ANALYSIS OF TRAINING PINNS ON MULTIPLE COMPUTATIONAL FLUID DYNAMICS MODELS CONSISTING OF DIFFERENT SHAPES

Bryce Montgomery Kevin Bachelor Manju Shettar Aled dela Cruz

UC Santa Cruz
bamontgo@ucsc.edu, kwbachel@ucsc.edu, mshettar@ucsc.edu, alrdelac@ucsc.edu

Abstract

This paper presents an analysis on the learning capabilities of Physics Informed Neural Networks, **PINNs**, in the context of solving the Naiver Stokes equations. In general, industry standard prediction models perform well on commonly-constructed structures, such as airfoils. However, they cannot generalize well enough to obtuse shapes in the field of Computational Fluid Dynamics. Our objective for this paper was to train a Physics Informed Neural Network on simulated airflow data to improve performance on non-traditional airfoils.

BACKGROUND/MOTIVATION

Physics informed neural networks are deep learning models that are able to learn through incorporating input data and physical laws into governing partial differential equations, an idea initially introduced by Raissi et al(3). PINNs have shown their usefulness in solving a multitude of real world physics problems. In our study, we incorporate the use of PINNs into the solving of the Naiver Stokes Equations for differing inputs, particularly air flows.

The original goal of this project was to create a generalised PINN model capable of predicting aerodynamic coefficients for non-uniform use cases. Previous literature focuses on specific optimization problems, such as tweaking airfoil shapes for certain drag-to-lift ratios, or predicting wake turbulence around a cylinder. In sum, our model would be able to work just as well for optimizing car designs as it does for airplanes or drones.

The envisioned solution to this problem was to find some way to obtain simulated and/or measured wind tunnel data, and use it to train the model on a diverse range of objects. Hopefully, over the course of training, the model would be able to accurately replicate the results of traditional CFD methods while cutting down on computational cost. The accuracy factor would be tested by comparing the output of the model with the real/simulated data. It is very important that the model perform well in unfamiliar scenarios (meaning it can successfully model situations it did not encounter in training). This would prove the efficacy of PINNs.

METHODOLOGY

Overview

We used a PINN detailed in the original paper trained on a simple simulation. The simulation in question was a CFD simulation performed on a cylinder to calculate the aerodynamic coefficients. This provided us with a framework with which to test different types of airflow models with easy to reproduce outputs.

In order to train this PINN further, we visualized and parsed data from open source simulations of aerodynamic space optimization. OpenFOAM is an open-source computational fluid dynamics simulator used commonly in aerodynamic research. The software provides solvers for fluid-flow problems, which assists our inquiries about the Naiver Stokes equations. Since there was no standardized technique of transferring and transforming our data between technologies, we had design our own pipeline.

Pipeline

- **OpenFOAM** - to simulate airflow around various airfoils and generate aerodynamic coefficients
- **ParaView** - to visualize our simulation results and produce a formatted .csv file containing the coefficients and data we required
- **Pandas** - to parse data stored in our .csv files

OpenFOAM Simulation

OpenFOAM's docker binary extension allows us to simulate airflow experiments through command line interfaces. We used a simple command;

$$simpleFoam\ -case\ simulationName \quad (1)$$

where simulationName is the name of the in-compressible simulation, to run the tutorial simulations found as part of the OpenFOAM library. Each simulation contains a block mesh on which the simulation is run, an object on that mesh, and a variable time limit for the airflow. At each constant time step, the aerodynamic coefficients are calculated for each point on the block mesh.

To visualize our data, we used;

$$foamToVTK\ -case\ simulationName \quad (2)$$

to generate a vtk (visualization toolkit) file. After gathering vtk files from each simulation, we were able to open them using ParaView and visualize our data.

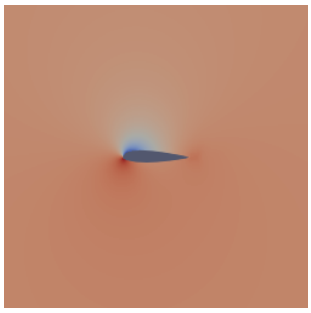

Figure 1: Pressure around a simulated airfoil visualized in ParaView

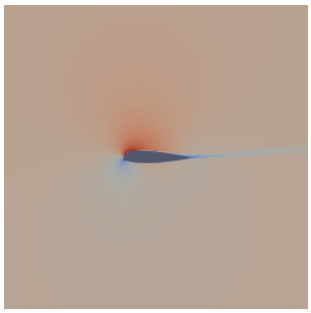

Figure 2: Velocity around a simulated airfoil

ParaView allowed us to manipulate the results of the tutorial simulations, choosing the variables that we wanted to visualize. Since the Naiver Stokes equations describe the motion of fluids, we extracted the continuous pressure and velocity values from the simulations. Other than visualizing vtk files easily, ParaView allowed us to export the coefficient values from the simulations into a formatted .csv file.

Data

The resulting .csv files were formatted with the following variables:

- time steps: While all of the simulations had a standardized time step of 50 seconds, they varied in the total time they ran for. The inconsistencies in the time is something we address in our results.
  - airfoil: 500 total seconds
  - building: 400 total seconds
  - car: 208 total seconds
  - cylinder: 200 total seconds
- spatial coordinates: (X, Y) denotes a position in the mesh.
- pressure (p)
- flow velocity (u)

Since every 50th step was reported in the .csv files, that means for every time step, our .csv files contained all of the pressure and velocity information for each spatial coordinate. Extrapolating this information from the .csv file was fairly simple. We used pandas to parse the file by columns, turn the input dataframe into tensors and plugged our tensors into our model.

Model Training and Testing

The model we used was specifically chosen over other models because of the way it took input. Many other models used specialized shape describing parameters to train(1), which would have made it very difficult to generate the random shapes for training in a way that they could take in. The model we ended up using on the other hand took in Matlab files and a relatively generic representation of pressure and airflow velocities, and location of points (given as p, U [u, v] and X [x, y])(2).

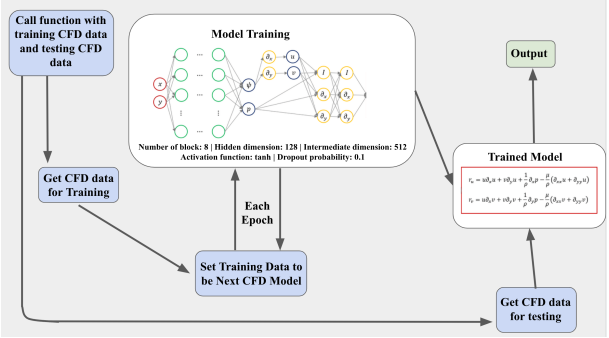

Figure 3: Velocity around a simulated airfoil

The training process involved taking in the .csv file instead of a Matlab file that contained the same values. We took 4 total airflow models, each with a different shape, and trained a version of the model on each one. The computational airflow models we used to train the PINN were airflow around : a simple car, a building, and an airfoil, each with 40 epochs.

In addition to training one for each of these individually, we also trained a model on a combination of them all with 120 epochs. We did this by switching which model it was trained on for each epoch sequentially. We aren't sure and will probably need further testing to see if this was the most effective or efficient method.

Once we had each of these four models (3 individually trained on single shapes and one trained on all the different shapes), we then tested it on the cylinder airflow model which wasn't used in the training of any of the models as we wanted to specifically see how they would work on shapes they hadn't seen before.

RESULTS

Below are the different shapes being used to test the models' peformances.

Airfoil Tests					
Model	u	v	p	$\lambda 1$	$\lambda 1$
Cylinder	1.04	0.99	inf	0.02	0.75
Car	1.32	1.04	inf	0.04	0.91
Building	2.23	3.07	inf	0.17	1.27
Airfoil	2.11	1.20	inf	0.12	0.55
Cylinder-Car	1.49	1.016	inf	0.06	2.91
Cylinder-Building	1.00	1.19	inf	0.16	0.07
Cylinder-Airfoil	1.03	1.25	inf	0.13	0.52
Car-Building	1.59	1.17	inf	0.13	0.98
Car-Airfoil	1.83	0.97	inf	0.09	1.12
Building-Airfoil	1.02	1.03	inf	0.08	1.05
Cylinder-Car-Building	1.31	1.25	inf	0.11	1.08
Cylinder-Car-Airfoil	1.09	1.02	inf	0.10	2.64
Cylinder-Building-Airfoil	1.06	0.99	inf	0.08	0.57
Car-Building-Airfoil	1.56	2.20	inf	0.17	0.65
Cylinder-Car-Building-Airfoil	1.14	1.04	inf	0.11	1.37

Cylinder Tests					
Model	u	v	p	$\lambda 1$	$\lambda 1$
Cylinder	1.08	1.40	9704.15	0.02	0.76
Car	0.91	2.40	6802.09	0.04	0.91
Building	121.99	523.66	60629.73	0.17	1.27
Airfoil	798.16	3688.13	78254.74	0.12	0.55
Cylinder-Car	1.19	1.65	12046.18	0.06	2.91
Cylinder-Building	14.55	132.57	8416.237	0.16	0.07
Cylinder-Airfoil	25.90	680.59	905.79	0.13	0.52
Car-Building	24.83	192.62	37388.11	0.13	0.98
Car-Airfoil	9.58	41.55	27326.38	0.09	1.12
Building-Airfoil	17.46	234.19	5182.07	0.08	1.05
Cylinder-Car-Building	4.077	18.87	17294.91	0.11	1.08
Cylinder-Car-Airfoil	38.48	181.19	13125.18	0.10	2.64
Cylinder-Building-Airfoil	25.73	620.90	11438.77	0.08	0.57
Car-Building-Airfoil	19.62	84.46	9049.02	0.17	0.65
Cylinder-Car-Building-Airfoil	56.45	61.95	10694.19	0.11	1.37

Building Tests					
Model	u	v	p	$\lambda 1$	$\lambda 1$
Cylinder	1.17	2811.82	inf	0.02	0.75
Car	1.75	27297.64	inf	0.04	0.91
Building	6.39	2130509	inf	0.17	1.27
Airfoil	38.37	4889911	inf	0.12	0.55
Cylinder-Car	2.14	33698.78	inf	0.06	2.91
Cylinder-Building	1.32	460241	inf	0.16	0.07
Cylinder-Airfoil	1.52	1430978	inf	0.13	0.52
Car-Building	7.04	1482711	inf	0.13	0.98
Car-Airfoil	8.27	980395.2	inf	0.09	1.12
Building-Airfoil	2.76	1010753	inf	0.08	1.05
Cylinder-Car-Building	3.01	244685	inf	0.11	1.08
Cylinder-Car-Airfoil	5.28	6711215	inf	0.10	2.64
Cylinder-Building-Airfoil	2.16	1257593	inf	0.08	0.57
Car-Building-Airfoil	3.04	3875865	inf	0.17	0.65
Cylinder-Car-Building-Airfoil	4.23	175462.8	inf	0.11	1.37

Car Tests					
Model	u	v	p	$\lambda 1$	$\lambda 1$
Cylinder	1.11	1.26	inf	0.02	0.75
Car	1.58	2.81	inf	0.04	0.91
Building	48.83	307.37	inf	0.17	1.27
Airfoil	149.51	759.73	inf	0.12	0.55
Cylinder-Car	1.94	1.85	inf	0.06	2.91
Cylinder-Building	1.84	45.81	inf	0.16	0.07
Cylinder-Airfoil	6.18	309.18	inf	0.13	0.52
Car-Building	21.28	171.47	inf	0.13	0.98
Car-Airfoil	7.75	43.74	inf	0.09	1.12
Building-Airfoil	7.53	155.84	inf	0.08	1.05
Cylinder-Car-Building	4.49	17.59	inf	0.11	1.08
Cylinder-Car-Airfoil	44.35	646.60	inf	0.10	2.64
Cylinder-Building-Airfoil	9.70	186.49	inf	0.08	0.57
Car-Building-Airfoil	14.70	438.85	inf	0.17	0.65
Cylinder-Car-Building-Airfoil	20.21	39.19	inf	0.11	1.37

Analyzing the data from the given confusion matrix, it is clear that every model performs relatively well when tested on the airfoil which we assume is caused by the minimal complexity of the test. From the evidence provided, we have come to the conclusion that training on foreign shapes led to an opposite effect to what we had originally anticipated. The presumption that training on more complex shapes would lead to better results when encountering foreign airfoils was incorrect as the cylinder shape model performed significantly better than the other models even when tested on their own training data.

Another assumption that had been made was that the complexity of the car flow model would generate the best results. This was proven to be partially true as the car model had only been outperformed by the cylinder model. The converse assumption had also been made about the airfoil as it was the least complex. This also proved to be a relatively correct hypothesis as it had significantly more error relative to the other models, even under performing on the data it was training on.

The model trained on all the data provided gave us underwhelming results as well, leaving us in relative confusion as to why the model with more exposure to training data was doing similar to the other models with little training data. We hypothesized that the issue could have been that the ac-

Table 1: Confusion Matrix with all the given data

Model	Airfoil	Building	Car	Cylinder
Airfoil	0.47	9.52	5.82	8.72
Building-Airfoil	0.03	7.42	3.53	5.62
Building	0.96	8.21	4.81	7.36
Car-Airfoil	0.28	7.95	2.91	5.40
Car-Building-Airfoil	0.62	8.14	4.39	5.51
Car-Building	0.31	8.08	4.10	6.12
Car	0.16	5.39	0.75	3.20
Cylinder-airfoil	0.13	7.30	3.78	5.53
Cylinder-Building-Airfoil	0.02	7.41	3.75	6.34
Cylinder-Building	0.09	6.66	2.22	5.53
Cylinder-Car-Airfoil	0.05	8.69	5.13	6.11
Cylinder-Car-Building-Airfoil	0.08	6.76	3.34	5.81
Cylinder-Car-Building	0.25	6.76	2.18	4.70
Cylinder-Car	0.21	5.59	0.64	3.36
Cylinder	0.02	4.05	0.17	3.20

curacy of the car and cylinder models were hindered by the other models who had worse results individually.

These results told us that our model was not training the simulation data optimally. When we ran the simulations, we noticed that the simulations did not all run for the same amount of time. Despite having even time steps at first, the simulations sometimes ended at different and awkward times. This means that our models might have been learning inconsistent relationships between the physics of our simulations and the loss. Furthermore, the simulations are a sort of black-box technology. The outputs of the binary files are not very clear and exploring alternatives for converting data between our simulation platforms to neural network input is part of our future goals.

Looking at our models' errors in the parameters given left us with inconclusive assumptions that made us question the accuracy of our results. One clear outlier from our results was the sizeable error in pressure, either being a substantial amount or just outright infinite. Another area of anomalous error was the recorded values for the velocity for all models in the building tests, giving us substantial error in addition to the infinite pressure error. Although as a collective we were able to bring forth our own conclusions given the data, we realize that fixing the issues stated above would lead to significantly more accurate results.

FUTURE WORKS

The major setback we had faced in regards to testing the model on new air flows was the lack of data we had to train our model. The original incentive for the use of the OpenFoam software was to export the spatial coordinates, flow velocity, and pressure at specific time points for a multitude of airfoils into a .csv file to use as either train or test data. Due to the complexity of the OpenFoam software, we did not have sufficient time to figure out how to generate our own data. Because of this difficulty, we were forced to use the limited data sets provided in OpenFoam, most of which were unusable for our task. Given more data to train the model, we would be able to create more accurate and substantial results when testing the model on other airfoils.

References

[1] Elijah Ang and Bing Feng Ng, "Physics-Informed Neural Networks for Flow Around Airfoil," *Aerospace Research Central*, Dec. 29, 2021, doi: 10.2514/6.2022-0187.

[2] Maizer Raissi, "Physics Informed Neural Networks," *GitHub*, May 26, 2020.

[3] Maizer Raissi et al., "Physics Informed Deep Learning (Part I): Data-driven Solutions of Nonlinear Partial Differential Equations," *arXiv*, Nov. 28, 2017, doi: 10.48550/arXiv.1711.10561.