ACCELERATING CFD USING MACHINE LEARNING

SRCFD: A FRAMEWORK FOR SUPER RESOLVING COARSE-RESOLUTION TO FINE- RESOLUTION SIMULATIONS

Pradeep Singh*
pradeepsingh7890@live.com
Computational Science Research Center
San Diego State University
CA, USA

Nath Gopalaswamy, Francisco Martínez
{Nath.GOPALASWAMY, Francisco.MARTINEZ}@3ds.com
Dassault Systemes Simulia Corporation.
Rhode Island, USA

ABSTRACT

In this paper, we present a machine learning framework for accelerating CFD simulations by super-resolving coarse resolution simulation into fine resolution simulation using convolutional neural network. The basic idea behind our framework is to exploit the similarities between coarse and fine resolution simulations by learning end-to-end mapping between them and this is done by training an (graph) convolutional neural network model on input-output pairs where inputs and outputs are coarse resolution simulations and fine resolution simulations respectively. We test our model on different shapes and geometries and present our result. We also show that our framework can super-resolve by a factor of 2, 3 and 4. Finally, we will discuss how we can extend our work to predict super-resolved simulations in time and other future projects.

1 INTRODUCTION

CFD simulations are known for their high usage of computational resources, both in terms of memory and CPU/GPU time. While direct numerical simulations (DNS) of turbulent flows, down to the Kolmogorov scale, are still out of reach of most practical fluid flow cases (NASA vision CFD 2030), over the past decades a series of increasingly rich turbulence-modeling methods have been invented. This has led to the establishing of CFD as a practical, oftentimes go-to tool to assist engineers in the design and optimization process. Still, CFD is far from a solved problem, and a good deal of expertise is required from CFD engineers to discern which simplifications and models are viable for a given flow problem.

Beyond the method choice, for any given CFD method there is a number of parameters that can be tuned and adjusted, and only through comprehensive checks, convergence studies and comparisons, both with other methods and with experiment, the necessary craft is acquired to make CFD a valid tool for a particular problem (CFD High Lift Prediction Workshop).

A good part of this work is currently at the boundary of what is computationally feasible given the resources of CFD-using organizations. A "simple" reduction in spatial resolution by a factor of 2 for any given three-dimensional simulation amounts to a reduction in a factor between 8 and 16, depending on the nature of the numerical method.

When the size of the computational support (mesh, lattice) is progressively refined in convergence studies, a series of inter-related simulations are obtained. The goal of the CFD practitioner is typically to refine the support until a convergence in the solution is obtained. Different methods trade faster or slower convergence for lower or higher accuracy. The nature of the relation between one simulation and the following one is far from simple, as it is between different methods.

The goal of this work is to study whether machine-learning can be effectively used to predict such relations, and if so, to speed up the convergence of simulation series, by rendering the finer steps

*Work done while author was intern at Dassault Systemes Simulia Corp., RI.

unnecessary. We are proposing a convolutional neural network based framework that could learn end-to-end mapping between coarse resolution and corresponding high resolution simulation. This model is able to predict the high resolution simulation given its low resolution and could bring down the simulation time by a factor of 2 (or even more) at least.

Our original contributions are, firstly, we introduce a framework that can accelerate CFD simulations by super-resolving coarse (low) resolution simulations into fine (high) resolution; and as per our knowledge there is little to no work in this field. Secondly, we also introduce and apply graph convolutional neural networks (Kipf et al., 2017) to unstructured meshes (by converting it to graphs).

2 BACKGROUND

2.1 ML FOR CFD

Machine learning in CFD can be broadly divided into two parts: dimensionality reduction and reduced order modeling. Dimensionality reduction involves extracting key features and dominant patterns that may be used as reduced coordinates where the fluid is compactly and efficiently described (Taira et al. 2017). Reduced-order modeling describes the spatiotemporal evolution of the flow as a parametrized dynamical system (ML for Fluid Mechanics, Brunton et al. 2020).

As reported in ML for Fluid Mechanics (Brunton et al. 2020), there have been significant efforts to build methods and models that can extract, reduce, simplify or capture the dynamics of flow physic; however any such model will be intrusive, requiring human expertise to build models from working simulation. To this end, with advancement in simulation capabilities and experimental techniques, fluid dynamics is becoming a data rich field, thus becoming amenable to data-driven methods or machine learning algorithms as we call them (Brunton et al. 2020).

2.2 SUPER-RESOLUTION

Super-resolution is a well-known and defined problem in engineering and computer science (Irani et al., 1991). As reported in Dong et al., 2015, most of the work in this field mostly uses the example-based strategy. These methods either exploit internal similarities of the same image or learn mapping function from low- and high-resolution pairs. But most of these methods included series of pre-processing steps on input-output (images) pair and have rarely optimized the super-resolution pipeline, until a deep convolutional neural network based pipeline was introduced by Dong et al., 2015 that could learn end-to-end mapping between low- and high-resolution images.

Since then, many CNN based methods have been developed that achieve state-of-the-art performance on various kinds of super-resolution problems. For example, in image super-resolution based problems, methods using GANs (Ledig et al., 2017), Skip Connections and Network in network (Yamanaka et al., 2017), Residual networks (Ahn et al., 2018) and many more (Wang et al., 2019) were introduced.

Super resolution based approaches have also been introduced in tasks like Video Super-resolution (Wang et al., 2019; Younghyun Jo et al., 2018), 3D object Super-resolution (Smith et al., 2018), Point Cloud Super Resolution (Huikai Wu et al., 2019) based problems. Taking inspiration from all these methods we have developed a method that could accelerate CFD simulations by super resolving coarse simulation into fine.

2.3 SUPER-RESOLUTION ON NON-EUCLIDEAN DATA

So far most of the success in applying machine learning to different kinds of problems is limited to euclidean data only. In recent times, machine learning community has shown great deal of interest in applying machine learning to unstructured or non-euclidean data (Bronstein et al., 2017) as well, thus giving birth to new field known as, "Geometric Deep Learning".

Geometric Deep Learning is mostly driven by modeling unstructured data as a graph and then applying graph convolutions neural networks (Zonghan Wu et al., 2019) on it. Geometric Deep Learning can be broadly divided in two parts depending upon how convolutions on graph is defined.

First, Spatial-based ConvCNNs, where convolution on graph is inspired by classical convolution on images. In this method, we model convolution on the basis of relation between central nodes and it's neighbours; representation of central nodes is convolved with it's neighbours to get the updated representation of central nodes. This sort of methods have many limitations when it comes to aggregating features from (1-hop, 2-hop, etc.) neighbours but is quite intuitive and have show great results in classification based problem (Kipf et al., 2017).

Second, Spectral-based ConvGNNs, where convolution is defined using filters from spectral graph theory (Schuman et al., 2013).

3 OUR FRAMEWORK

We named our framework SRCFD, which stands for Super Resolution Computational Fluid Dynamics. It takes a coarse resolution mesh as input and up-scales it into a fine resolution mesh along with keeping shape, geometry of mesh intact and can works on any kind of mesh – structured, unstructured and multi resolution mesh. In it's current form, our framework is limited to 2D geometry.

At it's core our framework uses two different kinds of convolution neural network: classical convolutional neural network for structured mesh and graph convolutional networks for unstructured mesh. In following sections we will describe both the approaches separately.

3.1 NOTATIONS

G = G represents our graph
V = vertices/ nodes of G
E = edges between nodes V of G
A = adjacency matrix of a graph G
c = number of channels in input field
f = number of filters in feature extraction block
f_s = filter size in feature extraction block
d = number of filters in downsampling block
d_s = filter size in downsampling block
t = number of filters in transformation block
m = number of layers in transformation block
t_s = filter size in transformation block
u = number of filters in upsampling block
u_s = filter size in upsampling block
r = number of filters in reconstruction block
r_s = filter size in reconstruction block
p = leakage coefficient for PReLU

3.2 FORMULATION

Consider a low- and high-resolution[1] simulation denoted as x and y. Our goal is to learn a function $f(x, y)$, which given pairs of x and y, should be able to predict $\hat{y}$, where dimensions of y and $\hat{y}$ are same.

$$f(x, y) \xrightarrow{Predict} \hat{y} \tag{1}$$

3.3 MODEL

We can broadly divide our framework into two parts depending upon the type of mesh it can process – structured mesh and unstructured mesh.

[1]We consider coarse- and fine-resolution simulation as a low- and high- resolution input and output pair and throughout this paper, we will address both interchangeably.

3.4 Structured Mesh

For the structured mesh based problems we use classical convolutional neural networks to build models. We can broadly divide our model into 5 blocks: feature extraction, down-sampling, transformation, up-sampling and reconstruction block, of which reconstruction block is build using deconvolution layers and rest all blocks are build using convolution layers. Output from all convolution layers, also known as feature maps are transformed using parametric ReLU activation function.

3.4.1 Feature Extraction

The Feature extraction block directly takes different fields as input depending upon different architectures as described in section 4 and extract features from it using convolution filters of size 3 by 3. It then transforms extracted features using PReLU activation function.

Our feature extraction layer is expressed as an operation:

$$F_1(X) = \sigma(W_1 * X + B_1), \tag{2}$$

where, W_1 and B_1 represents the filter weights and biases respectively. Here W_1 is of a size $c * f_s * f_s * f$, where c is the number of channels in the input field, f_s is the spatial size of filters, and f is the number of filters. B_1 is an f-dimensional vector, whose each element is associated with a filter. We then transform the feature maps by Rectified Linear Unit and this transformed feature map is then fed into next layer in our model.

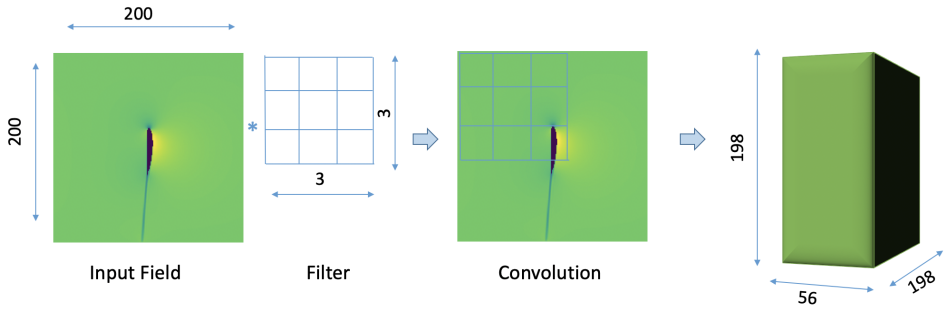

Figure 1: Feature Extraction Block

The dimension of filters play a very important role in extracting right kind of features. We have experimented with different filter size like, 2 x 2, 3 x 3, 4 x 4 and 5 x 5. Among all these, 3 x 3 filters have worked very well for us.

Another important parameter in this block is f the number of filters; in our experiments we have seen that setting this parameter arbitrarily somewhere in the range 36 to 52 gives good and consistent behavior.

3.4.2 Downsampling

The intuition behind downsampling block is to put constraint on our model to reduce the size of feature maps by fixing the size of filters to 1 and using less number of filters as compared to previous block. This reduces the feature maps dimensions from f to d. Given that architecture for super-resolution based models are usually very deep, having downsampling block reduces the computational cost by a factor of f - d. This also helps in feature extraction by selecting most useful features from previous block.

The downsampling layer is expressed as:

$$F_2(X) = \sigma(W_2 * F_1(X) + B_2), \tag{3}$$

where, W_2 and B_2 represents the filter weights and biases respectively. Here W_2 is of a size $f * d_s * d_s * d$, where f is the number of channels in the input to downsampling layer, d_s is the spatial

size of filters, and d is the number of filters. B_2 is an d-dimensional vector, whose each element is associated with a filter.

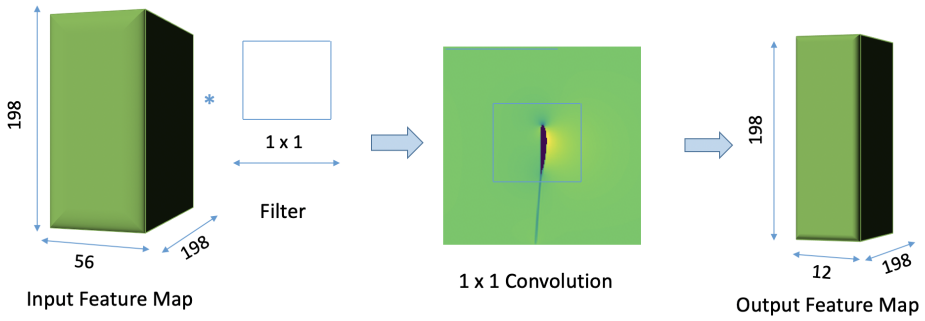

Figure 2: Downsampling Block

3.4.3 TRANSFORMATION

Transformation block non-linearly maps the extracted features from one dimension to other dimension. This is by far the most important block in entire pipeline. As per our experiments having more number of layers in this block leads to better quality output. Number of layers and filter size is denoted by m and t_s respectively. We have chosen m and t_s to be 4 and 3 by 3 as per our experiments.

The transformation layer is expressed as:

$$F_3(X) = \sum_{n=1}^{m} \sigma(W_3 * F_2(X) + B_3), \tag{4}$$

where, W_3 and B_3 represents the filter weights and biases respectively. Here W_3 is of a size $d * t_s * t_s * t$, where d is the number of channels in the input to transformation block, t_s is the spatial size of filters, and t is the number of filters. B_3 is an t-dimensional vector, whose each element is associated with a filter and m is number of transformation layers.

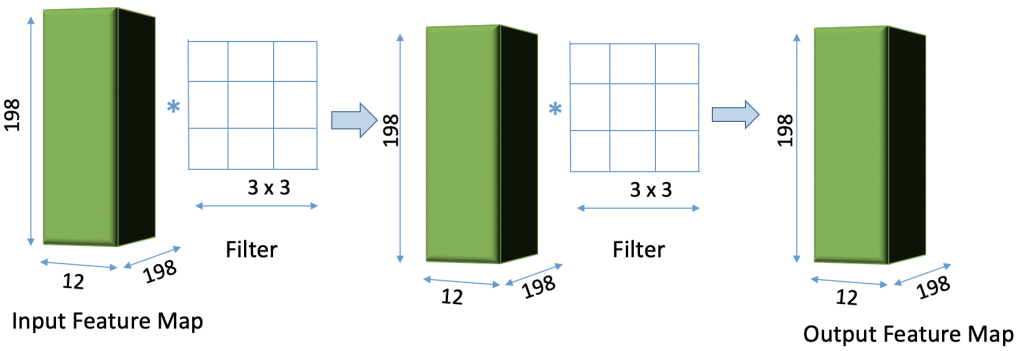

Figure 3: Mapping Block

3.4.4 UPSAMPLING

First step towards super resolving a LR input is taken in upsampling block. Upsampling block increases the dimensions of features after they have been transformed from High dimensional space to low dimensional space by previous layers. It has been noted that without upsampling block the super resolved output will be of poor quality.

The upsampling block layer is expressed as:

$$F_4(X) = \sigma(W_4 * F_3(X) + B_4), \tag{5}$$

where, W_4 and B_4 represents the filter weights and biases respectively. Here W_4 is of a size $t * u_s * u_s * u$, where t is the number of channels in the input to transformation block, u_s is the spatial size of filters, and u is the number of filters. B_4 is an u-dimensional vector, whose each element is associated with a filter.

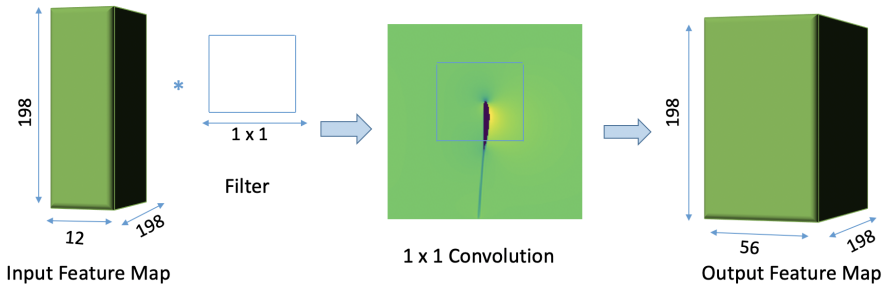

Figure 4: Upsampling Block

For upsampling block, filter size u is fixed to be 1, just like the downsampling block and number of filters u_s are set to equal to f_s.

3.4.5 Reconstruction

Reconstruction block is built using deconvolution layers unlike all other blocks. In this block we finally reconstruct the high-resolution output using feature that has been extracted, transformed and learned by our model. The filter used are of size 5 x 5 and determines the dimensions of predicted field. The reconstruction block layer is expressed as:

$$\hat{Y} = \sigma(W_5 * F_4(X) + B_5), \tag{6}$$

where, $\hat{Y}$ represent predicted field, W_5 and B_5 represents the filter weights and biases respectively. Here W_5 is of a size $t * r_s * r_s * r$, where t is the number of channels in the input to transformation block, r_s is the spatial size of filters, and r is the number of filters. B_5 is an r-dimensional vector, whose each element is associated with a filter.

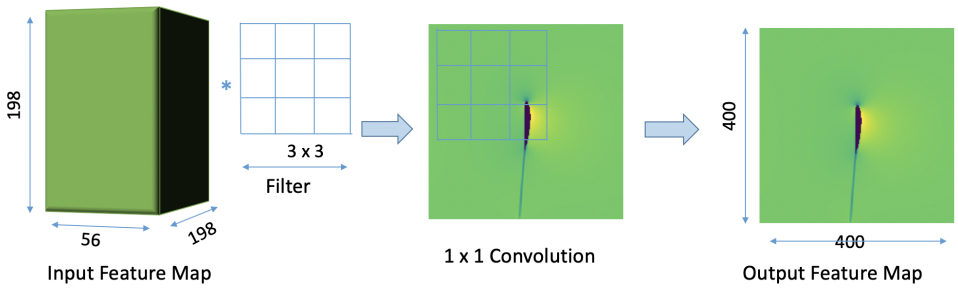

Figure 5: Deconvolution Block

3.5 Unstructured Mesh

Unlike regular domain, like structured mesh where we already have well defined ways to define convolution operators, there is no direct or trivial way to define feature mining operators. To solve

this problem, we propose that if we can convert (unstructured) mesh into a graph, we can define something like graph convolutions on it, this will solve our problem of not being able to define convolutions on unstructured mesh.

3.5.1 Graph Convolutions

Unlike euclidean or regular domain, where data follow a well defined structure and always have ordered and fixed number of neighbors, in graphs every node have unordered and variable number of neighbours. So, to define convolution on graph we make use of connectivity between nodes to aggregate features. We simply take average value of the features of the node and it's neighbours.

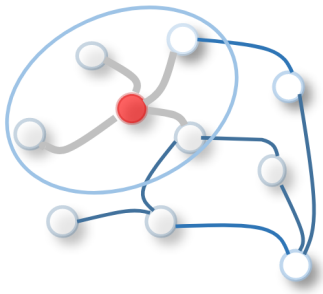

Figure 6: Graph Convolution

For example in the above figure for red node, its aggregated features would be the average of its own features and its neighbours represented in that circle. This simple idea leads to a new kind of neural network called Graph convolution network.

3.5.2 Graph Convolution Network

Graph convolution network (GCN) is a neural network that operates on graphs. Let's say we have a simple graph represented by $G = (V, E)$, where $V = 1, 2, ...N$ are N number of nodes connected together by set of edges E. Then GCN will take as input:

1. an feature matrix X of size N x F_1, where N is the number of nodes and F_1 is the number of input features for each node, and
2. an matrix representation of the graph structure of size N x N such as the adjacency matrix A of G.

and produces an output matrix Z of size N x F_2 where F_2 is the number of output features per node. In GCN, hidden layer is represented as,

$$H^{(l+1)} = f(H^{(l)}, A), \tag{7}$$

with $H(0) = X$, f is the propagation and $H(L) = Z$, and L being the number of layers. With this, we can define propagation rule in GCN like this;

$$f(H^{(l)}, A) = \sigma(AH^{(l)}W^{(l)}), \tag{8}$$

where, $W^{(l)}$ is a weight matrix for the l-th neural network layer and $\sigma()$ is a non-linear activation function like the ReLU.

3.5.3 GCN Model

We can divide graph based model into 3 parts: Input block, GCN block and Output block. Input block takes LR resolution graph and transforms it into HR graph by using Nearest-neighbour interpolation in N dimensions and then interpolated HR graph is fed into GCN block. GCN block is consist of multiple layers built using graph convolutional network. All the layers aggregate features

from one hop neighbours. After every layer, new updated features are transformed by an activation. Output block gets input from GCN block and it then transforms it using identity activation function. Output block also predicts the final features for every node.

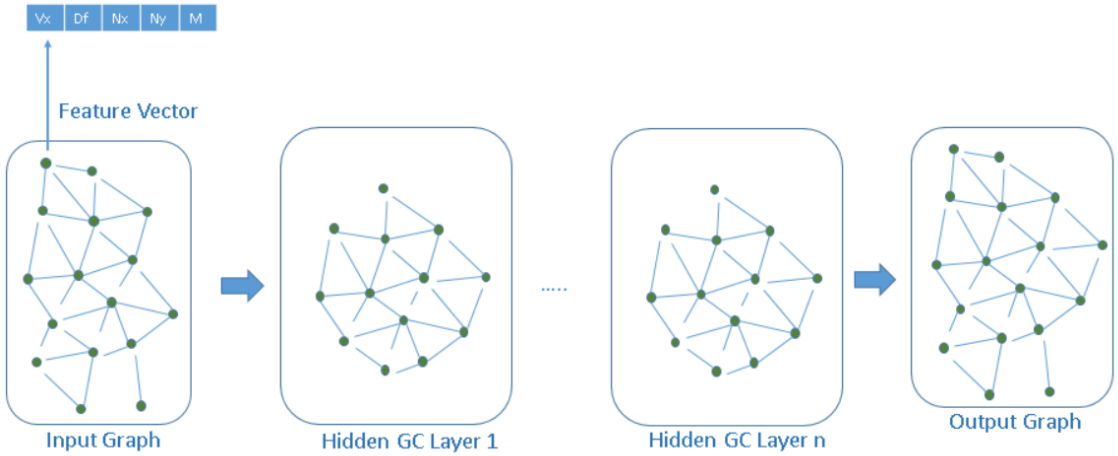

Figure 7: Graph Convolution Network

3.6 Activation Function

We have used PReLU as our activation function, which is very much similar to leaky ReLU, except that coefficient of leakage, p here is considered as a parameter and is learned as part of training. We have tried other activation functions as well, like ReLU, tanh, sigmoid, but ReLU and it's variants gave us the best results.

$$f(x) = \begin{cases} x & \text{if } x > 0 \\ ax & \text{otherwise} \end{cases} \tag{9}$$

where a is learn-able parameter which controls the leakage.

3.7 Loss Function

All machine learning models compute a loss functions which is minimized using stochastic gradient descent with the standard backpropagation to learn parameters $\theta = \{w, b\}$, where w and b are weights and biases of all the layers.

In our model, we employ Mean Squared Error (MSE) as the loss function. Through MSE we compute loss between predicted output $\hat{\mathbf{Y}}$ and original ground truth high-resolution output $\mathbf{Y}$. So, given a set of low- and high-resolution pairs $_(X_1, Y_1), (X_2, Y_2), ..., (X_n, Y_n)$, we compute MSE as:

$$L(\theta) = \frac{1}{n} \sum_{i=1}^{n} \left\| \hat{\mathbf{Y}} - \mathbf{Y} \right\|^2 \tag{10}$$

where n is the number of LR-HR pairs in our training set.

4 Architectures

We have developed different models that takes different inputs in different ways that helps our model to learn mapping between low- and high-resolution input-output pairs. On the bases of how you feed inputs we can divide our framework into 3 parts:

1. Basic model: Take single input field
2. Stacked Input Model: Take multiple fields as input stacked one top of the other.
3. Multi Input Model: Take multiple fields as input not stacked together.

4.1 Basic Model

In basic model, a single field is passed as input-output (LR-HR) pair during training and model learns to maps LR to HR field. Then at testing time, we just feed LR input field and model predict corresponding HR field.

But during our experiments, we have observed that this model could be limiting in the sense that it won't be able to predict given field for varying shapes and geometry as CFD simulations are non-linear in nature and going from coarse (LR) to fine (HR) simulations doesn't just depend on one field. Rather other fields like pressure field, normal fields also play import role how LR simulations looks in HR. So, in order to tackle this problem we propose two models as mentioned in section 4.2 and 4.3.

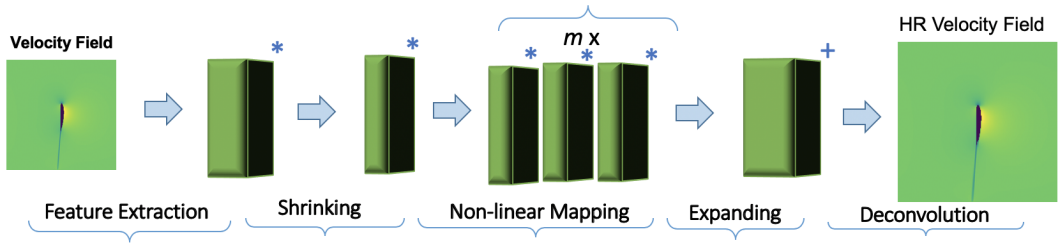

Figure 8: Basic Model

where, $*$ is convolution layer and + is deconvolution layer.

4.2 Stacked Input Model

In stacked input model, we stack different input fields one top of the other as mentioned in section 5.2. And then, convolution filters are applied along depth of the stacked inputs at the feature extraction step.

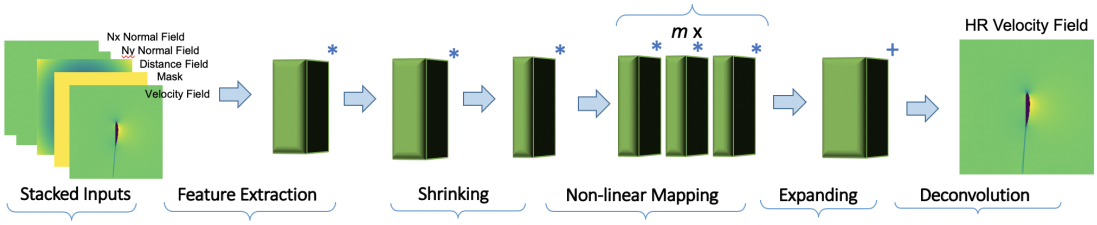

Figure 9: Stacked Input Model

where, $*$ is convolution layer and + is deconvolution layer.

4.3 Multi Input Model

In multi input model, we process all the fields separately unlike stacked input model. In this case, all input fields are passed through different convolution layers and then corresponding feature maps are merge together in feature aggregation step.

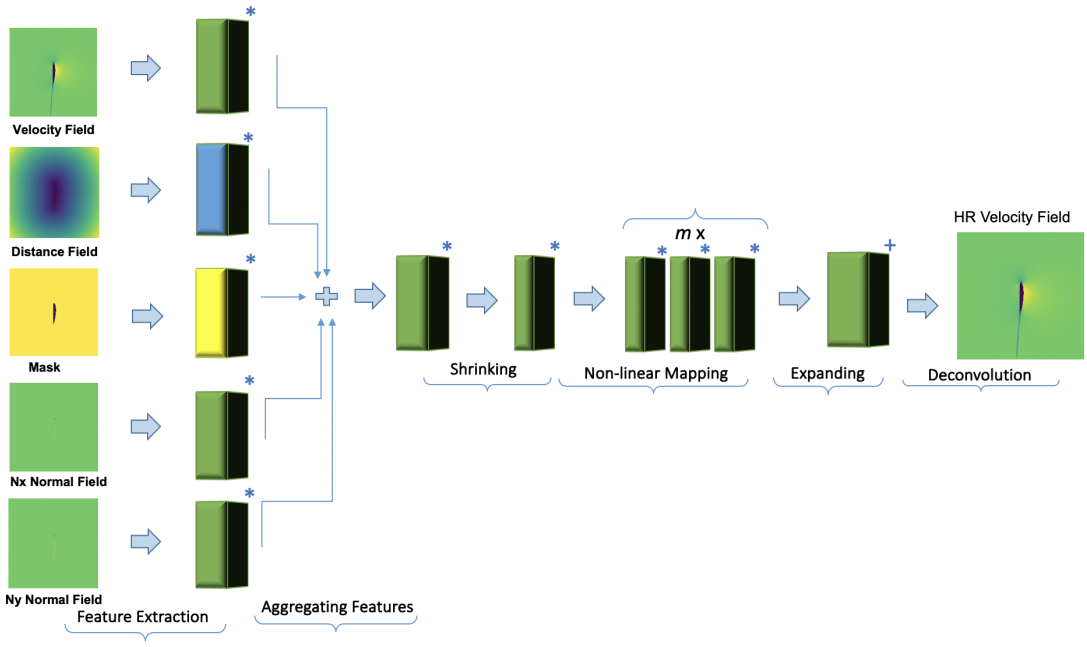

Figure 10: Multi Input Model

where, $*$ is convolution layer and + is deconvolution layer.

5 Dataset

We have build our own in-house dataset for training and testing purpose using XFlow. Our dataset is consist of 55 samples for both low- and high-resolution simulations for each shape. As of now we have 2 shapes in our dataset: NACA 4412 Airfoil and Cylinder.

In order to make every sample different from each other, we changed two parameters while generating data: Angle of attack (AOA): -10 to 10 (degrees) Velocity value: 10 to 50 (m/sec)

Our dataset is consist of 5 different fields, V_x, V_y, S_p, N_y, N_x, D_f and $Mask_f$.

5.1 Preparing Data for Training and Testing

To prepare data for training and testing we divide our dataset into two parts: Train set and Test set. Train and Test are consist of 50 and 5 samples respectively. We don't expose our model to any sample in the test set while training models or even any other kind of experiments. Test set for our model is as good as out of sample set.

5.2 Pre-processing Data

In theory, it's not necessary to normalize or scale data as long as data is in numeric format. But in practice, having features on similar scale makes training more efficient and leads to better prediction. Different shapes and geometry will have different scale for pressure and velocity values, so to be able to generalize and predict on data with varying scale we propose to normalize our data using following techniques.

Standard Scaler : Standardize features by removing the mean and scaling to unit variance and is calculated as:

$$z = \frac{(x - u)}{s} \tag{11}$$

where x is the velocity field, u is the mean of x and, s is the standard deviation of x.

Min-Max Scaler : Transforms features by scaling each feature to a given range, for example between zero to 1. It is calculated as:

$$z = \frac{(x - max)}{(max - min)} \tag{12}$$

where x is the velocity field, max is the maximum value in x and, min is the minimum value in x.

5.3 STACKING INPUT FIELDS

While training and designing our model it was clear to us that we might need to provide more information to our model at the training time as learning how to super-resolve a low-resolution simulation into high-resolution is a non-linear and very difficult task.

To tackle such a problem, we propose that giving different kind of fields as input to the model will help to learn how to super-resolve. And, this will also help in generalizing on different shapes and geometry.

So, say if dimensions of all the fields are:

V_x: [200, 200, 1]
N_y: [200, 200, 1]
N_x: [200, 200, 1]
D_f: [200, 200, 1]
$Mask_f$: [200, 200, 1]

Then, stacked Input: [200, 200, 5], where 5 represent five different fields, which are stacked one top of the other.

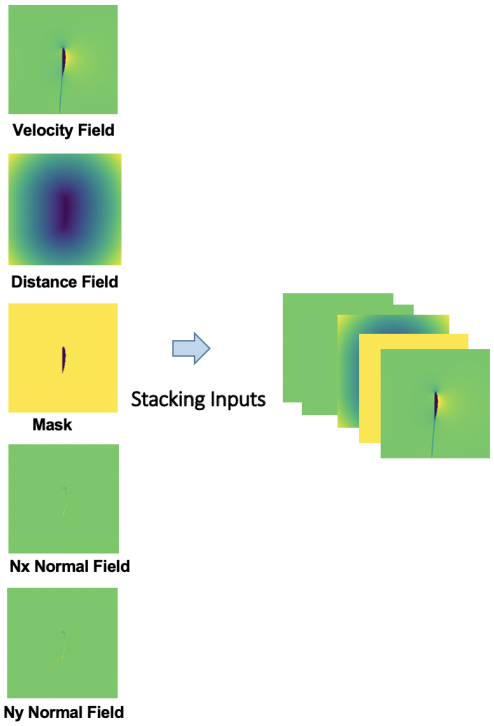

Figure 11: Stacked Input

5.4 Distance Field

We use signed distance function (SDF) to compute the distance field for every input field. Signed distance function provides a universal representation for different geometry shapes and works efficiently with neural networks (Guo et al. 2016).

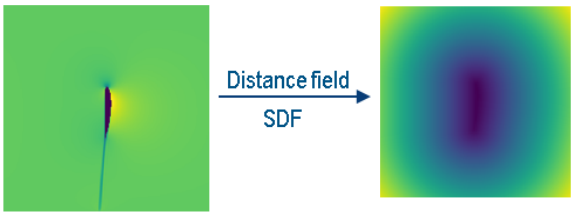

Figure 12: Generating Distance field from Velocity field for NACA 4412 Airfoil

5.5 Mask Field

We generate mask field, which is the binary representation of input velocity field. We set all velocity values inside the object to be 0 and everything on and outside boundary of object to 1. This is done to create a representation of where we (don't) want to predict values in the output field. All the values in mask field are either 1 or 0 only, everywhere we have 1, we would like our model to predict at those place and vice-versa.

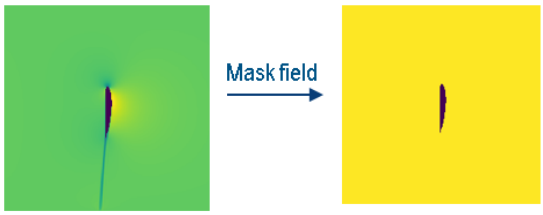

Figure 13: Generating Mask field from Velocity field for NACA 4412 Airfoil

In our experiment, we have observed without using mask field, our model predicts values inside the object as well; this is probably because of weight sharing capabilities of CNNs. To tackle this problem we are proposing to use mask field as one of the input to our model, this helps the model to learn where to predict values. Also, computing mask field is computationally efficient.

6 Implementation

We have implemented the entire project in Python and have written two packages, one for machine learning tasks and other for the data. Some other tools and packages that were used are; numpy, scipy, matplotlib, h5py.

6.1 Machine Learning

Our machine learning framework is implemented in python using tensorflow. We have named it **SRCFD** and can be used for training, evaluating, testing models and visualization of results.

6.2 Data

For data, we have used XFlow to generate training and testing samples as explained in section 5. All the data is stored in numpy array format which is easy to work with and is computationally efficient

for numerical computation like matrix multiplication. To make working with data easy, we have developed a python package, named **gen_data** for generating, processing and extracting data.

7 RESULTS

In this section we will present results from various experiments that we did as part of the project.

7.1 STRUCTURED MESH: SRCFD BASIC MODEL

We trained and tested a basic model on cylinder dataset. You can see the results in Figure 14, on the left side we have input LR velocity field, in the middle we have original HR velocity field and on the right side we have predicted velocity field by SRCFD.

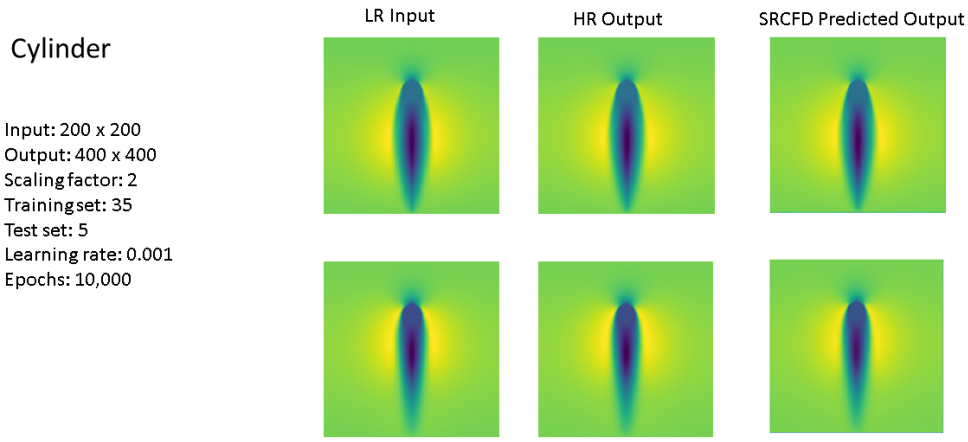

Figure 14: Results from SRCFD basic model trained on Cylinder dataset.

Now, the natural step for us was to test our model against interpolation. So, we compared our model prediction against LR interpolated velocity field as shown in figure 15.

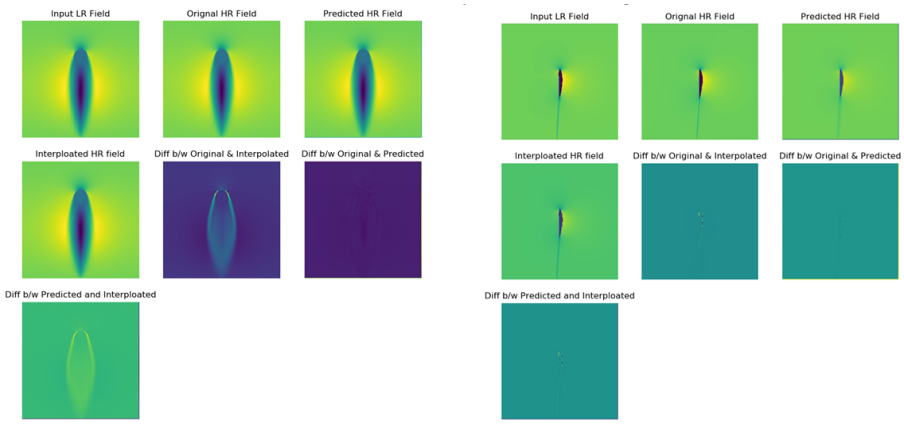

Figure 15: Comparing SRCFD basic model results against linear interpolation. Difference between predicted, interpolated and original fields. On left you see results from cylinder dataset and on right from airfoil dataset.

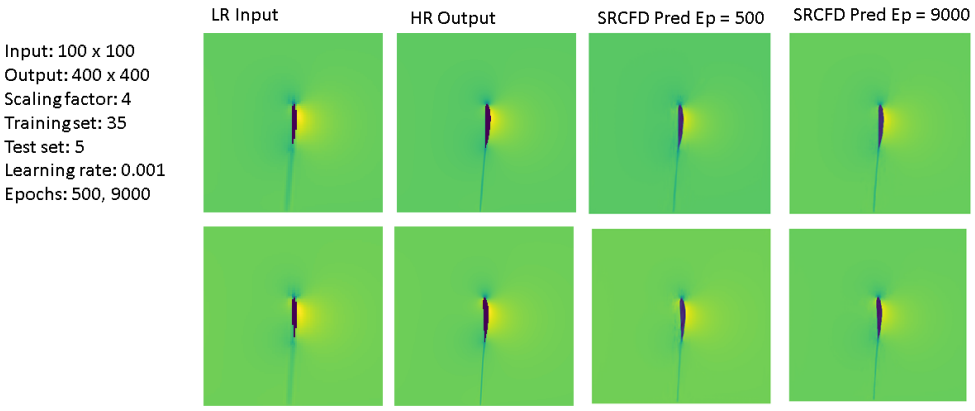

Figure 16: Results shown for super-resolution with a factor of 4

7.2 Structured Mesh: SRCFD Multi Input Model

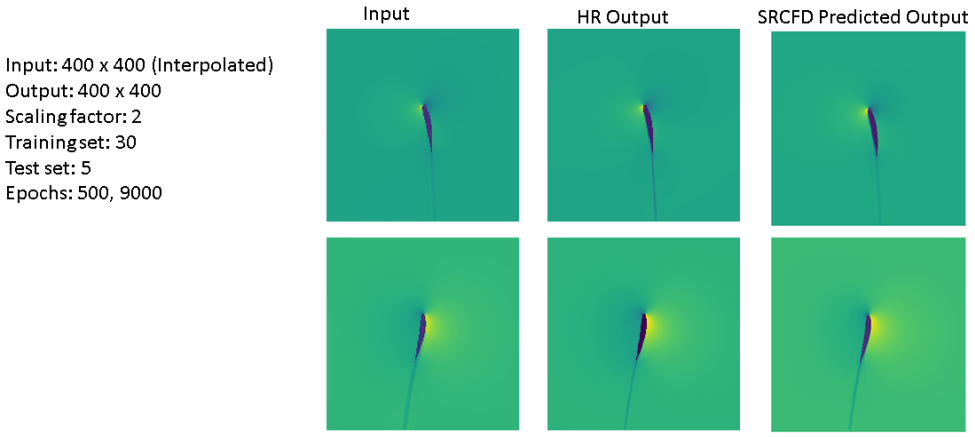

Figure 17: Results from SRCFD Multi-Input Model, trained on airfoil dataset. Left image is of coarse resolution velocity field, middle image is of original high resolution velocity field and right image is of predicted high resolution velocity field by our model

7.3 Structured Mesh: SRCFD Generalized Model - Cross Training

In this section, we show results form experiments with cross-training, where we created a dataset consist of different shapes (airfoils and cylinder) and then model is trained on both shape simultaneously. And, then in figure 19 we compare our cross-training results with interpolation.

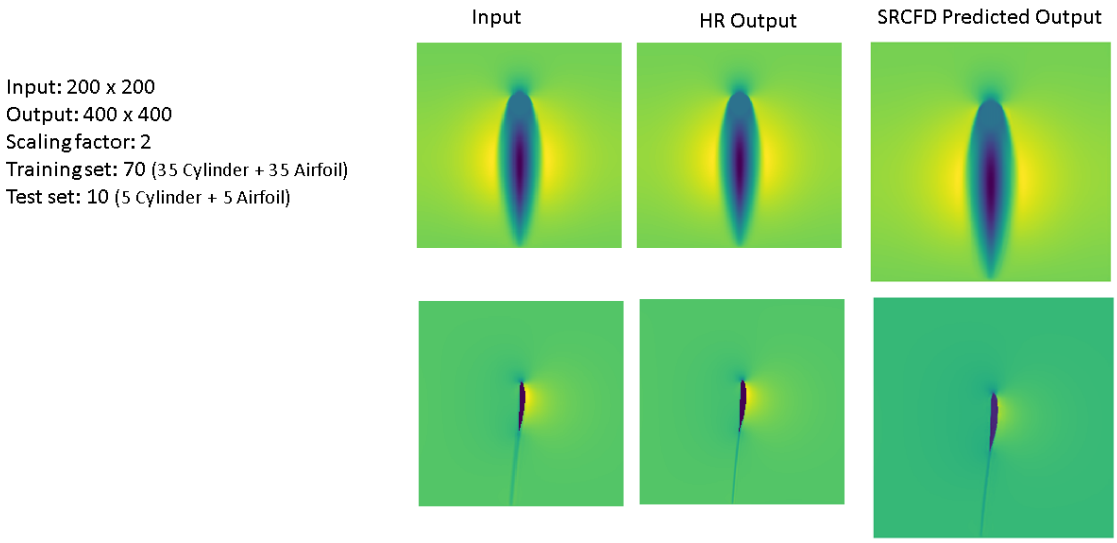

Figure 18: Results with generalized model trained on airfoil and cylinder dataset together

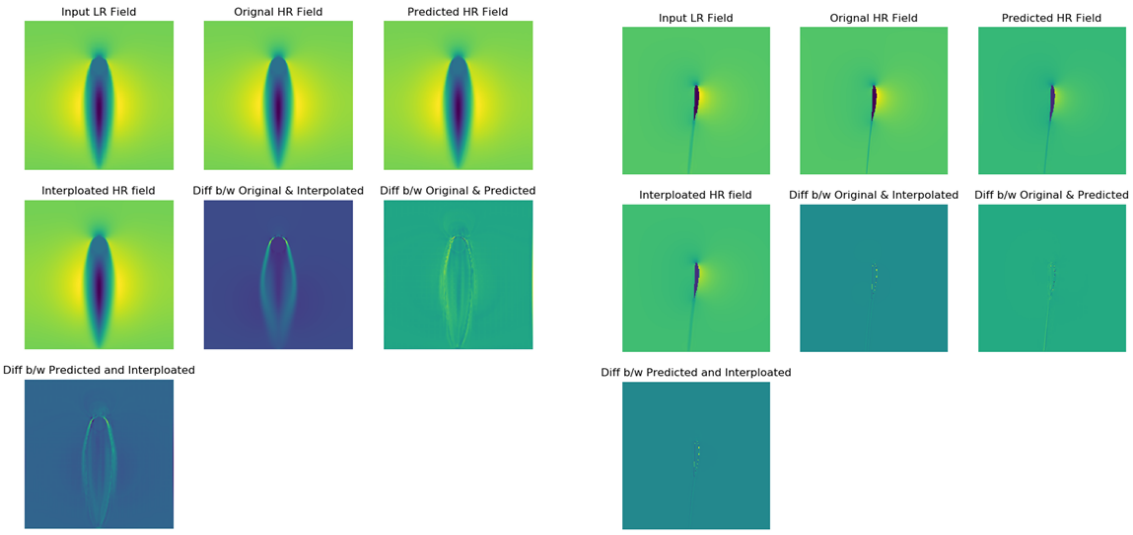

Figure 19: Comparing SRCFD Generalized model results against linear interpolation. Difference between predicted, interpolated and original fields. On left you see results from cylinder dataset and on right from airfoil dataset.

8 Conclusion

We have presented a novel machine learning framework that can super-resolve CFD simulation. We show that our framework can generalize to out of sample simulations as well and does a better job then a normal linear interpolation. We also extend our framework to unstructured and multi-resolution mesh. Our proposed framework can also be trained to super-resolve by different up-scaling factors.

9 Future Work

9.1 Predicting Super Resolved Simulation in Time

Natural extension to this project would be to predict super-resolved simulation in time. Any model that has to predict upscaled version of input and that in time has to be built using combination or mixture of convolutional neural network (CNN) and recurrent neural network (RNN).

A possible architecture for this sort of task is shown in Fig 14. We will feed input simulations in time to an RNN based encoder at every time-step. This RNN encoder is then going to encode the feature representation of input simulation at all time-step into a vector representation. This vector is then used to initialize our RNN based decoder, which using the output simulation at every step predicts the simulation in time. Simultaneously, these predicted simulation are fed into SRCFD model, that predicts the super-resolved version.

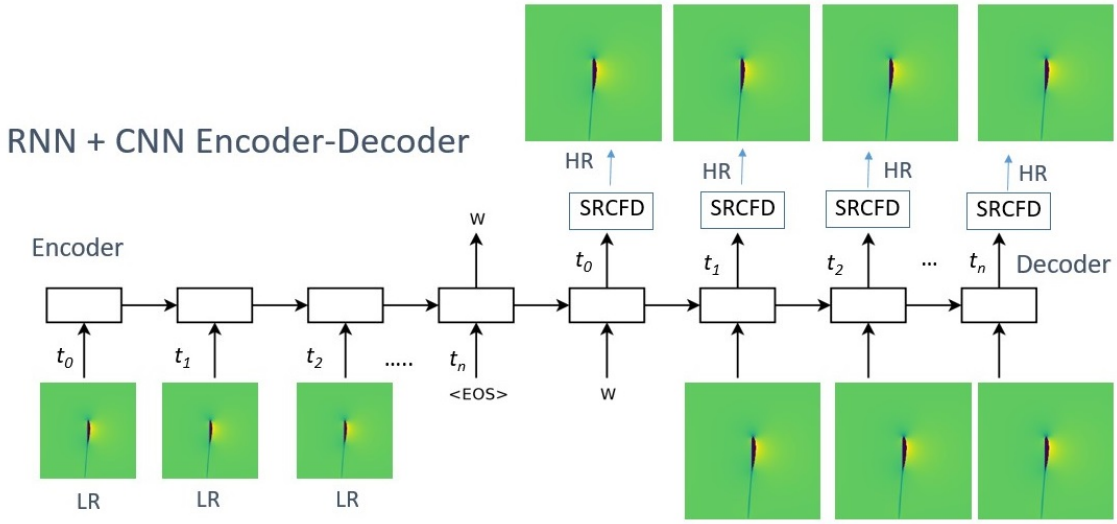

Figure 20: Architecture for Predicting SR in Time using RNN and CNN

9.2 Other Projects

Other possible projects could include:

1. To extend SRCFD framework to 3D geometry. There is still not much work done as far as SR in 3D is concerned, but with approaches like graph convolution coming up which can possibly model any kind of geometry, there is lot of value in working in this direction. Some work as far as 3D super-resolution is concerned includes, 3D appearance SR (3DASR) by Yawei Li et. al.
2. Our end goal is to accelerate CFD simulation, we will most likely have to embed ML model in our CFD solvers, so we can possibly have a project where we explore approaches how can we embedded ML in existing CFD solvers like XFlow and FMK. Other way of looking this project would be to work on a project, building models for real time super-resolution. In current machine learning framework, there are some reference for real-time super resolution, for e.g. ESPCN (W Shi et al. 2016).
3. We can explore and research more on different kind of architectures for our framework. We can research more on different kind of inputs that we could fed into our model to make our prediction more robust and accurate. We can also possibly research more on feature engineering. We can experiments with transfer learning and residual learning based architectures too.

4. Finally, if we could explore physics based learning too. Something like CFD which is highly non-linear in nature, having combination of both physics-based learning and data-driven learning approaches, would be really beneficial.

ACKNOWLEDGMENTS

I would like to thanks Ruddy Brionnaud for providing support in generating data using XFlow and the entire CFD RD team for their valuable suggestions during many discussions and team meetings. Finally this work wouldn't have been possible without the support and guidance of my managers Nath Gopalaswamy and Francisco Martinez.

REFERENCES

CFD Vision 2030 Study: A Path to Revolutionary Computational Aerosciences

4th AIAA CFD High Lift Prediction Workshop (HiLiftPW-4)

Kunihiko Taira. Modal Analysis of Fluid Flows: An Overview. In Aerospace Research Central (ARC) 2017

Steven L. Brunton, Bernd R. Noackand, Petros Koumoutsakos. Machine Learning for Fluid Mechanics. In Annual Review of Fluid Mechanics.

Xiaoxiao Guo, Wei Li, Francesco Iorio. Convolutional Neural Networks for Steady Flow Approximation. In KDD '16 Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining.

Thomas N. Kipf, Max Welling. Semi-Supervised Classification with Graph Convolutional Networks. In International Conference on Learning Representations (ICLR) 2017.

Michal Irani, Shmuel Peleg. Improving Resolution by Image Registration. In CVGIP: Graphical Models and Image Processing 1991.

Chao Dong, Chen Change Loy, Kaiming He and Xiaoou Tang. Image Super-Resolution using Deep Convolutional Networks. In IEEE Transactions on Pattern Analysis and Machine Intelligence (TPAMI) 2015.

Christian Ledig. Photo-Realistic Single Image Super-Resolution Using a Generative Adversarial Network. In Conference on Computer Vision and Pattern Recognition (CVPR) 2017.

Jin Yamanaka, Shigesumi Kuwashima and Takio Kurita. Fast and Accurate Image Super Resolution by Deep CNN with Skip Connection and Network in Network. In 24th ICONIP: International Conference on Neural Information Processing 2017.

Namhyuk Ahn, Byungkon Kang, Kyung-Ah Sohn. Fast, Accurate, and Lightweight Super-Resolution with Cascading Residual Network. In European Conference on Computer Vision 2018.

Zhihao Wang, Jian Chen, Steven C.H. Hoi. Deep Learning for Image Super-resolution: A Survey

Xintao Wang, Kelvin C.K. Chan, Ke Yu, Chao Dong, Chen Change Loy. EDVR: Video Restoration with Enhanced Deformable Convolutional Networks. In Conference on Computer Vision and Pattern Recognition (CVPR) 2019.

Younghyun Jo, Seoung Wug Oh, Jaeyeon Kang, Seon Joo Kim. Deep Video Super-Resolution Network Using Dynamic Upsampling Filters Without Explicit Motion Compensation. In Conference on Computer Vision and Pattern Recognition (CVPR) 2018.

Edward Smith, Scott Fujimoto, David Meger. Multi-View Silhouette and Depth Decomposition for High Resolution 3D Object Representation. In NeurIPS 2018.

Huikai Wu, Junge Zhang, Kaiqi Huang. Point Cloud Super Resolution with Adversarial Residual Graph Networks. arXiv:1908.02111

Michael M. Bronstein, Joan Bruna, Yann LeCun, Arthur Szlam, Pierre Vandergheynst. Geometric deep learning: going beyond Euclidean data. In IEEE Signal Processing Magazine, 2017.

Zonghan Wu, Shirui Pan, Fengwen Chen, Guodong Long, Chengqi Zhang, Philip S. Yu. A Comprehensive Survey on Graph Neural Networks. arXiv:1901.00596v3