Project Report : CS 7643
Graph Convolution Networks Performing Finite Element Analysis

Hayden Cornwell
Georgia Tech
hcornwell3@gatech.edu

Brandon Wetzel
Georgia Tech
bwetzel6@gatech.edu

Parth Joshi
Georgia Tech
pjoshi79@gatech.edu

Abstract

Numerical methods, particularly finite element analysis (FEA), have greatly assisted engineers in innovating across all industries. Although FEA provides the flexibility to model complex shapes and loads, there is a computational ceiling that results from models containing too many elements. Deep learning may be a way to reduce the run time and memory of large models. In this paper, we propose a way to predict FEA analysis results by training graph convolution networks (GCNs) on a previous set of FEA results. Using 4 different GCN architectures (GCNConv, TAGConv, MFConv, and ChebConv) the mean square error (MSE) and visual results were compared against one another. ChebConv, a spectral GCN method, demonstrated the best results with a test MSE of 0.0015.

1. Introduction

Numerical methods have been an integral part in advancing technology by allowing for complex partial differential equations to be solved through discretization [22]. A particular area of interest for this application was engineering simulation due to the complex shapes of components and assemblies. Finite element analysis (FEA) and computational fluid dynamics (CFD) are the most used methods of engineering simulation that work by splitting up a shape into many smaller elements with simple shapes. Since each simple shape can be solved for analytically, the entire shape can be solved by connecting all the elements, applying boundary conditions, and loads [21, 24, 15]. This technique was critical in determining deformations and stresses in complex parts for the design of the space shuttle, which is what lead to the development of the FEA software, and first pieces of software altogether, NASTRAN (Nasa Structural Analysis) [18].

Although FEA has allowed engineers and scientists to flexibly predict part performance, there are still issues that limit its applicability. Firstly, as the number of elements in a model increases so does the run time and memory of a simulation. The complexity of an FEA model generally is $O(N^2)$, where N is the number of elements in the model. [12]. Even considering Moore's law, models of large assemblies (as seen in Figure 1) and parts will never be feasible using this method. Other issues and considerations with FEA include numerical stability, model accuracy, and convergence with varying time step, element type, and mesh refinement.

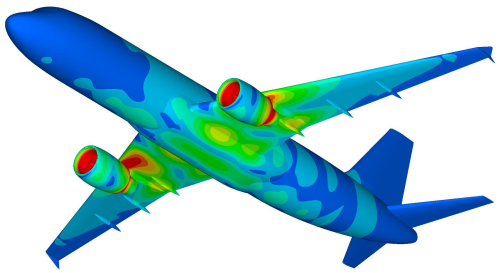

Figure 1. Example of a large FEA model of an airplane showing the stress contours of a given load.

Over the last 50 years FEA has been improved but the main methodology remains intact. Researchers are now attempting to enhance, or replace, FEA and CFD with machine learning [17]. In general, machine learning, and particularly deep learning, has demonstrated powerful results when applied to specific applications, but have difficulty being transferred to other fields. For example, the stresses of an artery stent design were predicted using a fully connected deep learning linear layer. However, this is only effective if the number and arrangement of nodes remains consistent. As of now, the most established application of deep learning in numerical analysis is with shape optimization of aerodynamic structures [14, 2, 20, 6].

To improve the flexibility of deep learning applied to simulation, graph neural networks (GNNs) may be a solution. Like an FEA mesh, graphs are complex, unstructured networks that can contain large number of nodes, vary-

ing node degrees, and varying graph architectures. These attributes have made graph deep learning a very active and new field of research, especially given the implications of social analysis, bioinformatics, and computer vision [23, 3, 5, 13].

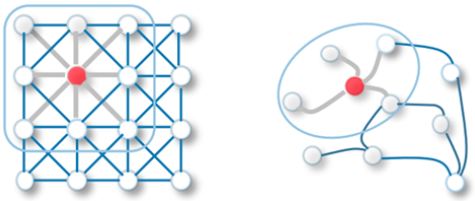

Figure 2. Representation of a Convolutional Neural Network (left) and a Graph Convolutional Network (right). Both methods involve weight sharing via a fixed size kernel which is strided over the image or graph.

Graph convolution networks (GCNs) can be split up into two different approaches: spectral-based models and spatial-based models. Spectral-based methods are constructed with frequency-based filtering that perform an eigenvalue decomposition on the graph Laplacian matrix [4, 16, 8]. While this shows promise for specific application of fixed graph networks, there are known limitations when transferring the learned Eigen parameters to different graph architectures. Spatial-based methods are more closely related to CNN models for grid-like data (images) in that they rely on extracting locally connected regions from graphs (as seen in Figure 2). Many architectures have been derived off this base principle and has shown great success in teasing out complex data from irregular, varying sized graphs [19, 10, 7, 11].

This paper investigates the application of graph convolution networks for mechanical simulation traditionally performed by FEA. Firstly, the training and testing data was created by the team using the simulation software, ANSYS [1]. A 2 dimensional, static structural model was created with constant material properties and loading conditions, but varying geometry in the mid-section. Training was performed on the first 6 models and validation was performed on the last model. Four different graph convolution network layers available in Torch Geometric were explored, comparing the models by the test case's mean squared error (MSE) and images of the stress distribution. All of the models predicted the similar stress distributions to the ground truth FEA simulation with the "ChebConv" architecture demonstrating the best results.

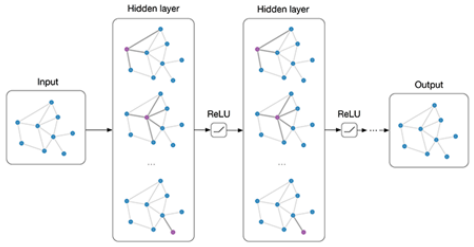

Figure 3. Visualization of an example GCN model.

2. Approach

2.1. Data Creation

To test the performance of different GCN architectures applied to FEA, a data set was created using the simulation software, ANSYS. There are some simulation data sets available publicly, but most of them are not in a format that can be easily transformed into a graph format. By building and running our own models the input and output can be controlled and eventually transformed into a PyTorch friendly data set.

Seven different 2-dimensional static, structural models were created. Seven data points is traditionally a very small data set for deep learning but for this application we are investigating a determinate system. In other words, randomness is not involved in numerical analysis and as long as the models do not differ greatly between data points, seven models is sufficient. The model contains two flat edges with rounded edges that make up a bulged bar shape. The radius of the rounded edges was altered to create different data points. The left and right edges contain the same number of nodes across all data points. The left edge of the structure is fully constrained with the right edge displaced in the "x" direction by a fixed value of 1mm. The material used in the model was "Structural Steel," which is available in the default ANSYS material library.

The FEA model has a "mesh" associated with the structural model which is a series of nodes and elements. A mesh example is visualized in the Figure 4. This mesh is essentially a graph and is used to create node adjacency vector. For this study there are only two continuous input features, "x" node coordinate and "y" node coordinate, and one continuous output variable, "Von-mises Stress" for each node in the mesh. There could be many more input and output features for this data set but we opted to keep our data set simple as a proof of concept.

One challenge that we encountered was regarding how to feed training examples to our models. Initially, we attempted to created one large disconnected graph containing all data points and masking off different portions during training. Later it was decided that it would be better to keep

the data points in separate graphs and train with them individually at each iteration with the hopes that the models would learn how to more completely predict the distribution of stresses for a given data point. This seemed to yield slightly better results regarding how stress was distributed.

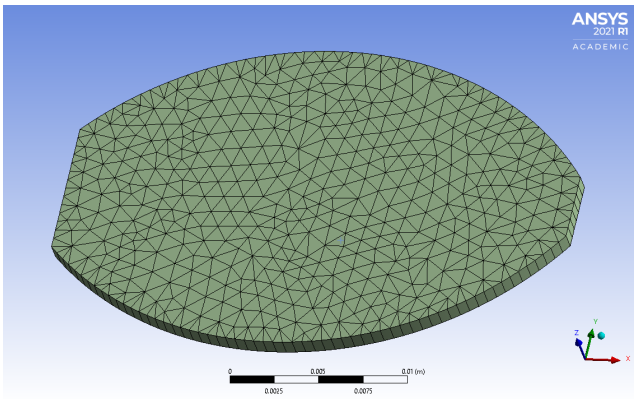

Figure 4. 2D mesh representation of a data point generated by ANSYS. Each intersection represents a node.

2.2. Training and Testing Process

The objective of this project is to predict the stress value at every node in the graph based on the graph network and individual node positions and their proximity to neighboring nodes. PyTorch was the library used to perform the optimization and back propagation, with PyTorch Geometric library providing specialized graph convolution modules. Each graph convolution architecture was trained using 6 of the data points in 6-fold cross validation and tested on the final, 7th data point. The quantitative metric used for the model loss and results evaluation was the MSE of the node stress across the entire graph. This loss function and metric was chosen as it heavily penalizes outliers and is a commonly used, general purpose loss function used in regression problem domains.

The network architecture used for each model was the same and consisted of a convolution layer, ReLu activation layer, Dropout layer, and convolution layer. This is a common architecture used for GCN's and we kept it static to allow easier comparison between models. The Adam optimizer provided by PyTorch was used as it is easy to tune and incorporates some of the best features of Adagrad and RMSProp. For each model, the hyperparameters were tuned to yield best results. The hyperparameter values selected for each model can be seen in Table 1.

2.3. Convolution Modules

GCNConv

GCNConv is a graph-based semi-supervised learning algorithm outlined in Kipf and Welling [16] where the learner is provided with an adjacency matrix, A, and node features, X, as input and a subset of node labels, Y, for training. GCN is spectral based, where eigen-decomposition of the graph Laplacian is used in network propagation. This spectral method is used to aggregate neighboring nodes in a graph to infer the value of the current node. In GCNConv, eigen-decomposition is performed via approximation to reduce runtime.

Model	Learning Rate	Weight Decay	Hidden Layer Size	Epochs	Filter Size K
GCNConv	0.01	1e-7	16	400	
MFConv	0.01	1e-7	16	300	
TAGConv	0.01	1e-7	16	300	5
ChebConv	0.005	3e-7	32	600	5

Table 1. Empirically found best hyperparameters used in experiments.

The propagation rule is represented by the following equation. GCNConv equation:

$$\mathbf{X}' = \hat{\mathbf{D}}^{-1/2}\hat{\mathbf{A}}\hat{\mathbf{D}}^{-1/2}\mathbf{X}\mathbf{\Theta}$$

Where $\mathbf{X}'$ represents the output of a particular layer, X is the input of the layer, $\hat{\mathbf{A}}$ is the adjacency matrix including self-loops, $\hat{\mathbf{D}}$ is the degree matrix of $\hat{\mathbf{A}}$, and $\mathbf{\Theta}$ is a matrix containing the learned filter parameters. As noted, the adjacency matrix is modified to include self-loops with the implication that node labels depend on the features of neighboring nodes as well as its own features.

ChebConv

ChebConv uses spectral methods for graph convolution. It uses the Chebyshev spectral graph convolutional operator showcased by Defferrard *et al*. [9]. It uses Graph Signal Processing to generalize convolution and which incorporates the entire structure and individual components of the graph.Computation works by finding the eigen-decomposition of the graph Laplacian (L in the below formulae).

$$\mathbf{X}' = \sum_{k=1}^{K} \mathbf{Z}^{(k)} \cdot \mathbf{\Theta}^{(k)}$$

where $\mathbf{Z}^{(k)}$ is computed recursively by,

$$\begin{aligned}\mathbf{Z}^{(1)} &= \mathbf{X}\\ \mathbf{Z}^{(2)} &= \hat{\mathbf{L}} \cdot \mathbf{X}\\ \mathbf{Z}^{(k)} &= 2 \cdot \hat{\mathbf{L}} \cdot \mathbf{Z}^{(k-1)} - \mathbf{Z}^{(k-2)}\end{aligned}$$

and $\hat{\mathbf{L}}$ denotes scaled and normalized Laplacian $\frac{2\mathbf{L}}{\lambda_{\max}} - \mathbf{I}$.

In summary, the graph and the kernel are transformed into spectral domain using eigen-decomposition. Convolution is performed using the spectral graph and spectral

kernel. Results are converted back to spatial domain using inverse fourier transform. The kernel used is Chebyshev polynomials of the diagonal matrix of Laplacian eigenvalues. The above formuale represents the kernel. Pooling methods used is graph coursening, which groups together similar vertices.

TAGConv

TAGConv is the topology adaptive graph convolutional networks operator demonstrated by Du *et al*. [10]. This convolution layer is similar to GCNConv, but it is expanded to multiple learning kernels for groups of nodes with different node degrees. The TAGConv equation:

$$\mathbf{X}' = \sum_{k=0}^{K} \left(\mathbf{D}^{-1/2} \mathbf{A} \mathbf{D}^{-1/2} \right)^k \mathbf{X} \mathbf{\Theta}_k$$

Where $\mathbf{A}$ denotes the adjacency matrix and $D_{ii} = \sum_{j=0} A_{ij}$ its diagonal degree matrix. The intent of this architecture is to separate learned features from nodes that have more or less neighbors. These node degree separated learned parameters allow for flexibility in regards to irregular graph structures and has demonstrated promising results in graphs with a large number of node degrees.

MFConv

MFConv is the graph network operator from the methodology depicted by Duvenaud *et al*. [11]. This convolution layer follows the same idea as TAGConv, but instead of expanding GCNConv to average across node degrees, this method directly trains a distinct weight matrix or each possible node degree. The equation:

$$\mathbf{x}_i' = \mathbf{W}_1^{(\deg(i))} \mathbf{x}_i + \mathbf{W}_2^{(\deg(i))} \sum_{j \in \mathcal{N}(i)} \mathbf{x}_j$$

Where $\mathbf{W}_1^{(\deg(i))}$ is the weight matrix applied to every node with the same node degree. Because this directly trains separate weight matrices, the number of learned parameters can be large since it scales linearly with the number of node degrees and hidden layers.

3. Results

The results of our experiments were promising. GCNConv, MFConv, TAGConv and ChebConv models were each trained with hyperparameter configurations shared in Table 1. Following training, a single test was performed on a hold out data point and results were compiled in Table 2. It is worth noting that test results show minor variations following separate training trials and different hold out data points but the same general trends were observed.

To measure success we looked at quantitative aspects such as MSE between the test predictions and ground truth and the wall-clock time. We also looked at qualitative aspects of model performance such as the distribution of stresses in the structural models. We were able to achieve low test MSE for all models, with ChebConv achieving the lowest. Generally, we can see that as model complexity increases, MSE decreases. Additionally, we generally observe higher training and test times as complexity increases.

Qualitively, we observe in Figure 6 that even with a low MSE there are still differences in how stress is distributed in the structural models. Most notably, in the ground truth we observe pronounced points of high stress in the corners that are not as extreme in any of the stress mappings produced by the models. This may be due to the distributive bias of GCN models, especially in spectral models, where there is a smoothing among node labels. The two models that best characterize these stress concentrations were MFConv and ChebConv. This may be due to the inherent separation of learning parameters of both of these methods. MFConv separates learned parameters based on the node degree and ChebConv performs an Eigen decomposition, using however many of the first values. These both allow for the model to potentially localize learned parameters to particular areas of the graph.

One item of note is that throughout our experiments we observed no evidence of overfitting. This can be confirmed by the learning curves provided in Figure 5. This is due to the deterministic nature of the stress calculations and the lack of noise. We can speculate that further training, through additional data points and epochs, can only improve results. Additionally, due to the reasons above, the models also had no issue with generalizing. In all of the test cases the models were able to produce a similar approximation of the ground truth as seen in Figure 6. Since overfitting is not possible it stands that generalization is not a concern.

The results of the experiments were successful as we have shown that Graph Convolution Networks can be competitive with traditional FEA simulations. We have seen that our experimental results can approximate the output of FEA simulation both in stress values and distribution. The results here were not perfect but due to the lack of overfitting it can be inferred that with more complex model architectures, hyperparameter tuning, and experimentation we can achieve even closer approximations. Additionally, a major benefit of using GCN's for FEA simulation is the run time. The FEA simulation run-time of 1.859 seconds is multiple orders of magnitude higher than the highest test time of our models, 0.010 seconds. This difference is likely to be even more pronounced in more complex simulations. Although we worked with simple 2D structural models with the same material and external forces among all our data points, there is no reason that the methodology used here cannot be expanded to more complex simulations.

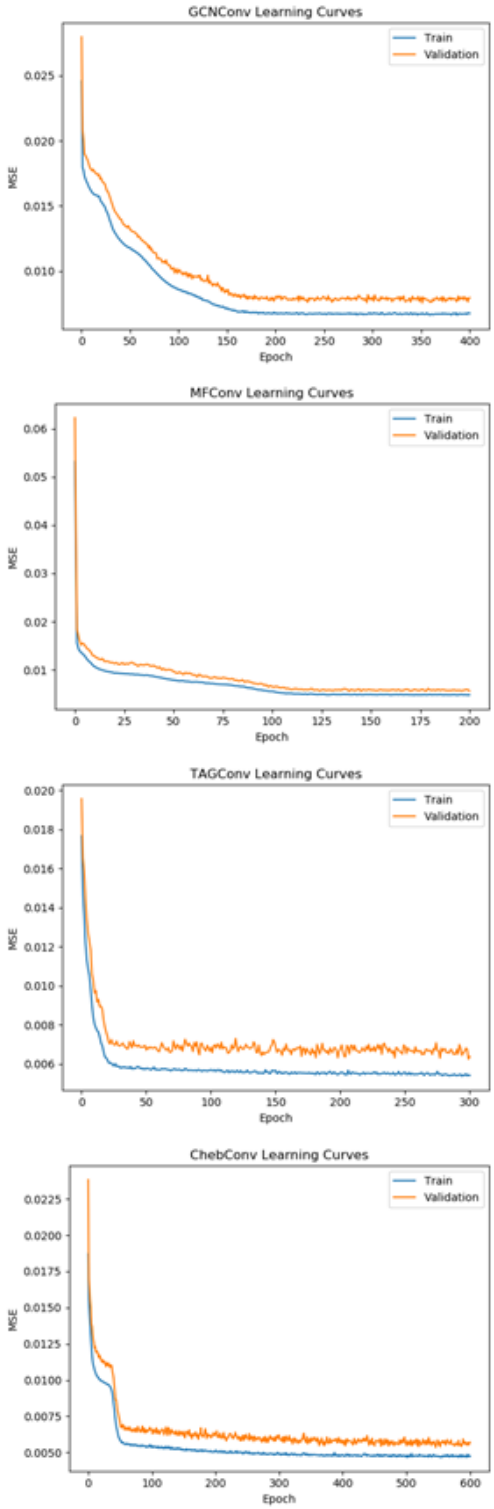

Figure 5. Training and Validation Learning Curves for each model. Even at high epochs no overfitting is observed in any model.

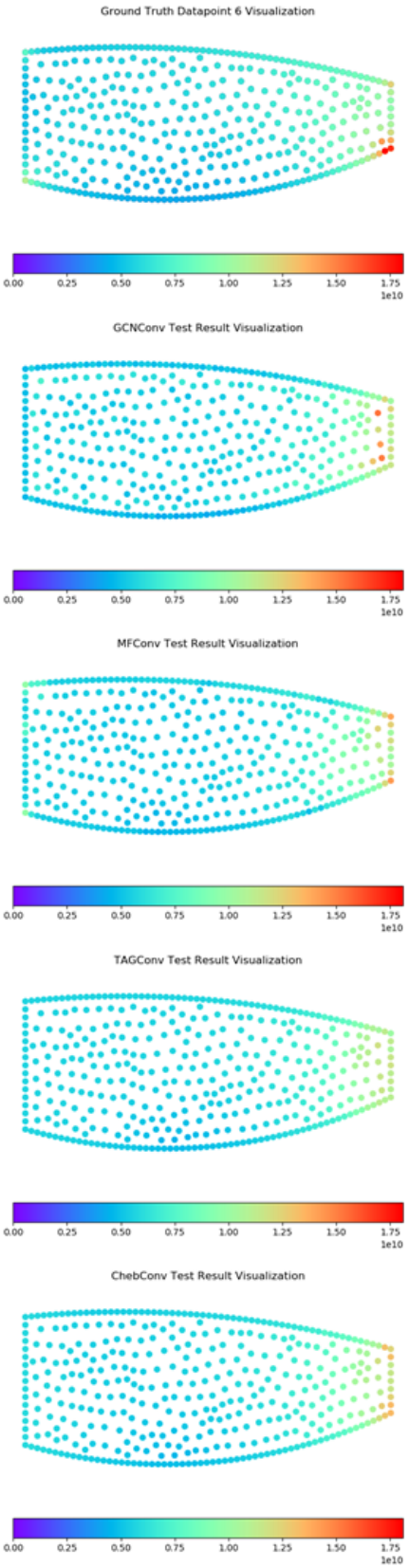

Figure 6. 2D Visualization of structural model test data point including ground truth generated by ANSYS FEA simulation. Color represents stress magnitude in Pascals. Magnitudes of stress are not as extreme as in the ground truth but distribution of stresses are similar.

Model	Params	Test MSE	Train Time (sec / 100 epochs)	Test Time (sec)
GCNConv	65	0.00338	24.282	0.002
MFConv	1243	0.00234	77.27	0.008
TAGConv	625	0.00203	76.11	0.008
ChebConv	513	0.00150	281.367	0.010
ANSYS FEA				1.859

Table 2. Results of models including number of trainable parameters, test MSE, and wall-clock times. ANSYS FEA run-time included for comparison.

4. Conclusion

Using ANSYS as a simulation software, a data set of FEA results were created to investigate the feasibility of deep learning to replace traditional numerical analysis. By running different models, GCN models were trained to predict the Von-mises stress throughout the model with only the nodal x and y coordinates as input features. In general, the experiment was a success and proved that GCNs can predict the results of an FEA model within a reasonable degree of accuracy, despite having a traditionally small data set. The spectral GCN method of ChebConv proved to be the most accurate, demonstrating not only a very low MSE, but great visual agreement with the ground truth results regarding the location of the stress concentrations. Additionally, we have seen that predictions can be made from these models in a fraction of the time of traditional FEA simulations.

There are a few avenues for future work on this topic. Firstly, more input features can be added to the graph including material properties, loading conditions, and time. Next, the breadth of models can be increased to create a more general GCN model to perform more types of analyses. Finally, testing the computational performance on a very large model to benchmark performance will be important for demonstrating the potential impact of this methodology.

5. Work Division

Work was divided between the team members as follows:

Brandon Wetzel

Created data import utility and code framework for running individual experiments. Ran experiments for GCNConv. Contributed to results section of report.

Hayden Cornwell

Created data set of FEA results. Parsed data into a clean and Python friendly form. Ran experiments for TAGConv and MFConv, along with other Conv layers in Geometric documentation. Wrote Intrdoduction section and contributed to Approach section and Conclusion. section of report.

Parth Joshi

Contributed to the Code repository. Tuned and computed results for ChebConv Graph convolution architecture. Experimented with other graph convolution architectures(eg: GraphConv) for better results. Contributed to different sections of the report.

References

[1] ANSYS, Inc. *Theory Reference for the Mechanical APDL and Mechanical Applications*, 12.0 edition, 2009. 2

[2] Pierre Baque, Edoardo Remilli, Francois Fleuret, and Pascal Fua. Geodesic convolutional shape optimization. *ICML*, 2018. 1

[3] S. Boccaletti, V. Latora, Y. Moreno, M. Chavez, and D.U. Hwang. Complex networks: Structure and dynamics. *Physics Reports*, 2006. 2

[4] Joan Bruna, Wojciech Zaremba, Arthur Szlam, and Yann LeCun. Spectral networks and locally connected networks on graphs. *arXiv Preprint*, 2014. 2

[5] H. Cai, V.W. Zheng, and K. Chang. A comprehensive survey of graph embedding: problems, techniques and applications. *TKDE*, 2018. 2

[6] Angel X. Chang, Thomas Funkhouser, Leonidas Guibas, Pat Hanrahan, Qixing Juang, Zimo Li, Silvio Savarese, Manolis Savva, Shuran Song, Hao Su, Jianxiong Xiao, Li Yi, and Fisher Yu. Shapenet: An information-rich 3d model repository. *arXiv Preprint*, 2015. 1

[7] J. Chang, J. Gu, L. Wang, G. Meng, S. Xiang, and C. Pan. Structure-aware convolutional neural networks. *Advances in neural information processing systems*, 2018. 2

[8] M. Defferrard, X. Bresson, and P. Vandergheynst. Convolutional neural networks on graphs with fast localized spectral filtering. *NIPS*, 2016. 2

[9] Michaël Defferrard, Xavier Bresson, and Pierre Vandergheynst. Convolutional neural networks on graphs with fast localized spectral filtering. *arXiv Preprint*, 2017. 3

[10] J. Du, S. Zhang, G. Wu, J.M. Moura, and S. Kar. Topology adaptive graph convolutional networks. *arXiv Preprint*, 2017. 2, 4

[11] David Duvenaud, Dougal Maclaurin, Jorge Aguilera-Iparraguirre, Rafael Gomez-Bombarelli, Timothy Hirzel, Alan Aspuru-Guzik, and Ryan Adams. Convolutional networks on graphs for learning molecular fingerprints. *arXiv Preprint*, 2015. 2, 4

[12] Ihor Farmaga, Petro Shmigelskyi, Piotr Spiewak, and Lukasz Ciupinski. Evaluation of computational complexity of finite element analysis. *CAD Systems in Microelectronics*, 2011. 1

[13] R. Girshick, J. Donahue, T. Darrell, and J. Malik. Rich feature hierarchies for accurate object detection and semantic segmentation. *IEEE conference on computer vision and pattern recognition*, 2014. 2

[14] Xiaoxiao Guo, Wei Li, and Francesco Iorio. Convolutional neural networksfor steady flow approximation. *International Conference on Knowledge Discovery and Data Mining*, 2016. 1

[15] F.H. Harlow. A machine calculation method for hydrodynamic problems. *LAMS*, 1956. 1

[16] Thomas Kipf and Max Welling. Semi-supervised classification with graph convolutional networks. *ICLR*, 2017. 2, 3

[17] Liang Liang, Minliang Liu, Caitlin Martin, and Wei Sun. A deep learning approach to estimate stress distribution: a fast and accurate surrogate of finite-element analysis. *Interface*, 2018. 1

[18] Richard MacNeal. *The NASTRAN Theoretical Manual*, 1972. 1

[19] M. Niepert, M. Ahmed, and K. Kutzkov. Learning convolutional neural networks for graphs. *International Conference on Machine Learning*, 2016. 2

[20] Edoardo Remilli, Artem Lukoianov, Stephan R. Richter, Benoit Guillard, Timur Bagauldinov, Pierre Baque, and Pascal Fua. Meshsdf: Differentiable iso-surface extraction. *ICML*, 2020. 1

[21] M.J. Turner, R.W. Clough, H.C. Martin, and L.J. Topp. Stiffness and deflection analysis of complex structures. *Journal of the Aeronautical Sciences*, 1956. 1

[22] John von Neumann and Herman Goldstine. Numerical inverting of matrices of high order. *Bulletin of the AMS*, 1947. 1

[23] Si Zhang, Hanghang Tong, Jiejun Xu, and Ross Maciejewski. Graph convolutional networks: a comprehensive review. *Computational Social Networks*, 2019. 2

[24] O.C. Zienkiewicz. *The Finite Element Method*. 1967. 1