Thornton Tomasetti

Aortas, Aneurysms, & AI

With the advanced computational tools and skillset at Thornton Tomasetti, the life sciences team has developed an exemplar workflow for in silico trials in the medical device industry, studying biomechanical properties of life-saving therapies. We used simulations and machine learning (ML) to acquire and analyze data representing aortic device implantation.

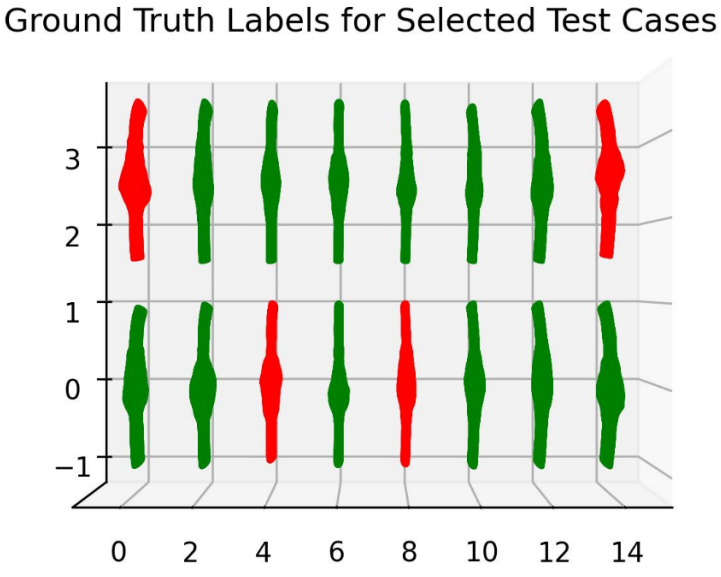
In a previous whitepaper, we described how our virtual cohort was acquired with anatomy parametrization and finite element analysis (FEA), and how we assessed traditional classifiers like decision tree (DT) and support vector machine (SVM) trained on a handful of measurable parameters from the FEA results. The purpose of this project was to model surgical outcomes for assisting device design. We explored model interpretability and model performance. Depending on the application, the appropriate model will address the needs of the particular medical solution (e.g., a high cost for incorrect predictions might call for more interpretable models) [1].

Approach	Model Complexity	Model Interpretability	Model Performance
Traditional ML models (DT, Linear regression, SVM, etc.)	Low	With simple relationships between inputs and outputs, understanding the decision can be accomplished via intrinsic analysis (e.g. feature weight/importance) or post-hoc analysis	With lower complexity, these models tend to have lower accuracy because they can not capture the complex, non-linear relationships
Deep learning models (NN)	High	Neural networks can lend themselves to post hoc analysis (e.g. some learned feature maps may be interpreted depending on the solution)	These models tend to have higher accuracy by capturing non-linear relationships

Our ML journey began with simple models that were reasonably interpretable and provided sufficient performance. With such a valuable dataset, we want to similarly test performance of more complex AI/ML models using the entire geometry data available to us. In this post, we showcase our capabilities in AI using neural networks (NN) on the raw spatial geometry dataset in the pursuit of high-performance modeling.

A Simple Task for Graph Neural Network (GNN) Demonstration

We chose a simple task to perform on our graph data: graph-level classification. This means each of our patients (represented as a graph with nodes and edges with geometrical features) is labeled as a success or failure, using exactly the same binary labels as before. Specifically, a failure is considered when the proximal or distal landing zone length is less than 10 mm. It is important to note that expert labeling could influence the model's learning and performance. Presumably, our labeling identifies endoleak failure where there is not enough coverage to form a proper seal for the stent graft, but a more realistic labeling might reflect more complex relationships among the data (e.g., a cardiologist might decide a 9 mm landing zone length is acceptable based on the shape of the aneurysm in one case due to a constricting geometry, but the same cardiologist might decide an 11 mm landing zone length is insufficient in a patient due to a growing aneurysm), which would theoretically call for more complex models. Therefore, testing models of various complexity is good practice.



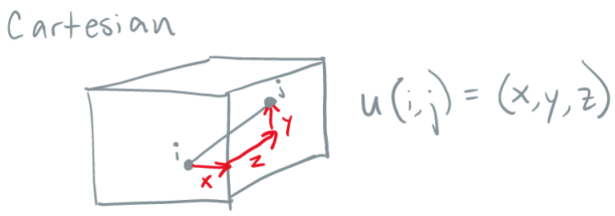
Ground truth labels of selected test cases. In these visualizations, both the aorta and the graft are colored corresponding to the label, so the 2 materials are not distinct here; the focus here is to model the distinction between success (green) and failure (red) where geometrical information is used from both materials. These ground truths correspond to predicted labels in the following results section.

Given runtime constraints, we chose to only process the last timepoint of the aorta and graft materials, excluding the nitinol stent—the 3D rendering of the stent geometry has too many nodes (more than 100X that of the graft and aorta) to feasibly train on the GPU, and including more time points from the implantation simulation would also increase the complexity of our model.

Meet the Models

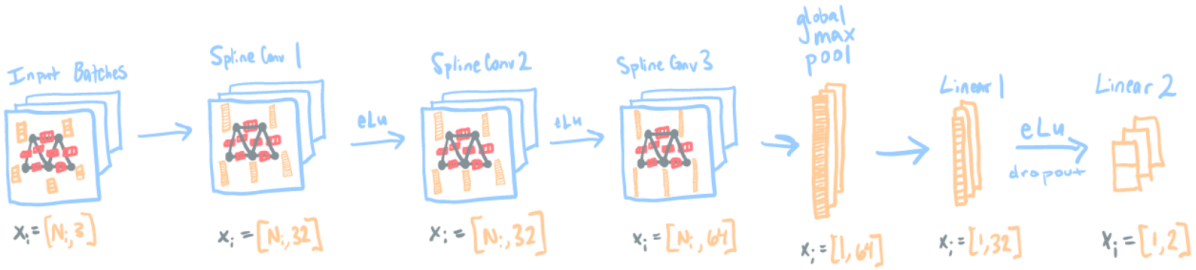
Our dataset consists of 76 samples, each with ~25,000 nodes with spatial features. The high dimensionality of this dataset points to deep learning, and recent developments in open-source tools support the graphical nature of the 3D mesh data. Particularly, we found a method that reports to perform incredibly well on graphs representing 3D geometry: SplineCNN [2]. The math behind this special SplineConv operation is very well explained in the original work and is implemented in PyTorch geometric for user-friendly integration into GNN pipelines [3].

Appropriate processing and model architecture design depends on the task at hand. Our data is structured as triangular meshes in .stl format. After loading each sample as a torch_geometric data object, we process these with a Cartesian transformation, which saves normalized edge attributes necessary for the SplineConv operation. Other edge weight mappings include 2D Cartesian, polar, and spherical coordinates. The figure below shows a toy example demonstrating how to generate the edge attributes matrix with 3D Cartesian coordinates.



Cartesian transform for SplineConv. This transform is implemented in pytorch geometric. Each edge is weighted by a vector which is mapped to a scalar by the trainable, continuous B-spline kernel function.

We chose a simple architecture with three SplineConv layers and two linear layers, similar to the scene-level classification task demonstrated with SplineCNN. Using two SplineConv layers performed worse compared to three layers, and modeling more channels in the hidden layers increased runtime with no gain in performance. Our pooling scheme is different than SplineCNN, but seems to work for our case. Global max pooling performed better than mean or add.



meshGNN model architecture. Our model consists of three SplineConv layers, a pooling layer, and two Linear layers. We chose a kernel size of 5 for the convolutions. We also added a dropout layer in between the linear layers to address overfitting. The activation function for each of the convolution and the first linear layer was Exponential Linear Unit (eLu). We chose log_softmax as the final activation function to return prediction probabilities for each class, but a sigmoid function also works in the binary classification task. The loss function was Negative Log Likelihood, and the optimizer was Adam with learning rate 0.001.

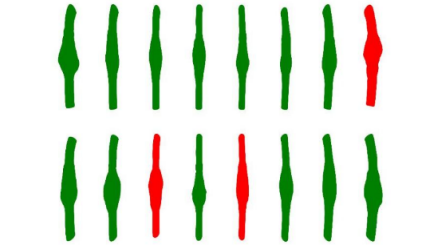
For training and testing models, we split the data with 53 training samples and 23 testing samples, where the split was stratified with seven failures in the training set and four failures in the testing set.

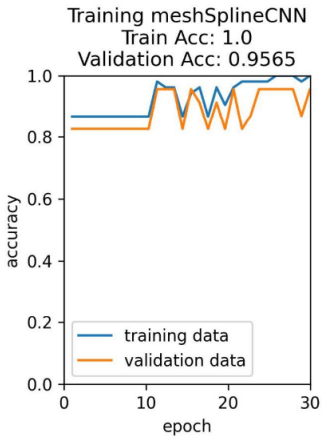
The Verdict

Ultimately, we began this endeavor to compare traditional ML models and deep learning models on virtually the same classification task. In terms of speed and performance, there are pros and cons for each.

The speed of the traditional ML models is famously fast, where training with low dimensional input is complete in seconds. This method requires serious dimensionality reduction and is not feasible to train these models on the raw 3D geometry. In contrast, our meshGNN model trains for 30 epochs in about an hour on a NVIDIA GeForce MX150. The runtime depends greatly on the size of the input, the complexity of the model architecture, and the hardware.

In this demonstration, we saw similar performance across classifiers. Both traditional ML and deep learning approaches are able to achieve greater than 90% accuracy on this classification task using the same data in different forms. One consideration is the different train-test splits: for the DT and other traditional classifiers, we included eight fail cases in the train set, but only seven fail cases for the meshGNN classifier train set. With less failure examples in the training set, the deep learning model was still able to distinguish these cases. For now, we will say that these approaches are tied and each reveals valuable knowledge from the data in terms of interpretability and performance.





Training Results. The predicted labels for a selection of cases in the hold-out dataset at the 30th epoch are shown on the left. On the right is shown the plot of the performances of both sets across 30 epochs of training.

More to Explore

With a small dataset of only 76 virtual patients and an awkward class imbalance of only 11 failures, we saw limited performance similar to our previous methods with traditional ML models. Future directions with this dataset could potentially address this issue, such as data augmentation—a common image processing step where the dataset size is increased by rotating or flipping original geometries. Alternatively, this dataset could be used to build generative models that can synthesize more failure cases. Another exciting direction would be to model the time-series data of the whole simulation. We could also see extensions of this project exploring interpretability methods, which is important for model robustness. Albeit small, this dataset is a valuable resource for any data scientist, and we look forward to applying these solutions for clients!

References

[1] Amazon Web Services, Inc., "Machine Learning Best Practices in Healthcare and Life Sciences AWS Whitepaper," Amazon Web Services, Inc., 2021.

[2] M. Fey, J. E. Lenssen, F. Weichert and H. Muller, "SplineCNN: Fast Geometric Deep Learning with Continuous B-Spline Kernels," Proceedings of the IEEE conference on computer vision and pattern recognition, pp. 869-877, 2018.

[3] M. F. (rusty1s), "Spline-Based Convolution Operator of SplineCNN," GitHub, [Online]. Available: https://github.com/rusty1s/pytorch_spline_conv.