A Hybrid Surrogate Model for Computational Fluid Dynamics for Aortic Flow using Geometric Deep Modeling and Differentiable Solvers

Pan Du
Department of Aerospace and Mechanical Engineering
University of Notre Dame
South Bend, IN
pdu@nd.edu

Deepak Akhare
Department of Aerospace and Mechanical Engineering
University of Notre Dame
South Bend, IN
dakhare@nd.edu

Abstract

With the rapid development in AI and GPU computing, there has been a growing interest in developing deep learning (DL)-based surrogate models due to their higher efficiency and scalability. However, solving large complex partial differential equations (PDEs), such as those that arise in computational fluid dynamics (CFD), is a computationally expensive process. Moreover, conventional DL approaches (e.g., CNN) suffer from significant approximation errors due to non-intrinsic geometrical representation resulting in simulations that do not generalize well with truly novel scenarios. To address this issue, a hybrid surrogate model that combines a traditional graph neural network (GNN) with a differentiable fluid solver is created to predict cardiovascular flow hemodynamics. The report will provide a detailed discussion of the model implementation and will be demonstrated for 2D Aortic flow, where fluid field will be predicted given a random inlet velocity profile.

1 Introduction

The past few years' success in deep learning has motivated scientists to explore its application in other areas, such as predicting the evolution of physical systems. Recently, several research papers have been published where deep learning models are used to approximate the solutions to partial differential equations (PDEs), particularly for simulating fluid dynamics(1; 2; 3; 4). The behavior of fluids is a well-established field; obtaining its dynamics requires solving Navier-Stokes partial differential equations (PDEs), which are solved numerically to get computational fluid dynamics (CFD) simulations. However, these CFD models are computationally expensive; some of them take many days to weeks on massive supercomputing infrastructure. This makes it infeasible for optimization studies, where many simulations need to be conducted on different geometry and parametric conditions. Recently deep learning models to create fast surrogate models have drawn a great deal of interest in solving the issue. However, despite the work in this area, most deep learning models cannot capture the full complexity of the underlying equations and produce poor results when testing on the out-of-training domain.

36th Conference on Neural Information Processing Systems (NeurIPS 2022).

In this article, a hybrid approach developed(5) that combines the benefits of (graph) neural networks for fast predictions with the physical realism of an industry-grade CFD simulator is tested on flow over cylinder problem. The model consists of two main components. First, a graph convolution network(6) (GCN) is constructed that operates directly upon the non-uniform mesh used for typical CFD simulations. The use of GCNs is an essential part of the model as it enables the model to operate on unstructured mesh and not limit it to the regular grid used by most prior work, where a convolutional neural network is used to approximate CFD simulation. Second, a differentiable CFD solver operates on a much coarser resolution and is directly embedded into the GCN. Recently, differentiable solvers are emerging by virtue of their capability of computing gradients beside flow outputs. In our case, we use an open-source solver SU2(7) that can calculate the gradient of the flow field with respect to the input mesh coordinates on a coarse mesh, which is embedded into the hybrid surrogate model. During the training process, both the GCN and the course mesh coordinate are optimized simultaneously when the desired fitting result is reached. In principle, the hybrid model performs substantially better than the coarse CFD simulation alone and generalizes well to the out-of-training domain compared to a pure graph network-based approach. In addition, the approach is substantially faster than running the CFD simulation on the original size mesh itself. With the rapid development in AI and GPU computing, there has been a growing interest in developing deep learning (DL)-based surrogate models due to their higher efficiency and scalability. However, solving large complex partial differential equations (PDEs), such as those that arise in computational fluid dynamics (CFD), is a computationally expensive process. Moreover, conventional DL approaches (e.g., CNN) suffer from significant approximation errors due to non-intrinsic geometrical representation resulting in simulations that do not generalize well with truly novel scenarios. To address this issue, a hybrid surrogate model that combines a traditional graph neural network (GNN) with a differentiable fluid solver is created to predict cardiovascular flow hemodynamics. The report will provide a detailed discussion of the model implementation and will be demonstrated for 2D Aortic flow, where the fluid field will be predicted given a random inlet velocity profile.

2 Methodology

The general outline of the hybrid surrogate model is described in this section. Initially, broad CFD-GCN architecture is described, and after that, each component is described in detail.

2.1 CFD-GCN architecture

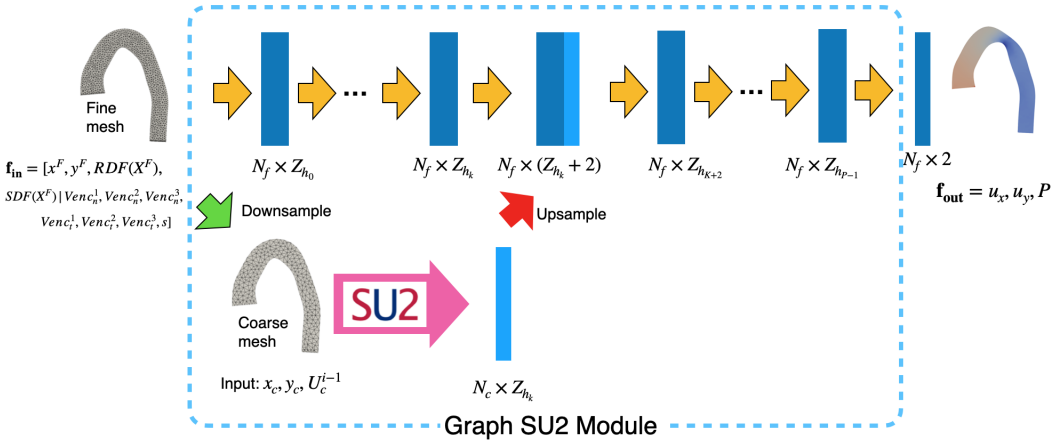

Figure 1: A diagram of the CFD-GCN model and its corresponding equations.

Figure. 1 shows the overall architecture of the CFD-GCN model. The model is designed to predict fluid quantities (e.g., pressure, velocity) given a initial mesh and boundary conditions (i.e., the velocity profile at the inlet) as input nodal features. The input information is propagated in two scales: on the fine scale, the inputs are passed through multiple graph convolutional neural network (GCN) layers

and connect to the field output. On the other hand, the fine mesh is down-sampled to a coarse mesh on which SU2 simulation is performed to produce coarse solution, which is then upsampled back to the fine scale and concatenated with a hidden GCN layer. Note that the trainable parameters include parameters from GCN layers and also the coarse mesh coordinates in the SU2 solver.

Graph input feature design. Initially, a 2-D triangular mesh $M^F = (X^F, T^F)$ is generated given the geometry of a aorta, where F denotes the fine scale and $X^F \in R^{N\times 2}$ denotes the coordinates of N vertices of the 2-D mesh. $T^F \in R^{P\times 3} = \{(i_1, j_1, k_1), ..., (i_M, j_M, k_M)\}$ represents a total of P triangular faces where (i_q, j_q, k_q) is a set of vetex indices of the points of the triangle. The mesh is converted to a graph $G^F = (X^F, E^F)$ whose nodes and connectivity remains the same, where $E \in R^{Q\times 2} = \{(i_1, j_1), ..., (i_M, j_M)\}$ denotes the edges. Conversely, a graph can be converted back to a mesh. Both geometric and BC parameters are included in the input nodal features of the hybrid model. Equation 1 shows the components of those features.

$$\mathbf{f_i n} = [x^F, y^F, RDF(X^F), SDF(X^F)|Venc_n^1, Venc_n^2, Venc_n^3, Venc_t^1, Venc_t^2, Venc_t^3, s] \quad (1)$$

The geometric descriptors include mesh coordinates "x^F, y^F" and radial "$RDF(X^F)$" and streamwise distance function "$SDF(X^F)$". The former measures the distance from a node to the centerline and the latter measures how far a node is from the inlet. In terms of the inlet velocity profile, we use a 1-D Gaussian process 2 to parameterize the profile.

$$\begin{aligned} f(x) &\sim \mathcal{GP}\left(0, k\left(x, x'\right)\right) \\ k\left(x, x'\right) &= \sigma_0^2 \exp\left(-\frac{\left|x - x'\right|^2}{2l^2}\right) \\ x &\in [0, 1] \end{aligned} \quad (2)$$

where 3 parameters are used to generate a random velocity profile in the inlet normal direction "$[Venc_n^1, Venc_n^2, Venc_n^3]$" and another 3 parameters in the tangential direction "$[Venc_t^1, Venc_t^2, Venc_t^3]$" and a scalar "$s$" that scales the velocity magnitude. The output of the solver and the last GCN layer is velocity and pressure on the fine mesh.

The SU2 Fluid Simulator. The essential component of the CFD-CGN model is the integrated differentiable fluid dynamics simulator. The initial input fine mesh M^F is downsampled to a coarse mesh M^C, whose vertices coordinates are passed into the SU2 solver. The SU2 performs the CFD simulation on the course mesh and outputs predictions of the velocity and pressure at each node in the coarse graph. Specifically, the SU2 solver uses the finite volume method (FVM) to solve the incompressible Navier-Stokes equations in a forward run. When adjoint information is needed, it uses reverse-mode differentiation to obtain the gradients in a back-propagation manner. The SU2 solver utilizes CODIPack, a fast gradient evaluation tool written in C++, to record the computational graph and can compute gradients using either forward or back propagations. In the implementation, we first modified the C++ base code to enable adjoint extraction for pressure and velocity with respect to the input coordinates and then created a python wrapper function for the SU2 solver. Then we were able to add the python wrapper into a customized PyTorch autograd function, which can return gradients in tensor forms. The customized function can be directly added into a neural network module(NN) and seamlessly works together with any other regular NN layers. In addition, the SU2 python wrapper also features parallel computing using the MPI4PY package(8). Therefore, samples in a mini-batch are computed in parallel for the simulation part.

Upsampling. The fluid output, including velocities and pressure, obtained from the SU2 solver is on a coarse mesh. To append the output information into an intermediate GCN layer, it is upsampled to the fine mesh via successive applications of squared distance-weighted, k-nearest neighbors interpolation(9). Let $U^F \in R^{N_F\times 3}$ be the upsampled version of coarser mesh $U^C \in R^{N_C\times 3}$. For vertex X_i^F on the fine mesh, we find k closest nodes $\{X_{i_1}^C, ..., X_{i_k}^C\}$ on the coarse mesh obtain the value U_i^F via interpolating in the neighbourhood values $\{U_{i_1}^C, ..., U_{i_k}^C\}$:

$$U_{(i)}^F = \frac{\sum_{j=1}^k w\left(i_j\right) U_{i_j)}}{\sum_{j=1}^k w\left(i_j\right)} \quad (3)$$

where

$$w\left(i_j\right) = \frac{1}{\|X_i^F - X_{n_j}^F\|_2^2} \tag{4}$$

Where the number of neighboring nodes is set to $k = 3$.

GCN. As shown in Fig. 1, several GCN layers are used, before and after the concatenation of the solver output, to map the input nodal features to the flow outputs. The GCN layer follows the design in (6). This architecture defines a convolutional layer for graphs.

Given a graph $G = (V, E)$ with of N nodes and input features matrix $U \in \mathbb{R}^{N\times p}$. Let $A \in \mathbb{R}^{N\times N}$ be the adjacency matrix, We can calculate $\tilde{B} = \tilde{D}^{-\frac{1}{2}}(A + I)\tilde{D}^{-\frac{1}{2}}$, where I is the identity matrix and $\tilde{D}$ the diagonal degree matrix, with its diagonal given by $\tilde{D}_{ii} = 1 + \sum_{j=0}^{N_Z} A_{ij}$. Then, a GCN layer maps the input features U to the output features U' using a convolution operation defined by the weight matrix $W \in \mathbb{R}^{p\times p'}$ and the bias term $b \in \mathbb{R}^{N\times p'}$, and a non-linearity σ (i.e., ReLU).

$$U' = \sigma(\mathrm{GCN}\left(U\right)) = \mathrm{ReLU}\left(\tilde{B}UW + b\right) \tag{5}$$

CFD-GCN. Now that all the components of the CFD-GCN have been described, we can now bring them all together to describe the full pipeline shown in Fig. 1.

Initially, a SU2 simulation is run on the coarse mesh for given physical parameters, and the output of this coarse simulation is upsampled.

$$U_L = \mathrm{Upsample}\left(\mathrm{SU2}\left(x_C, y_C, U_{inlet}\right)\right) \tag{6}$$

$$\begin{aligned} U_0 &= [\mathbf{f}] \\ U_{i+1} &= \mathrm{ReLU}\left(\mathrm{GCN}_i\left(U_i\right)\right), i = 0, \ldots, k-2 \\ U_k &= \left[\mathrm{ReLU}\left(\mathrm{GCN}_k\left(U_{k-1}\right)\right), U_L\right] \\ U_{k+i+1} &= \mathrm{ReLU}\left(\mathrm{GCN}_{k+i}\left(U_{k+i}\right)\right), \quad i = 0, \ldots, K-k \\ \hat{Y} &= \mathrm{GCN}_K\left(U_K\right) \end{aligned} \tag{7}$$

Here, $[\cdot, \cdot]$ is the matrix concatenation operation over the column dimension. We put 3 GCN layers before and after the concatenation of the solver output. The dimension of the input feature is $n_0 = 11$, hidden features are $n_{i=1\sim5} = [512, 512, 515, 512, 512]$, and output feature is $n_6 = 3$.

2.2 Training CFD-GCN

The entire CFD-GCN, as formulated above, can be treated as a single differentiable deep network. The model is trained to predict the output fields $Y \in R^{N\times3}$, consisting of the x and y components of the velocity and the pressure at each node in the fine mesh, by minimizing the mean squared error (MSE) loss between the prediction $\hat{Y}$ and ground truth

$$l(Y, \hat{Y}) = \frac{1}{3N}||Y - \hat{Y}||_2^2, \tag{8}$$

where the ground truth Y is obtained by running the full SU2 solver on the original fine mesh. The training procedure optimizes the parameters W_i and b_i of the GCNs, and the positions of the nodes in the coarse mesh X_C by backpropagating through the CFD simulation. The flow outputs are normalized within the range [-1,1] before training. The loss is minimized using the Adam optimizer(10) with a learning rate $\alpha = 1 \times 10^3$. With 1800 training samples and 200 testing samples, the training is executed on a Nvidia A6000 Graph Card for 3000 epochs with a batch size of 4, which takes about 2 hours.

3 Results and discussion

Random velocity inlet. Using the Gaussian process, we randomly generate the velocity profile shown in Figure 2, where the normal and tangent velocities are illustrated over 3 random samples. Note

that x^* is the normalized distance from a point on the inlet to the upper end of the inlet. The normal velocity is $U_{inlet\_n} = u_0 + \lambda_n \tilde{u}_n$, where $u_0 = 1$ is a base value and $\lambda_n = 0.5$ is the scaling coefficient of the fluctuation terms generated by the 1-D Gaussian process. The tangent velocity $U_{inlet\_t} = \lambda_t \tilde{u}_t$ use a smaller scaling coefficient $\lambda_t = 0.2$. Finally, the scaling factor $s \sim \mathbf{Uniform}[0.2, 1]$ scales the velocity vector which yields the random velocity profiles as:

$$\mathbf{U}_{inlet} = s\left((u_0 + \lambda_n \tilde{u}_n)\mathbf{e}^n + \lambda_t \tilde{u}_t \mathbf{e}^t\right) \tag{9}$$

A total of 2000 random inlet profiles are generated and the corresponding flow solutions are generated on the fine mesh using the python interface of the SU2 solver. Afterward, the simulation data is converted into graphs containing the input and output features. Thus the training dataset is established.

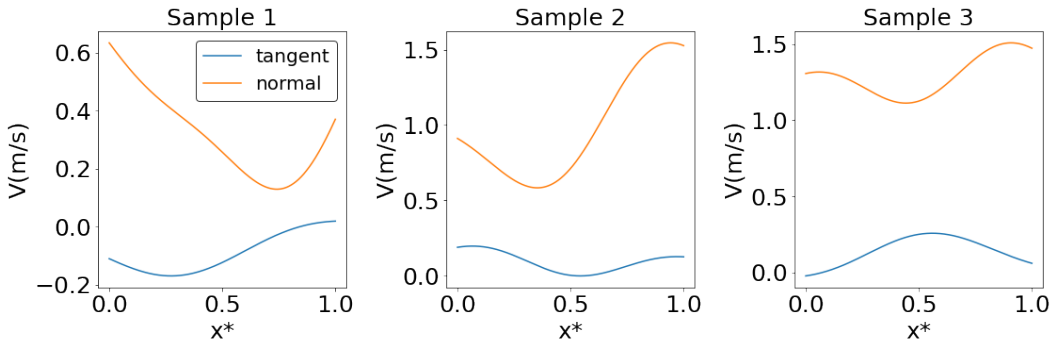

Figure 2: Random samples of velocity profiles using Gaussian process

Prediction results. Figure 3 shows the training error as a function of epochs. The error drops about 2 orders of magnitudes (there is a jump from 10^0 to 10^4 in the beginning), indicating a good fitting result. Unfortunately, we didn't notice evident mesh vertices location changes during the training process due to small gradients returned on the mesh coordinates. This is probably caused by inappropriate coarse mesh resolution or parameter settings of SU2, which will be fine-tuned in future studies. Figures 5 and 4 shows the prediction results for velocity and pressure for 4 different testing

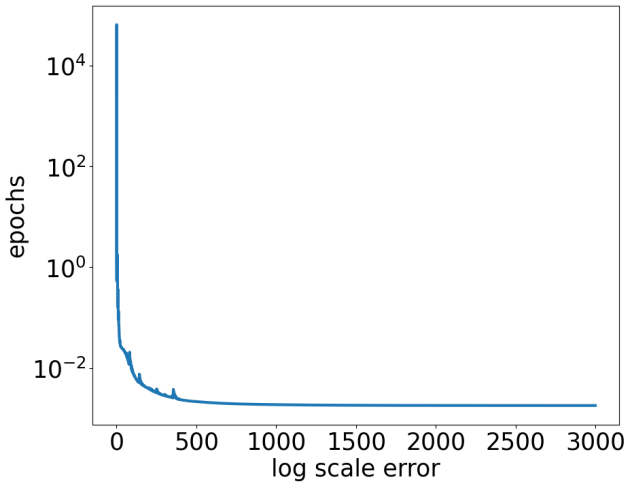

Figure 3: training error

samples, with the first, second, and third row representing the label, prediction, and the absolute difference between them, respectively. The results show that the predictions from the hybrid model are in good agreement with the ground truth, especially for the velocity. The hybrid model can accurately capture the flow patterns near the inlet caused by the inlet profiles. The pressure prediction is less ideal. For now, the relative MSE error is 27.4% evaluated on the test dataset. The performance of the proposed model will be improved by better normalization of the output or further fine-tuning of the hybrid model. In general, the hybrid model has a good performance in predicting flow fields.

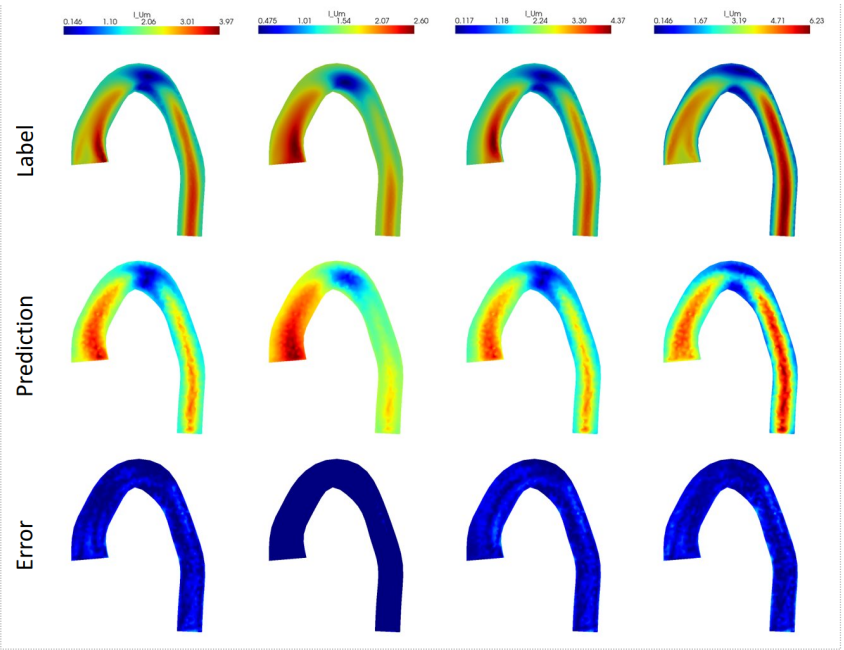

Figure 4: Comparison between CFD-GCN velocity prediction and label for test cases with different inlet velocity profiles

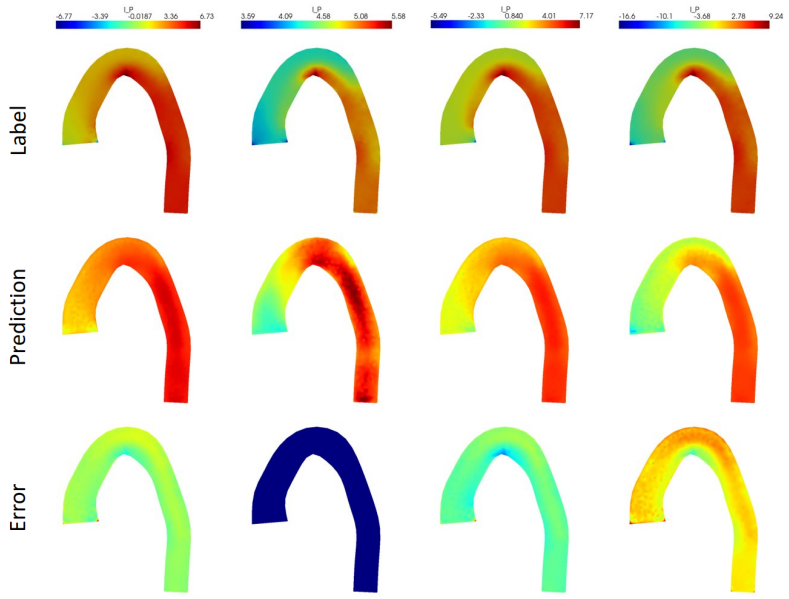

Figure 5: Comparison between CFD-GCN pressure prediction and label for test cases with different inlet velocity profiles

4 Conclusion

In this project, a hybrid model that integrates a differential CFD solver and GNN has been presented. The graph operator of the CFD-GCN system allows us to operate on unstructured meshes. The experiment conducted in this work demonstrated that the combination of the fluid simulations performed on a coarser version of the original mesh and the learned parts of the deep learning model

creates a system that generates predictions that are faster than a full-order simulation while still accurately predicting flow fields. In future work, the model will be fine-tuned and compared with a pure neural network model to demonstrate its superiority in terms of generalizability. In addition, we are expecting to extend the application of this hybrid model such that it can predict a dynamic process given an initial condition. For example, we planned to add the velocity field into the original input nodal feature and let the model predict the flow field at the next time step. This required significant code-intrusive modifications which we didn't finish during the course time span. However, this endeavor will continue in future works.

The code of this project is the github:https://github.com/shanjierenyidp/CFD_GCN.git

References

[1] S. Bhatnagar, Y. Afshar, S. Pan, K. Duraisamy, S. Kaushik, Prediction of aerodynamic flow fields using convolutional neural networks, Computational Mechanics 64 (2) (2019) 525–545.

[2] X. Guo, W. Li, F. Iorio, Convolutional neural networks for steady flow approximation, in: Proceedings of the 22nd ACM SIGKDD international conference on knowledge discovery and data mining, 2016, pp. 481–490.

[3] S. Wiewel, M. Becher, N. Thuerey, Latent space physics: Towards learning the temporal evolution of fluid flow, in: Computer graphics forum, Vol. 38, Wiley Online Library, 2019, pp. 71–82.

[4] K. Um, X. Hu, N. Thuerey, Liquid splash modeling with neural networks, in: Computer Graphics Forum, Vol. 37, Wiley Online Library, 2018, pp. 171–182.

[5] F. D. A. Belbute-Peres, T. Economon, Z. Kolter, Combining differentiable pde solvers and graph neural networks for fluid flow prediction, in: international conference on machine learning, PMLR, 2020, pp. 2402–2411.

[6] T. N. Kipf, M. Welling, Semi-supervised classification with graph convolutional networks, arXiv preprint arXiv:1609.02907 (2016).

[7] T. D. Economon, F. Palacios, S. R. Copeland, T. W. Lukaczyk, J. J. Alonso, Su2: An open-source suite for multiphysics simulation and design, Aiaa Journal 54 (3) (2016) 828–846.

[8] L. Dalcin, Y.-L. L. Fang, mpi4py: Status update after 12 years of development, Computing in Science & Engineering 23 (4) (2021) 47–54.

[9] C. Qi, L. Yi, H. P. Su, L. P. Guibas, Deep hierarchical feature learning on point sets in a metric space. arxiv 2017, arXiv preprint arXiv:1706.02413.

[10] P. Kingma Diederik, J. B. Adam, A method for stochastic optimization, arXiv preprint arXiv:1412.6980 (2014).