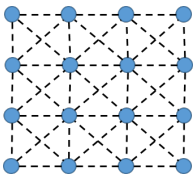


Figure 6: Representation of an image as a gridded graph. Vertices are pixels and lines are edge connections.

1) CIFAR-10: Every image in CIFAR is 32×32 pixels with RGB channels. They can each be represented as a graph with 1024 vertices, where every pixel is a vertex in the graph. We compare classification methods by learning the same number of parameters. Graph-CNN utilizes the 8-adjacency connection tensor described in Figure 3. The $\text{CNN}_{3\times3}$ is a standard CNN with 3×3 convolution filters. Table II shows classification accuracy on the CIFAR-10 dataset. We also compare spectral filters to CNNs and Graph-CNNs in the appendix. All models were trained for 60 epochs.

Table II: CIFAR-10 image graph classification results. nF is a convolution layer with n filters, $P/2$ is a max-pooling $/2$ layer and FC is a fully connected layer with 128 hidden vertices. All models have a FC-10 layer and a softmax loss layer.

#layers	Architecture	Accuracy (%)	
		Graph-CNN	$\text{CNN}_{3\times3}$
1	32F-FC	62.51	62.11
	64F-FC	64.12	63.4
2	32F-P/2-32F-FC	66.15	67.42
	32F-P/2-64F-FC	67.54	68.36
3	3×(32F-P/2)-FC	68.33	68.8

2) ImageNet: We also evaluate our method on the ImageNet 2012 image classification dataset. We use the ResNet-152 architecture [4] to demonstrate the compatibility between Graph-CNN and CNN. We replace the last residual layer (res5) with its Graph-CNN equivalent. We learn for 2e4 iterations and measure top-1 accuracy and use no data augmentation. With the unmodified ResNet we achieve 70.32% accuracy. With the ResNet modified with Graph-CNN layers, we achieve 70.02%. Though the accuracy we report is less than the actual ResNet performance, there is less than 1% disparity between the two values. Additionally, the learning curve in Figure 7 demonstrate that both CNN and Graph-CNN learn at the same rate.

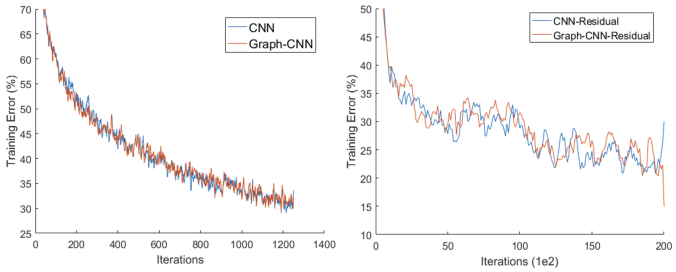


Figure 7: Comparison of training error for Graph-CNN and CNN architectures. **Left:** CIFAR-10 three layer architecture. **Right:** ImageNet with ResNet-152 architecture. Both the models show high compatibility between Graph-CNN and CNN.

The CIFAR and ImageNet experiments show that traditional CNN's can be modeled as Graph-CNN's. Since the formulation of the adjacency matrices allow for same number of learnable parameters in the convolution filters, the learning curves for both the datasets indicate high agreement between Graph-CNN and traditional CNN.

B. Human Connectome Project

The Human Connectome Project (HCP) [38] contains resting-state functional Magnetic Resonance Imaging (rsfMRI) data from each of 366 male and 454 female subjects. We seek to investigate whether we can reliably identify the subjects' biological sex from this data. We use the rsfMRI data preprocessed according to HCP's Minimal Preprocessing Pipeline [39]. We then subsequently apply denoising to the data according to the FIX protocol [40]. Each FIX-preprocessed rsfMRI is parcellated according to the Automatic Anatomical Labeling (AAL) atlas [41], which divides the brain into 90 cortical/subcortical regions and 26 cerebellar/vermis regions. Time series from voxels within each region are averaged to form a representative time series for each region.

We introduce four different weighted adjacency matrices modeling the functional connectivity of each subject based on computing similarities between the representative time series for each region: Fisher z-transformed Pearson correlation coefficients (CORR), Fisher z-transformed partial correlations (PCORR), and uniform (UNFM) and adaptive (ADPT) versions of the structure-aware affinity inference model for capturing subtle information distributed over discriminative feature subspaces [42]. These weighted adjacency matrices have the same vertices for each subject but different edge weights.

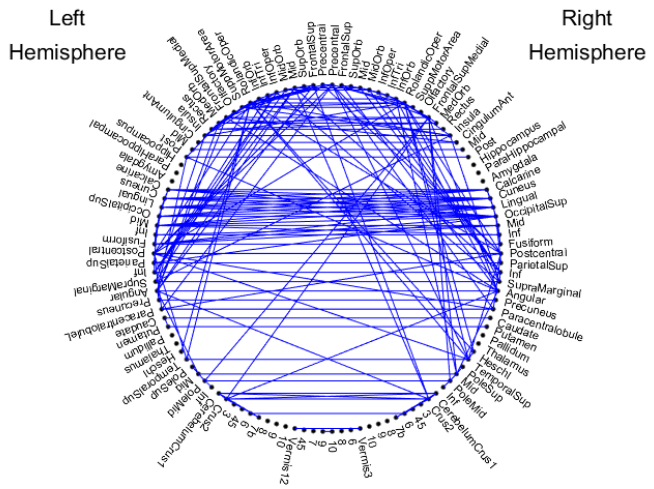


Figure 8: Graph-based representation of AAL atlas regions. Labels indicate the corresponding brain region, and blue lines correspond to the top 3% most correlated edges (edge weights) in the groupwise average CORR graph (across all male and female subjects).

Experiments were run with various types of similarity matrices. These are listed in Table III. It also reports results with combinations of the adjacency matrices. Multiple adjacency matrices are treated as edge features while defining the convolution filters. Since the number of training samples is less, complex structures formed through multiple adjacency matrices degrade the performance. The uniform representation of the similarity are listed in Table IV for this classification task. We observe that the best-performing model is the 2-layer Graph-CNN model with UNFM input matrices. We report a maximum accuracy of 83.78% in classifying male vs female using just fMRI graphs. These findings reveal that differences in intrinsic connectivity as measured with rs-fMRI exist between subjects. The Graph-CNN filters are capable of detecting and utilizing these differences for classification and gender prediction.

Table III: Comparison with different input graph matrices. All models have the architecture 2x(32F)-FC.

Input data	Accuracy(%)
CORR	74.51 ± 3.74
PCORR	74.75 ± 3.30
UNFM	$\mathbf{83.78} \pm 3.89$
ADPT	78.90 ± 3.18
CORR + PCORR + UNFM + ADPT	74.26 ± 2.49
UNFM + ADPT	78.90 ± 3.18

With increased number of imaging-based clinical studies on various diseases such as autism, attention-deficit hyperactivity disorder, etc. [43], the Graph-CNN seems a promising approach for distinguishing disease states from healthy brains on the basis of measurable differences in spontaneous activity. As the amount of available rs-fMRI data increases, learning based methods will be able to extract more meaningful information which can be used in complement with human clinical diagnoses to improve overall efficacy.

Table IV: Male-female classification using the UNFM matrices generated from the rsfMRI graphs. Training of all models is done for 400 epochs using 5-fold cross-validation. Parameters is the number of parameters in all layers before the last FC.

Architecture	Accuracy(%)	Parameters
FC	75.98 ± 2.05	–
FC-FC	78.05 ± 2.55	16384
32F-FC	81.09 ± 2.02	64
64F-FC	81.21 ± 3.59	128
2×(32F)-FC	$\mathbf{83.78} \pm 3.89$	128
3×(32F)-FC	82.68 ± 3.35	192
4×(32F)-FC	81.82 ± 3.64	256

C. Chemical Compound Classification

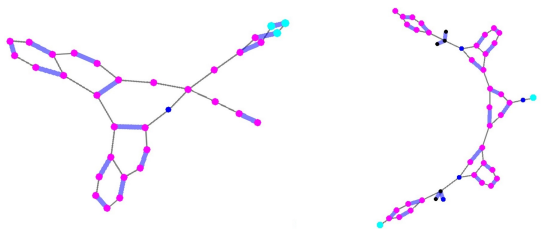


Figure 9: Samples of chemical compounds screened for activity against non-small cell lung cancer from the NCI1 dataset. Different colors and sizes represent different vertex features and edge features. **Left:** Negative sample. **Right:** Positive sample.

A popular task in the pharmaceutical industry is classification and retrieval of chemical compounds. The most common

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/JSTSP.2017.2726981, IEEE Journal of Selected Topics in Signal Processing

approach has been to use descriptors as inputs to a classifier. These descriptors can be individual fingerprints or substructures detected through a data mining step [44]. Recently, there has been some work to build an end-to-end model capable of learning the best descriptors or a suitable classifier [22], [21]. Deep learning based models were also proposed that extracts sub-structures by learning latent representations [45]. The main challenge is the inability to use a fully connected layer as a classifier since the size of input graphs is not constant. To address this, we used the graph embed pooling representation from Section III-D.

We use Graph-CNNs to address this problem and use the standard benchmark datasets – NCI1 and D&D, to compare classification performance. NCI1 is a balanced graph dataset of chemical compounds that are screened for activity against non-small cell lung cancer and ovarian cancer cell lines respectively [44]. D&D graphs are protein structures that can be classified into enzymes or non-enzymes categories [46]. These data are highly complex in terms of size and structure of individual samples. Each graph sample is heterogeneous and contains multiple adjacency matrices which indicate the presence of a specific bond type between two molecules. Detailed statistics and classification results on these datasets are listed in Table V. The Graph-CNN architectures achieve state-of-the-art performance compared with other recent approaches.

Table V: Comparison of classification accuracy for the chemical compound datasets. 5-hop DCNN # accuracies are over 3-fold cross validation; all other accuracies are reported over 10-fold cross validation. GK * and WL * results as reported in [22].

Data set	NCI1	D&D
Maximum graph size	111	5748
Average graph size	29.87	284.32
# Graphs	4110	1178
GK * [47]	62.28 ± 0.29	78.45 ± 0.26
WL * [48]	80.22 ± 0.51	77.95 ± 0.70
PSCN [22]	78.59 ± 1.89	77.12 ± 2.41
Deep GK [45]	80.31 ± 0.46	–
3×16F-3x32F-GFC32	83.69 ± 1.40	–
6×32F-GFC32	83.57 ± 1.99	–
2×64F-Pool32-FC256	84.08 ± 1.45	–
2×64F-Pool32-32F-Pool8-FC256	**84.62** ± 2.24	81.45 ± 2.87
2×64F-Pool32-32F-Pool8-64F-FC256	83.48 ± 1.36	–
2×64F-Pool32-64F-Pool8-FC256	84.35 ± 1.00	**81.88** ± 3.39
5-hop DCNN # [21]	62.61	–
2×64F-Pool32-32F-Pool8-FC256 #	**81.98** ± 0.76	–

D. Bosphorus 3D Facial Expressions

One common form of heterogeneous graph data is 3D mesh data. While sensors generally collect 3D data as point clouds [49] or images with a depth channel [50], these types of data can often be used to construct 3D meshes, posed as vertices and edges [51], [50]. Graph-CNN can perform object recognition tasks on 3D mesh data. This could be applied to autonomous driving applications that depend on LiDAR point clouds.

We test Graph-CNN on a toy 3D mesh classification problem. We use the facial expression-labeled faces in the Bosphorus 3D Face dataset [49]. 453 of the provided 3D faces are labeled with one of six facial expressions: Anger, Disgust, Fear, Happiness, Sadness, and Surprise. Bosphorus data is encoded as point cloud data, but with tools such as Meshlab [51] can be converted to 3D meshes. We attempt to classify these faces with Graph-CNN. RGB-D datasets were considered [50], [52], [53], [54], but RGB-D data is much less trivial to convert to a 3D mesh than point clouds. In addition, we were ultimately seeking a dataset that had focused samples neatly divided into a set of classes, similar to ImageNet [37].

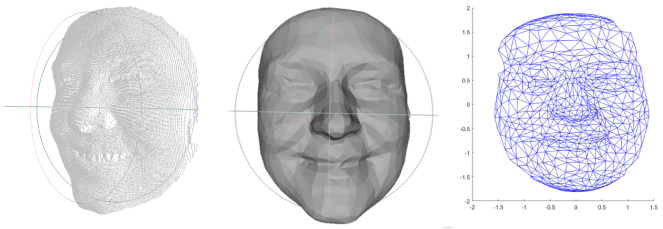


Figure 10: Different stages of Bosphorus face preprocessing. **Left:** Original point cloud. **Center:** Low-poly 3D mesh. **Right:** 2D projection of the front of 3D mesh that we input into Graph CNN.

It may seem curious to seasoned computer graphics experts that we formulated these 3D shapes as graphs, using only vertex and edge data. Other works treat 3D shapes as continuous manifolds, or discretely as a tuple $(\boldsymbol{V}, \boldsymbol{E}, \boldsymbol{F})$, where $\boldsymbol{E}$ is the set of edges and $\boldsymbol{F}$ is the set of faces [27], [28], [29], [30], [31], [32]. The purpose of this experiment was less to compete with the state-of-the-art in 3D shape retrieval and more to show the versatility of Graph-CNNs by applying them to a nontraditional graph dataset.

We discuss the preprocessing for this problem in the appendix in VII-C. We first evaluate our dataset by creating multiple Graph-CNN networks with different convolutional layers. Each Graph-CNN layer (labeled GConv in Table VI) learns 16 filters. Each network ends in a graph embed pooling layer as described in Section III-D, resulting in a new 32-vertex graph that is input into a fully-connected layer. Pooling was not used to reduce the data between graph convolutions. Table VI shows the results. The mean accuracy after 5-fold cross validation is displayed, plus or minus one standard deviation from the mean. Three Graph-CNN layers appears to underfit, and five appears to overfit the data. Since pooling only occurs at the end, and only as a method to fit heterogeneous data to a fixed fully-connected layer, the potential depth of this network is limited.

Table VI: Effect of Network Depth on Accuracy.

Architecture	Accuracy	Parameters
3× GConv16	$54.8 \pm 4.51\%$	8016
4× GConv16	$70.0 \pm 9.04\%$	10336
5× GConv16	$67.9 \pm 7.33\%$	12656

We also evaluate three other methods for comparison, also with 5-fold cross validation. First, we transform the 2D vertex data into PCA space and classify it with a Support Vector Machine (SVM). Second, we transform the 2D vertex data

into Locality-Preserving Projection space (LPP) [55], and pass that through an SVM. Finally we create a more typical 5 convolution layer CNN architecture (with slightly fewer total parameters) used for processing images. Each 3D face in Bosphorus has a corresponding image of that face, so we train the CNN on those images. Each layer in the CNN architecture is a 3×3 convolution, followed by a max-pooling with a stride of 2. The first 3 convolution layers have 16 filters, the last two have 8. At the end of these layers is a fully-connected layer with 6 outputs, and a softmax. Batch normalization, ReLUs, and the same training hyperparameters as Graph-CNN are used.

The final results of all these experiments are in Table VII. The SVM classifiers do not model the geometric data well. The traditional CNN with image data beats the Graph-CNN data, but its data does have some richer features. The input into the Graph-CNN does not encode color or depth. In addition, our network has some limitations. Our receptive field may be limited due to the shallowness of the network and the lack of pooling. Efforts to increase this field, such as pooling, increased depth, or atrous filtering techniques as in [56] would likely yield superior results. Also, no data augmentation was attempted. Augmenting the dataset with affine transforms of the vertices could address the small size of the dataset. Regardless, we have shown that Graph-CNNs are capable of processing 3D mesh data, and have plenty of room for iteration to become competitive.

Table VII: Bosphorus Expression Results.

Method	Accuracy (%)	Parameters
PCA+SVM	28.0	283
LPP+SVM	26.2	5
CNN+Images	**82.2**	8032
Graph-CNN	70.0	10336

E. Document Classification

A common form of graph-formatted data is a network of documents. For example, scientific documents in a database are related to one another through citations and references. The entire network is a single large graph, rather than a set of disparate graphs. Each document is a vertex in the graph with a certain features and a citation is an edge from one vertex to the other. Administrators of such large networks may desire to automatically label documents according to their relationship to the rest of the literature. We demonstrate the use of Graph-CNN architecture to model such a vertex classification task. Since the data is organized as a single graph, a label mask of zeros is applied on the test vertices during training. Hence, the loss layer does not back-propagate for the test vertices. At test time, only the test labels are used to compute the accuracy.

Evaluation of Graph-CNN for such an application is focused on the Cora dataset [57], a large network of scientific publications connected through citations. The vertex features in this case are binary word vectors that indicate the presence of a word from a dictionary of 1433 unique words. There are 2708 publications classified under seven different categories- Case Based, Genetic Algorithms, Neural Networks, Probabilistic Methods, Reinforcement Learning, Rule Learning and Theory. There is an edge connection from a cited article to a citing article and another edge connection from a citing article to a cited article. These edge features are binary representations. We perform cross validations with three different settings to form the training and test set for fair comparison with other recent studies.

Table IX lists document classification accuracies compared with the recent approaches. Our Graph-CNN architecture ($2\times$48F-7F) contains three Graph-CNN layers: first two layers with 48 filters and third layer with seven filters. The last Graph-CNN layer computes the prediction of each vertex. We then expand this network by adding 0-hop filters after each Graph-CNN. Dropout was also added before each 0-hop filter. We note our performance on the 1000 test* is not state-of-the-art. This test split has only 20 training samples per class and our method overfits, achieving 100% classification accuracy on the training set after only three iterations. We observed that with deeper architectures (3x48F-7F), the network quickly overfits on the training set and the performance degrades on the test set. We report these in Table VIII. For the model with Dropout and $0-$hop, the highest accuracy that we obtain are 89.14% and 91.51% on 3- and 10- folds, respectively. All models were trained using Adam optimization [35] and identical hyperparameters. The BatchNorm layers were modified to no longer use running average for mean and variance since there is only a single large sample graph.

Table VIII: Cora document classification accuracy.

Method	Accuracy (%)	
	3-fold	10-fold
$2\times$48F-7F	84.30 $\pm$ 1.66	87.11 $\pm$ 1.84
+ Dropout, $0-$hop	**87.55** $\pm$ 1.38	**89.18** $\pm$ 1.96
3x48F-7F	84.86 $\pm$ 0.42	87.44 $\pm$ 1.83

Table IX: Cora document classification accuracy comparison. 3-fold and 10-fold are cross validation tests, and "1000 test" is a popular split in the literature where 1000 samples are used in the test set, and the rest are used in training. "1000 test*" refers to an experiment split using only 20 samples from each class for training, and the same 1000 samples for testing.

Method	Split	Accuracy
Yang [58]	1000 test	75.7
Kipf [10]	1000 test*	81.5
Monti [31]	1000 test*	**81.69**
DCNN [21]	3-fold	86.77
Ours	1000 test*	76.3
Ours	1000 test	**86.56** $\pm$ 0.68
Ours	3 fold	**87.55** $\pm$ 1.38
Ours	10 fold	**89.18** $\pm$ 1.96

V. Computational Complexity

For a system to be a feasible solution to deep learning applications it must perform well not only in terms of accuracy but also in terms of required computational resources. It must also be efficiently computed using general purpose computation resources like CPUs and GPUs. The worst case scenario of our system is an application with fully-connected graph samples,

where each sample has N vertices, L adjacencies, C vertex features, and F filters. The time and space complexity for (10) is $O(N^2CLF)$ and $O(N^2CF+N^2L)$ respectively. It is trivial to see that the neighborhood of a node can be computed prior to applying specific filter weights. This modification is seen in (15) and (16), where $\mathcal{N}^{(\ell)}$ is computed once per sample with $\mathcal{N}$ in $\mathbb{R}^{N\times CL}$ and h (the weight matrix) in $\mathbb{R}^{CL\times F}$. Equation (15) is analogous to an *im2col* operation commonly used in CNN computations where input features are organized before a filter is applied. In fact, for images, the result is the same as an *im2col* operation. If neighborhood is computed before applying the filter, the time and space complexity becomes $O(N^2CL + NCLF)$ and $O(N^2L + NCL + NF)$ respectively. Note that (10) has the benefit of having a reusable $\boldsymbol{H}$ for all samples with the same adjacency matrices (e.g. images) reducing the complexity of batched operations on homogeneous problems.

$$\mathcal{N}^{(\ell)} = \boldsymbol{A}_\ell \boldsymbol{V}_{in} \tag{15}$$

$$\boldsymbol{V}_{out} = \mathcal{N}h \tag{16}$$

When using sparsely connected graphs, the adjacency matrices can be sparsely represented. Doing so reduces time and space complexity to $O(EC+NCLF)$ and $O(E+NCL+NF)$ respectively where E is the number of non-zero edge features ($E < N^2L$). For a CNN, each vertex has at most one neighbor in each adjacency matrix so $E \approx LN$ and the time complexity of Graph-CNN is reduced to $O(NCLF)$ which is equivalent to that of a CNN.

VI. Conclusion

Many types of graph data are heterogeneous and cannot be processed using traditional spectral convolutional filtering techniques. We introduce a general purpose Graph-CNN paradigm that offers the same breakthrough benefits currently only afforded to homogeneous data. Similar to traditional CNN architectures, Graph-CNNs operate directly in the spatial domain to generate semantically rich features. Our model operates on both homogeneous and heterogeneous data, learning properties from both graph vertices and edges. We establish that traditional CNNs are a subset of Graph-CNN for image data. We have proposed a graph embed pooling method that can reduce dimensionality of graphs throughout a network. We have shown results on graphs of fixed size and connections (images), graphs with fixed size but variable connections (rsfMRI), graphs with varying size and connections (chemical compounds, facial expression recognition), and large single-sample graphs (document classification).

Future work involves extending the flexibility and applications of our Graph-CNN method. We seek to increase the depth and receptive field of our networks through more sophisticated pooling methods, residual network formulations, and atrous filtering. The mechanics of these Graph-CNNs should be analyzed through filter visualization and more in-depth study of the distributions of graph data across the network. The theory should be expanded to enable filtering of edges as well as vertices. We anticipate methods such as Graph-CNN will be applied far and wide to bring the benefits of automatic feature learning to graph problems throughout the literature.

VII. Appendix

A. Graph-CNN is a Superset of CNNs

In this section we prove that Graph-CNN is a superset of CNNs.

A 3×3 convolutional filtering operation on a single 3×3 neighborhood can be written as in (17).

$$\begin{aligned} V'_{i,j} &= h_0 \cdot V_{i,j} \\ &+ h_1 \cdot V_{i-1,j-1} + h_2 \cdot V_{i-1,j} \\ &+ h_3 \cdot V_{i-1,j+1} + h_4 \cdot V_{i,j-1} \\ &+ h_5 \cdot V_{i,j+1} + h_6 \cdot V_{i+1,j-1} \\ &+ h_7 \cdot V_{i+1,j} + h_8 \cdot V_{i+1,j+1} \end{aligned} \tag{17}$$

Vertex features in Graph-CNN are represented as a one-dimensional vector. To pose two-dimensional pixels as a vertex feature vector, we can use the following mapping from $(i, j) \to n$.

$$n = i \cdot W + j \tag{18}$$

1	2	3
4	5	6
7	8	9

1	2	3	4	5	6	7	8	9

Where W is the horizontal width of the image. Equation (17) can be reposed in this indexing scheme as (19).

$$\begin{aligned} V'_n &= h_0 \cdot V_n \\ &+ h_1 \cdot V_{n-W-1} + h_2 \cdot V_{n-W} \\ &+ h_3 \cdot V_{n-W+1} + h_4 \cdot V_{n-1} \\ &+ h_5 \cdot V_{n+1} + h_6 \cdot V_{n+W-1} \\ &+ h_7 \cdot V_{n+W} + h_8 \cdot V_{n+W+1} \end{aligned} \tag{19}$$

Next, we define a series of adjacency matrices, each one only containing the pixel-wise relationships of a certain direction, as illustrated in the right-side image in Figure 3.

$$\uparrow \boldsymbol{A}_{i,j} = \begin{cases} 1, & \text{if } i - W = j \\ 0, & \text{otherwise} \end{cases} \tag{20}$$

$$\downarrow \boldsymbol{A}_{i,j} = \uparrow \boldsymbol{A}_{j,i} = \uparrow \boldsymbol{A}^T_{i,j} = \begin{cases} 1, & \text{if } i + W = j \\ 0, & \text{otherwise} \end{cases} \tag{21}$$

$$\leftarrow \boldsymbol{A}_{i,j} = \begin{cases} 1, & \text{if } i - 1 = j \\ 0, & \text{otherwise} \end{cases} \tag{22}$$

$$\rightarrow \boldsymbol{A}_{i,j} = \leftarrow \boldsymbol{A}_{j,i} = \begin{cases} 1, & \text{if } i + 1 = j \\ 0, & \text{otherwise} \end{cases} \tag{23}$$

$$\nwarrow \boldsymbol{A}_{i,j} = \begin{cases} 1, & \text{if } i - W - 1 = j \\ 0, & \text{otherwise} \end{cases} \tag{24}$$

$$\searrow \boldsymbol{A}_{i,j} = \nwarrow \boldsymbol{A}_{j,i} = \begin{cases} 1, & \text{if } i + W + 1 = j \\ 0, & \text{otherwise} \end{cases} \tag{25}$$

$$\nearrow \boldsymbol{A}_{i,j} = \begin{cases} 1, & \text{if } i - W + 1 = j \\ 0, & \text{otherwise} \end{cases} \tag{26}$$

$$\swarrow \boldsymbol{A}_{i,j} = \nearrow \boldsymbol{A}_{j,i} = \begin{cases} 1, & \text{if } i + W - 1 = j \\ 0, & \text{otherwise} \end{cases} \tag{27}$$

$$\boldsymbol{I} = \begin{cases} 1, & \text{if } i = j \\ 0, & \text{otherwise} \end{cases} \tag{28}$$

These matrices are now indicator functions to clarify which filter taps apply to which pixels in a given pixel neighborhood. This means that we can remove the indices entirely from (19) and pose it as a matrix operation, as in (29).

$$\begin{aligned} \boldsymbol{V}' = & h_0 \cdot \boldsymbol{I}\boldsymbol{V} \\ & + h_1 \cdot \nwarrow \boldsymbol{A}\boldsymbol{V} + h_2 \cdot \uparrow \boldsymbol{A}\boldsymbol{V} \\ & + h_3 \cdot \nearrow \boldsymbol{A}\boldsymbol{V} + h_4 \cdot \leftarrow \boldsymbol{A}\boldsymbol{V} \\ & + h_5 \cdot \rightarrow \boldsymbol{A}\boldsymbol{V} + h_6 \cdot \swarrow \boldsymbol{A}\boldsymbol{V} \\ & + h_7 \cdot \downarrow \boldsymbol{A}\boldsymbol{V} + h_8 \cdot \searrow \boldsymbol{A}\boldsymbol{V} \end{aligned} \tag{29}$$

The separate adjacency matrices can be packed into a tensor $\boldsymbol{\mathcal{A}}$ made up of the slices $\{\boldsymbol{I}, \nwarrow \boldsymbol{A}, \uparrow \boldsymbol{A}, \nearrow \boldsymbol{A}, \leftarrow \boldsymbol{A}, \rightarrow \boldsymbol{A}, \swarrow \boldsymbol{A}, \downarrow \boldsymbol{A}, \searrow \boldsymbol{A}\}$. At this point it can be processed by equations (9) and (10).

B. Spectral Domain

In [8], experiments on ImageNet classification are performed to compare learning for spectral graph convolutions and traditional image convolutions. Their results show that spectral graph convolutions learn more quickly than image convolutions at first, but ultimately converge to the same result.

In this section, we compare image convolutions (CNN), spectral convolutions (Spectral), and our Graph-CNN (GCNN). We trained a single-layer convolutional neural network of each of the three types on the CIFAR-10 dataset [36]. Each network learned 32 filters, and was followed by two fully-connected layers of equal size. The resulting learning curve is shown as in Figure 11.

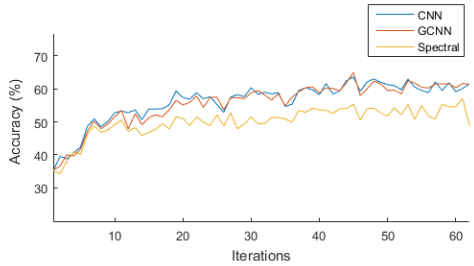


Figure 11: Comparison of accuracy for CNN, Graph-CNN, and Graph-Spectral methods trained on CIFAR-10 dataset. All architectures were single layer convolution with 32 filters followed by two fully connected layers.

Since we proved in VII-A that an image CNN can be exactly represented by Graph-CNN, these two methods understandably get the same results. However, our spectral performance in this test case is about 10% lower than the other methods. These findings contrast with the results of [8] which get identical performance between spectral and image convolutions. Several hyperparameters were explored, but it is possible further hyperparameter tuning could improve the Graph-Spectral performance.

C. Preparing the Bosphorus Model

One of the biggest challenges with this dataset as a testbed for Graph-CNN is its small number of samples, each with a very high dimensionality (tens of thousands of points in 3D space). This means that the data, without upfront reduction, can rapidly lead to overfitting and poor generalization in a network. In addition, adjacency matrices built from this many points would be expensive to compute. For this experiment we performed extensive data reduction, attempting to cut away as much detail from these point clouds to enable effective and efficient learning.

First, each face is processed as a raw point cloud. Outlier points are removed and a mesh is created from the remainder using Meshlab [51]. The mesh is simplified to a low dimension (roughly 1052 vertices) using Meshlab's mesh simplification algorithms. The mesh is then converted into a V and A matrix. The features of V are the X, Y, and Z coordinates in 3D Euclidean space of each vertex. The edges of A are 1 if the edge exists in the mesh and 0 if it does not. We did not use a distance measure such as Euclidean distance because that information would be implicit in the vertex features. To increase the number of edge features, we partitioned the edges into bins based on the angle of the vector formed by that edge. Edge ij forms a vector $\vec{ij}$ defined by subtracting i from j. Then the sign of each component (ij_X,ij_Y, and ij_Z) from that vector is used to separate into bins: ($ij_X \geq 0$,$ij_Y \geq 0$, $ij_Z \geq 0$) is one bin, ($ij_X \geq 0$,$ij_Y \geq 0$, $ij_Z < 0$) is another bin, and so on for all eight 3D octants.

Next we attempt to further reduce dimensionality by removing the back of the head formed by the mesh creation algorithm. These vertices and edges are generated in an attempt to create a closed 3D figure, but they provide no information for discerning facial expression. To do this programmatically, we normalize V for a given sample to be 0 mean, unit variance. Vertices with negative Z components are removed from V and A. Finally, we project the face onto a 2D-plane by removing the Z feature from V. The 8 adjacency matrices are still preserved, though the Z feature is removed. This much reduced model of the face retains useful information for identifying expression. Figure 10 illustrates the transformation.

VIII. Acknowledgements

We would like to thank the National GEM Consortium for funding one of the student authors through the GEM Fellowship.

REFERENCES

[1] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," in *Advances in neural information processing systems*, 2012, pp. 1097–1105.

[2] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," *arXiv preprint arXiv:1409.1556*, 2014.

[3] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich, "Going deeper with convolutions," in *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2015.

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

[5] S. Ren, K. He, R. Girshick, and J. Sun, "Faster R-CNN: Towards real-time object detection with region proposal networks," in *Advances in Neural Information Processing Systems (NIPS)*, 2015.

[6] J. Redmon, S. Divvala, R. Girshick, and A. Farhadi, "You only look once: Unified, real-time object detection," *arXiv preprint arXiv:1506.02640*, 2015.

[7] O. Abdel-Hamid, A.-R. Mohamed, H. Jiang, L. Deng, G. Penn, and D. Yu, "Convolutional neural networks for speech recognition," *IEEE/ACM Trans. Audio, Speech and Lang. Proc.*, vol. 22, no. 10, pp. 1533–1545, Oct. 2014. [Online]. Available: http://dx.doi.org/10.1109/TASLP.2014.2339736

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

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

[10] T. N. Kipf and M. Welling, "Semi-supervised classification with graph convolutional networks," in *ArXiv e-prints*, vol. 1609, Conference Proceedings. [Online]. Available: http://adsabs.harvard.edu/abs/2016arXiv160902907K

[11] S. I. Ktena, S. Parisot, E. Ferrante, M. Rajchl, M. Lee, B. Glocker, and D. Rueckert, "Distance metric learning using graph convolutional networks: Application to functional brain networks," *arXiv preprint arXiv:1703.02161*, 2017.

[12] M. Defferrard, X. Bresson, and P. Vandergheynst, "Convolutional neural networks on graphs with fast localized spectral filtering," in *Advances in Neural Information Processing Systems 29*, D. D. Lee, M. Sugiyama, U. V. Luxburg, I. Guyon, and R. Garnett, Eds. Curran Associates, Inc., Conference Proceedings, pp. 3844–3852.

[13] A. Sandryhaila and J. M. Moura, "Discrete signal processing on graphs," *Signal Processing, IEEE Transactions on*, vol. 61, no. 7, pp. 1644–1656, 2013.

[14] F. Petroski Such, "Graphcnn," https://github.com/fps7806/Graph-CNN, 2017.

[15] M. Edwards and X. Xie, "Graph convolutional neural network," in *British Machine Vision Conference*, R. C. Wilson, E. R. Hancock, and W. A. P. Smith, Eds. BMVA Press, Conference Proceedings. [Online]. Available: http://www.bmva.org/bmvc/2016/papers/paper114/paper114.pdf

[16] U. von Luxburg, "A tutorial on spectral clustering," *Statistics and Computing*, vol. 17, no. 4, pp. 395–416, 2007. [Online]. Available: http://dx.doi.org/10.1007/s11222-007-9033-z

[17] I. S. Dhillon, Y. Guan, and B. Kulis, "Weighted graph cuts without eigenvectors a multilevel approach," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 29, no. 11, pp. 1944–1957, 2007.

[18] D. I. Shuman, B. Ricaud, and P. Vandergheynst, "Vertex-frequency analysis on graphs," *Applied and Computational Harmonic Analysis*, vol. 40, no. 2, pp. 260–291, 2016. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S1063520315000214

[19] Y. Seo, M. Defferrard, P. Vandergheynst, and X. Bresson, "Structured sequence modeling with graph convolutional recurrent networks," in *ArXiv e-prints*, vol. 1612, Conference Proceedings. [Online]. Available: http://adsabs.harvard.edu/abs/2016arXiv161207659S

[20] Y. Lecun, L. Bottou, Y. Bengio, and P. Haffner, "Gradient-based learning applied to document recognition," *Proceedings of the IEEE*, vol. 86, no. 11, pp. 2278–2324, 1998.

[21] J. Atwood and D. Towsley, "Diffusion-convolutional neural networks," *arXiv preprint arXiv:1511.02136*, 2015.

[22] M. Niepert, M. Ahmed, and K. Kutzkov, "Learning convolutional neural networks for graphs," in *Proceeding of the 33rd International Conference on Machine Learning*, 2016, pp. 2014—2023.

[23] D. K. Duvenaud, D. Maclaurin, J. Aguilera-Iparraguirre, R. Gómez-Bombarelli, T. Hirzel, A. Aspuru-Guzik, and R. P. Adams, "Convolutional networks on graphs for learning molecular fingerprints," *CoRR*, vol. abs/1509.09292, 2015. [Online]. Available: http://arxiv.org/abs/1509.09292

[24] Y. Li, D. Tarlow, M. Brockschmidt, and R. S. Zemel, "Gated graph sequence neural networks," *CoRR*, vol. abs/1511.05493, 2015. [Online]. Available: http://arxiv.org/abs/1511.05493

[25] F. Scarselli, M. Gori, A. C. Tsoi, M. Hagenbuchner, and G. Monfardini, "The graph neural network model," *Neural Networks, IEEE Transactions on*, vol. 20, no. 1, pp. 61–80, 2009.

[26] B. Perozzi, R. Al-Rfou, and S. Skiena, "Deepwalk: Online learning of social representations," in *Proceedings of the 20th ACM SIGKDD*. ACM, 2014, pp. 701–710.

[27] J. Masci, D. Boscaini, M. M. Bronstein, and P. Vandergheynst, "Geodesic convolutional neural networks on riemannian manifolds," in *2015 IEEE International Conference on Computer Vision Workshop (ICCVW)*, Conference Proceedings, pp. 832–840.

[28] D. Boscaini, J. Masci, S. Melzi, M. M. Bronstein, U. Castellani, and P. Vandergheynst, "Learning class-specific descriptors for deformable shapes using localized spectral convolutional networks," in *Proceedings of the Eurographics Symposium on Geometry Processing*. 2853911: Eurographics Association, Conference Proceedings, pp. 13–23.

[29] D. Boscaini, J. Masci, E. Rodolà, M. M. Bronstein, and D. Cremers, "Anisotropic diffusion descriptors," *Computer Graphics Forum*, vol. 35, no. 2, pp. 431–441, 2016. [Online]. Available: http://dx.doi.org/10.1111/cgf.12844

[30] D. Boscaini, J. Masci, E. Rodolà, and M. Bronstein, "Learning shape correspondence with anisotropic convolutional neural networks," in *Advances in Neural Information Processing Systems 29*, Conference Proceedings.

[31] F. Monti, D. Boscaini, J. Masci, E. Rodolà, J. Svoboda, and M. M. Bronstein, "Geometric deep learning on graphs and manifolds using mixture model cnns," *arXiv preprint arXiv:1611.08402*, 2016.

[32] M. M. Bronstein, J. Bruna, Y. LeCun, A. Szlam, and P. Vandergheynst, "Geometric deep learning: going beyond euclidean data," in *ArXiv e-prints*, vol. 1611, Conference Proceedings. [Online]. Available: http://adsabs.harvard.edu/abs/2016arXiv161108097B

[33] X. Glorot and Y. Bengio, "Understanding the difficulty of training deep feedforward neural networks," in *JMLR W&CP: Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics (AISTATS 2010)*, vol. 9, May 2010, pp. 249–256.

[34] S. Ioffe and C. Szegedy, "Batch normalization: Accelerating deep network training by reducing internal covariate shift," *arXiv preprint arXiv:1502.03167*, 2015.

[35] D. Kingma and J. Ba, "Adam: A method for stochastic optimization," *arXiv preprint arXiv:1412.6980*, 2014.

[36] A. Krizhevsky and G. Hinton, "Learning multiple layers of features from tiny images," 2009.

[37] O. Russakovsky, J. Deng, H. Su, J. Krause, S. Satheesh, S. Ma, Z. Huang, A. Karpathy, A. Khosla, M. Bernstein *et al.*, "Imagenet large scale visual recognition challenge," *International Journal of Computer Vision*, vol. 115, no. 3, pp. 211–252, 2015.

[38] "Human connectome project," https://db.humanconnectome.org.

[39] M. F. Glasser *et al.*, "The minimal preprocessing pipelines for the Human Connectome Project," *Neuroimage*, vol. 80, pp. 105–124, 2013.

[40] G. Salimi-Khorshidi, G. Douaud, C. F. Beckmann, M. F. Glasser, L. Griffanti, and S. M. Smith, "Automatic denoising of functional MRI data: combining independent component analysis and hierarchical fusion of classifiers," *Neuroimage*, vol. 90, pp. 449–468, 2014.

[41] N. Tzourio-Mazoyer, B. Landeau, D. Papathanassiou, F. Crivello, O. Etard, N. Delcroix, B. Mazoyer, and M. Joliot, "Automated anatomical labeling of activations in SPM using a macroscopic anatomical parcellation of the MNI MRI single-subject brain," *Neuroimage*, vol. 15, no. 1, pp. 273–289, 2002.

[42] X. Zhu, C. Change Loy, and S. Gong, "Constructing robust affinity graphs for spectral clustering," in *Proc. CVPR*, 2014, pp. 1450–1457.

[43] S. Vergun, A. Deshpande, T. B. Meier, J. Song, D. L. Tudorascu, V. A. Nair, V. Singh, B. B. Biswal, M. E. Meyerand, R. M. Birn *et al.*, "Characterizing functional connectivity differences in aging adults using machine learning on resting state fmri data," *Frontiers in computational neuroscience*, vol. 7, p. 38, 2013.

[44] N. Wale, I. A. Watson, and G. Karypis, "Comparison of descriptor spaces for chemical compound retrieval and classification," *Knowledge and Information Systems*, vol. 14, no. 3, pp. 347–375, 2008.

[45] P. Yanardag and S. Vishwanathan, "Deep graph kernels," in *Proceedings of the 21th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. ACM, 2015, pp. 1365–1374.

[46] P. D. Dobson and A. J. Doig, "Distinguishing enzyme structures from non-enzymes without alignments," *Journal of molecular biology*, vol. 330, no. 4, pp. 771–783, 2003.
[47] N. Shervashidze, S. Vishwanathan, T. Petri, K. Mehlhorn, and K. M. Borgwardt, "Efficient graphlet kernels for large graph comparison." in *AISTATS*, vol. 5, 2009, pp. 488–495.
[48] N. Shervashidze, P. Schweitzer, E. J. v. Leeuwen, K. Mehlhorn, and K. M. Borgwardt, "Weisfeiler-lehman graph kernels," *Journal of Machine Learning Research*, vol. 12, no. Sep, pp. 2539–2561, 2011.
[49] A. Savran, N. Alyüz, H. Dibeklioğlu, O. Çeliktutan, B. Gökberk, B. Sankur, and L. Akarun, "Biometrics and identity management," B. Schouten, N. C. Juul, A. Drygajlo, and M. Tistarelli, Eds. Berlin, Heidelberg: Springer-Verlag, 2008, ch. Bosphorus Database for 3D Face Analysis, pp. 47–56. [Online]. Available: $http://dx.doi.org/10.1007/978-3-540-89991-4_6$
[50] S. Choi, Q.-Y. Zhou, S. Miller, and V. Koltun, "A large dataset of object scans," *arXiv:1602.02481*, 2016.
[51] P. Cignoni, M. Corsini, and G. Ranzuglia, "Meshlab: an open-source 3d mesh processing system," *ERCIM News*, no. 73, pp. 45–46, April 2008. [Online]. Available: http://vcg.isti.cnr.it/Publications/2008/CCR08
[52] P. K. Nathan Silberman, Derek Hoiem and R. Fergus, "Indoor segmentation and support inference from rgbd images," in *ECCV*, 2012.
[53] A. Janoch, S. Karayev, J. Yangqing, J. T. Barron, M. Fritz, K. Saenko, and T. Darrell, "A category-level 3-d object dataset: Putting the kinect to work," in *2011 IEEE International Conference on Computer Vision Workshops (ICCV Workshops)*, Conference Proceedings, pp. 1168–1174.
[54] K. Lai, L. Bo, X. Ren, and D. Fox, "A large-scale hierarchical multi-view rgb-d object dataset," in *IEEE International Conference on Robotics and Automation (ICRA)*, Conference Proceedings.
[55] X. He and P. Niyogi, "Locality preserving projections," in *Neural information processing systems*, vol. 16. MIT, 2004, p. 153.
[56] L. Chen, G. Papandreou, I. Kokkinos, K. Murphy, and A. L. Yuille, "Deeplab: Semantic image segmentation with deep convolutional nets, atrous convolution, and fully connected crfs," *CoRR*, vol. abs/1606.00915, 2016. [Online]. Available: http://arxiv.org/abs/1606.00915
[57] P. Sen, G. M. Namata, M. Bilgic, L. Getoor, B. Gallagher, and T. Eliassi-Rad, "Collective classification in network data," *AI Magazine*, vol. 29, no. 3, pp. 93–106, 2008.
[58] Z. Yang, W. Cohen, and R. Salakhutdinov, "Revisiting semi-supervised learning with graph embeddings," *arXiv preprint arXiv:1603.08861*, 2016.

Felipe Petroski Such received his BS and MS in Computer Engineering at the Rochester Institute of Technology, New York, USA, in 2017. His interest in machine intelligence ranges from hardware acceleration to applicable software. During his time at RIT he worked as a Teaching Assistant for deep learning as well as a Research Assistant where he developed state-of-the-art handwriting recognition and graph filtering software.

Shagan Sah obtained a Bachelors in Engineering degree from University of Pune, India and a M.S. in Imaging Science from Rochester Institute of Technology (RIT), USA with the aid of an RIT Graduate Scholarship. He is currently a Ph.D. candidate in the Center for Imaging Science at RIT. His current work and interests lie in the intersection of machine learning, natural language processing and computer vision applications in image and video understanding. In the past, he has worked at Motorola, Xerox-PARC and Cisco Systems.

Miguel Dominguez earned a B.S. in Computer Science and Engineering from University of Toledo in 2012 and an M.S. in Electrical Engineering from Rochester Institute of Technology (RIT) in 2016. He is currently a GEM Fellow pursuing a Ph.D. in Engineering at RIT. His research interests include deep learning, computer vision, graph signal processing, and audio and speech processing. He is inspired to do research by his Blessed Mother.

Suhas Pillai received his M.Sc in Computer Science from Rochester Institute of Technology in 2017. Since then he has joined TeraDeep as a Machine Learning Engineer, developing algorithms for detection of objects in satellite imagery. His research interests include speech and pattern recognition and computer vision.

Chao Zhang is an imaging science student at Rochester Institute of Technology and Autism Developmental and Medicine Institute of Geisinger Health System. His research focus is on functional magnetic resonance image (fMRI), applying data mining to associate fMRI metrics with subject measures.

Andrew Michael , PhD received his BE in Electronics and Communication Engineering (First Class) in 2001 at the National Institute of Technology, India. He received his MS in Electrical Engineering and PhD in Imaging science at the Rochester Institute of Technology, New York, USA, in 2004 and 2009, respectively. In 2009 he joined the Mind Research Network as research scientist and program manager where he researched on data fusion methods to identify brain markers of psychiatric illnesses. In 2013 he joined Geisinger Health Systems in Pennsylvania as Assistant Professor and the founding director of the Neuroimaging Analytics Laboratory. At Geisinger, Dr. Michael contributes his signal processing, image analytics and machine learning skills to the analyses of brain imaging data from multiple imaging modalities.

Nathan D. Cahill is the Associate Dean for Industrial Partnerships in the College of Science at RIT, where he is also an Associate Professor of Mathematical Sciences, a Graduate Program Faculty member of the Center for Imaging Science, and Director of the Image Computing and Analysis Laboratory. Prior to joining the RIT faculty in 2009, he spent 13 years in the Kodak Research Labs and at Carestream Health, during which time he was an inventor on 26 granted U.S. Patents and earned a DPhil in Engineering Science from the University of Oxford. His research is on the development of mathematical models and computational algorithms for solving problems in computer vision, medical imaging, and remote sensing.

Raymond Ptucha is an Assistant Professor in Computer Engineering and Director of the Machine Intelligence Laboratory at Rochester Institute of Technology. His research specializes in machine learning, computer vision, and robotics. Ray was a research scientist with Eastman Kodak Company where he worked on computational imaging algorithms and was awarded 31 U.S. patents with another 19 applications on file. He graduated from SUNY/Buffalo with a B.S. in Computer Science and a B.S. in Electrical Engineering. He earned a M.S. in Image Science from RIT. He earned a Ph.D. in Computer Science from RIT in 2013. Ray was awarded an NSF Graduate Research Fellowship in 2010 and his Ph.D. research earned the 2014 Best RIT Doctoral Dissertation Award. Ray is a passionate supporter of STEM education and is an active member of his local IEEE chapter and FIRST robotics organizations.