21st Annual CFD Symposium, August 8 - 9, 2019, Bangalore

Reconstruction of Flows using Convolutional neural networks

T Sree Harsha
tsharsha1998@gmail.com
Hyderabad, Telangana

Dr. Balaji Srinivasan
sbalaji@iitm.ac.in
Chennai, Tamilnadu

Abstract

Accuracy in computation of fluid flow data using finite volume methods improves with the resolution of the mesh. Increasing mesh resolution has a trade off with time required for convergence. Coarser the grid, the lesser number of iterations are required for arriving at a converged solution. With increasingly available flow data, we expect that information contained in previously computed fine mesh flows could help in reconstructing fine mesh flows from coarse mesh flows. This reconstruction called super-resolution analysis is a popular area of research in computer science. In this study, we leverage the use of deep learning models to reconstruct fine mesh flows from coarse mesh flows. We develop and experiment machine learning models used to reconstruct fine grid flows from coarse grid flows. This results in lesser computations compared to computing the flow using a fine mesh using traditional methods. Two machine learning models have been tested; namely *Down-sampled skip-connection multi-scale convolutional neural network* (DS-MSC CNN) and a novel *encoder decoder convolutional neural network*. We assessed the performance of these two models on reconstructing lid driven cavity flows over a wide range of Reynolds numbers as a preliminary test. Both the models have shown remarkable accuracy in reconstructing fine grid flows from heavily under-resolved flows(up-to 8 times finer mesh). We also addressed the inconsistencies of padding in convolutions with boundary conditions. We experimented with different ways of padding to improve predictions at the boundary and provide a comparison using drag force as metric. With increasing fluid flow data being collected everyday, our study motivates the development of more generic, robust and flexible models for reconstruction of a variety of flows.

Keywords*: Computational methods; Convolutional neural networks; Super-resolution reconstruction; Lid driven cavity flow; Deep Learning.*

1 Introduction and Mathematical Formulation

$$\frac{\partial \rho}{\partial t} + \nabla.(\rho \vec{V}) = 0, \qquad \textbf{Continuity Equation} \tag{1a}$$

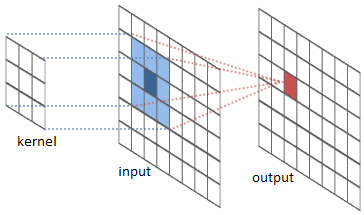
$$\rho \frac{D\vec{V}}{Dt} = -\nabla p + \nabla \vec{\tau} + \rho \vec{f}, \qquad \textbf{Conservation of momentum} \tag{1b}$$

$$\rho[\frac{\partial h}{\partial t} + \nabla.(h\vec{V})] = -\frac{Dp}{Dt} + \nabla.(k.\nabla T) + \phi, \qquad \textbf{Energy Equation} \tag{1c}$$

The partial differential equations that govern fluid flow and heat transfer (2.1, 2.2 and 2.3) do not have analytical solutions, except for very simple cases. Therefore, in order to analyze fluid flows, we split the domain into smaller sub-domains. The sub-domains are typically primitive geometric shapes - rectangles for 2-D, cuboids and tetrahedrons in 3-D. Finite volume method is used to approximate these partial derivatives over the sub-domains and model them as a set of linear equations across all mesh elements. These linear equations are then solved using

numerical procedures to arrive at a stable and converged solution. Finer the mesh, the closer we are to the actual solution. Unfortunately, the number of iterations required to arrive at a converged solution increases drastically with number of mesh elements. This motivates us to reconstruct fine mesh flows from coarse mesh flows using previously computed fine mesh flows. In this study, we leverage the properties of convolutional neural networks to arrive at the underlying non-linear mapping from coarse mesh flows to fine mesh flows.

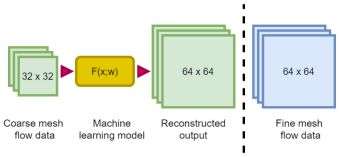
Convolutional neural network (CNN) is a powerful tool traditionally used for image and video processing applications. The basic building block of a convolutional neural network involves linear combinations of values within a receptive field followed by a non-linear activation. This property of CNNs motivates us to apply it to the domain of computational fluid dynamics.



The output for a convolution is calculated as follows

$$h_{i,j} = \sigma(\sum_{p=1}^{k}\sum_{q=1}^{k} w_{p,q} \times x_{i+p-1,j+q-1}) \tag{2}$$

where σ is a non-linear activation function, $k \times k$ is the size of the kernel
x is the input matrix, h is the output matrix



Mathematically, we formulate the problem as follows. Given the input data set $\mathbf{x} \in \mathbb{R}^{n \times n}$ and the desired output data set $\mathbf{y} \in \mathbb{R}^{m \times m}$, where $n \times n$ and $m \times m$ are coarse and fine grids respectively, we aim to find the optimal weight w in a machine learning model F that acts as a non-linear regression function such that $F(x; w) \approx y$.

In the present case, x and $F(x; w)$ represent the low-resolution and reconstructed high-resolution data, respectively. The weight w is optimized such that a loss function or criterion between the desired high-resolution output y and the machine learning model output $F(x; w)$ is optimized. So, we need to define a loss function L and rules to update the parameters w to minimize the loss function. We chose mean squared error as the loss function L and stochastic gradient descent to update the weights w.

$$L = \frac{1}{N}\sum_{i=1}^{N} ||y - F(x_i; w)||^2 \tag{3}$$

$$w = argmin_w \frac{1}{N}\sum_{i=1}^{N} ||y - F(x_i; w)||^2 \tag{4}$$

We use gradient descent to update the weights so as to minimize the loss function. Ideally, we are expected to calculate the gradients over sum of losses of all samples. This is called vanilla gradient descent algorithm. Unfortunately, this vanilla gradient descent is computationally expensive as it involves looping over the entire data-set for one update. To reach the optimum solution faster, we go with with stochastic gradient descent (or)

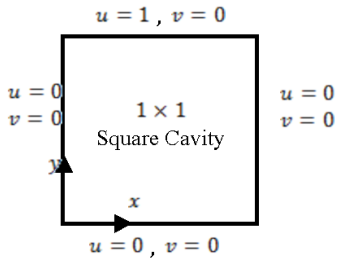
mini-batch gradient descent in which we update the weights by calculating loss over single data point (in case of stochastic gradient descent) (or) sub-set of data points (in case of mini-batch gradient descent). SGD or mini-batch GD doesn't converge to the optimal solution smoothly as we are optimizing a different objective function.

$$\theta_{j+1} = \theta_j - \eta \frac{\partial}{\partial \theta_j} \sum_{i=1}^{N} L_i, \qquad \text{Vanilla gradient descent} \tag{5a}$$

$$\theta_{j+1} = \theta_j - \eta \frac{\partial}{\partial \theta_j} \sum_{i=1}^{B} L_i - \lambda\, \theta_j, \qquad \text{Stochastic gradient descent} \tag{5b}$$

where η is learning rate, θ is any parameter, N is total number of examples
λ is coefficient of regularization, L_i is the squared loss of i^{th} example

2 Data



We report our results on lid driven cavity problem. The lid driven cavity is a well known benchmark problem for viscous incompressible fluid flow. We define a square cavity of size $1m \times 1m$ consisting of three rigid walls with no-slip conditions and a lid moving with a tangential velocity. The lower left corner has a reference static pressure of 0. We are interested in the velocity distribution for lid velocities in the range of 0.01 ms to 1.00 ms (Reynolds number from 10 to 1000).

We generated a lid driven cavity dataset with 100 different lid velocities uniformly distributed from *0.01m/s* to *1.00m/s* on a *1m×1m* cavity on meshes of size 8×8, 16×16, 32×32 and 64×64 using ANSYS Fluent.

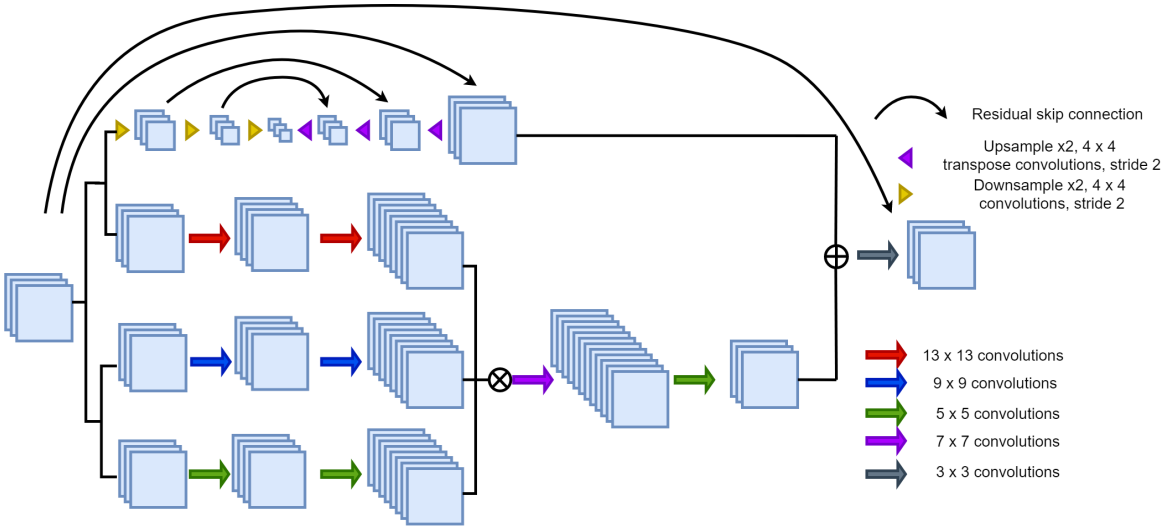
In order to avoid over-fitting, we also generated a validation set of 20 samples with intermediate values of lid velocities in range of *0.01m/s* to *1.00m/s* (which are not in training set) over meshes of size 8×8, 16×16, 32×32 and 64×64.

3 Machine Learning Models

We tested two models namely *Down-sampled skip-connection multi-scale convolutional neural network* (DS-MSC CNN) and a novel *encoder decoder convolutional neural network* for reconstructing lid driven cavity flows. The architectures and training methodologies are explained below.

Down-sampled skip connection multi-scale model

Architecture



This architecture has been inspired from the work of *Super-resolution reconstruction of turbulent flows with machine learning: Kai Fukami, Koji Fukagata and Kunihiko Taira*. It has four parallel lines, the outputs of which are concatenated in the end.

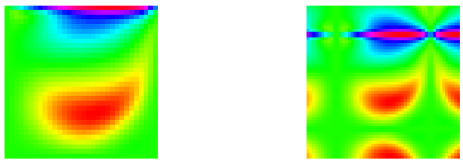
- One Down-sampling and up-sampling pipeline which increases robustness against translation and rotation of data elements
- Three 4-layer convolutions with different kernel sizes each. Using different kernel sizes help capture structures (or) vortices at multiple scales
- The network also has skip connections. Skip connections help in training a deep network better as we try to model only the error instead of the whole function. It also ensures that performance doesn't go worse with adding more layers

Methodology

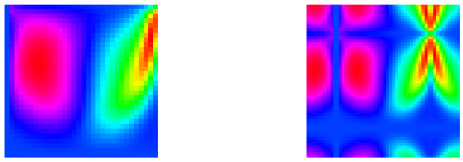
The coarse grid simulation $\mathbf{x} \in \mathbb{R}^{n \times n}$ goes as a input and fine grid simulation $\mathbf{y} \in \mathbb{R}^{m \times m}$ is our target output. We provide a 3-channel (u, v and p) coarse mesh input to predict a 3-channel (u, v and p) fine mesh output. For every input, we pool the input to 64×64 size using nearest interpolation.

When we perform a convolution with a kernel of size $k \times k$, the size of the output will decrease by $k - 1$. To maintain the same size, we usually use a kernel of odd size and pad the input symmetrically on both sides. But there are inconsistencies in the boundary conditions when we use zero padding. Padding with zeros on both sides is not consistent with the boundary conditions of velocity. We experimented with several ways to pad the tensor which are shown below

- **Zero padding:** When convolving with a kernel of size $k \times k$, we pad the output with $\frac{k-1}{2}$ zeros on both sizes.
- **Replication padding:** Instead of padding with zeros at each stage, we replace the zeros with nearest value (i.e; boundary) in the matrix.
- **Reflection padding:** We alter the input in such a way that we reflect the input over the boundary to an adequate size depending on the decrease in size over all convolutions. This method is similar to using ghost points for calculating gradients.



(a) Input u (b) Reflected u



(a) Input u (b) Reflected u

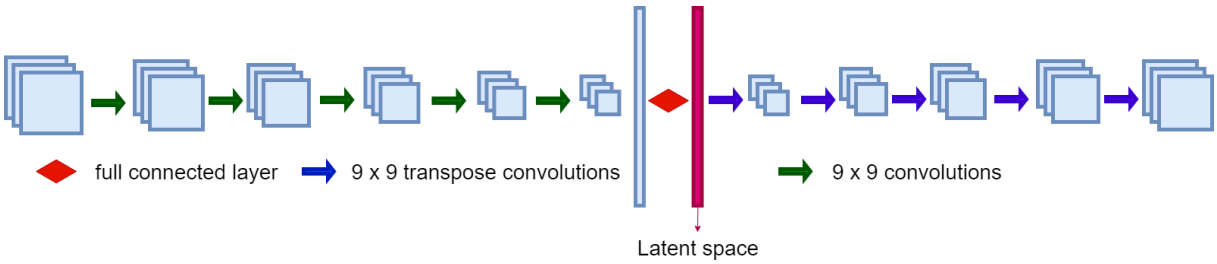
We train the DSC-MS CNN with training and validation sets. Validation set ensures that we don't over-fit the data. The model is trained till validation and training losses converge.

Encoder-decoder Convolutional neural network

Architecture

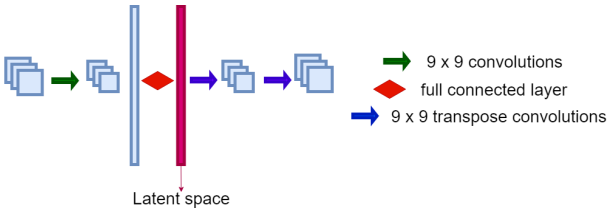
We propose a novel encoder-decoder convolutional neural network architecture to improve the results. Representations form a very important component in machine learning algorithms. We map the raw input to a latent space using an encoder network and this feature representation in latent space to output using a decoder network. In this way, we generate a feature representation for each simulation. To achieve a fine mesh simulation from coarse mesh simulation to another, we train a fully connected layer from coarse mesh latent space to fine mesh latent space and up-sample it through the decoder of fine mesh encoder-decoder CNN.

Generating feature representation for 64×64 mesh simulation



The architecture has a series of convolutional layers followed by a series of transposed convolutional layers. We reduce the input size from 64×64 to 24×24 and flatten it to get a latent vector representing the simulation.

Generating feature representation for 32×32 mesh simulation



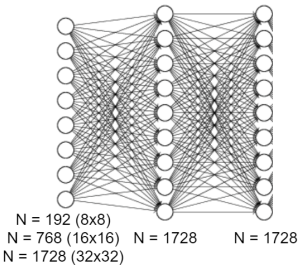
The architecture has a convolutional layer followed by a transposed convolutional layer. We reduce the input size from 32×32 to 24×24 and flatten it to get a latent vector representing the simulation.

Generating feature representations for 16×16 and 8×8 mesh simulations

As the mesh size is small, we flatten the entire matrix to a 1-dimensional vector representing the latent space.

Methodology

To generate a feature representation for a simulation, we train the encoder-decoder network with output same as the input for each mesh size. This produces a latent space which contains the features of the input simulation. The reconstruction process from a coarse grid simulation to fine grid simulation involves training a fully connected neural network from coarse grid latent space to fine grid latent space and passing it through the decoder of fine mesh encoder-decoder CNN.



The output from this fully connected network is passed through decoder of fine mesh encoder-decoder CNN to reconstruct the fine mesh simulation.

Mesh size	Latent space size	z-dim
8×8	$3 \times 8 \times 8$	192
16×16	$3 \times 16 \times 16$	768
32×32	$3 \times 24 \times 24$	1728
64×64	$3 \times 24 \times 24$	1728

4 Results

Reconstruction losses

We calculate the mean root squared loss for the output for each experiment with respect to actual fine mesh simulation as the metric for comparing various methods. We also provide a comparison with traditional interpolation methods.

$$L = \frac{1}{N}\sum_{i=1}^{n}\sum_{j=1}^{n}(\hat{y_{i,j}} - y_{i,j}) \tag{6}$$

where N is the number of training samples
$n \times n$ is the size of fine mesh
$\hat{y}$ is the output and y is the actual fine mesh simulation

The following tables show the reconstruction errors for *u* x-component and *v* y-component of velocities.

Input	8x8	16x16	32x32
Nearest interpolation	4.1834	2.4804	1.1025
Linear interpolation	4.1672	2.4382	1.0449
Cubic interpolation	4.0604	2.3596	0.9796
Multi-scale CNN (Zero padding)	1.2753	0.6111	0.2624
Multi-scale CNN (Replication padding)	1.3289	0.8062	0.6576
Multi-scale CNN (Reflection padding)	0.7181	0.5949	0.4992
Encoder-decoder CNN	0.4489	0.4298	0.2696

Table 1: Reconstruction error for *u* x-component of velocity

Input	8x8	16x16	32x32
Nearest interpolation	3.9913	2.3096	0.8778
Bi-linear interpolation	3.9841	2.2808	0.8250
Cubic interpolation	3.8953	2.2153	0.7716
Multi-scale CNN (Zero padding)	1.6588	0.5207	0.3045
Multi-scale CNN (Replication padding)	1.0306	0.5455	0.4354
Multi-scale CNN (Reflection padding)	0.8122	0.7198	0.2362
Encoder-decoder CNN	0.4054	0.3973	0.3045

Table 2: Reconstruction error for *v* y-component of velocity

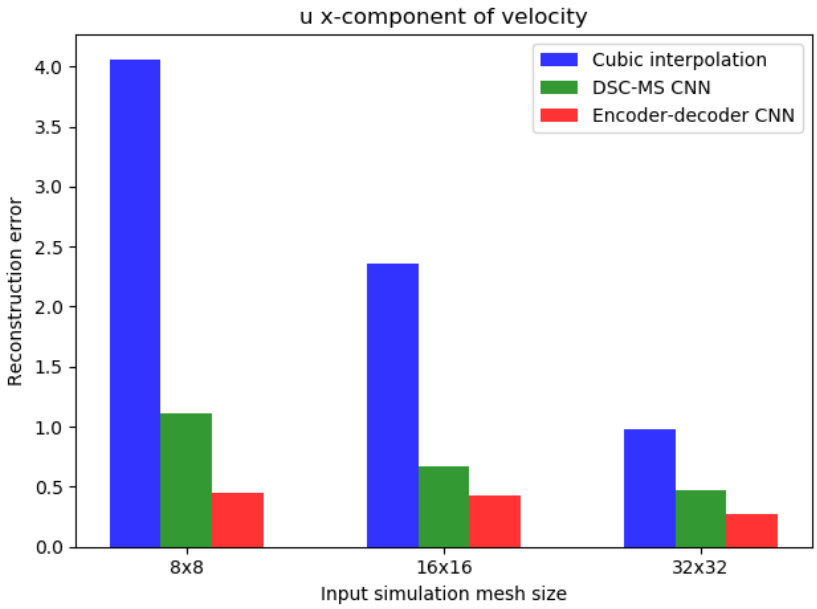


Figure 3: Comparison of 64x64 mesh reconstruction error of u x-component of velocity using various methods from input simulation of different sizes

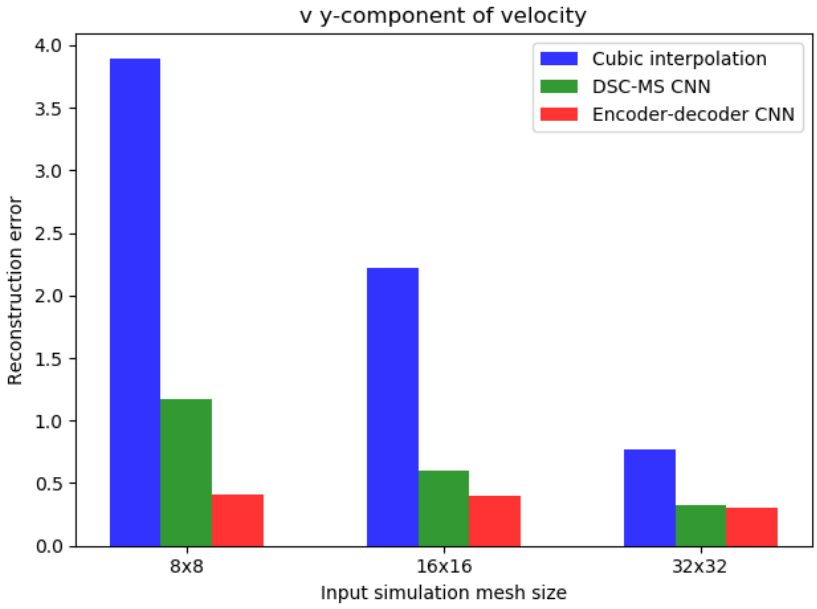


Figure 4: Comparison of 64x64 mesh reconstruction error of v y-component of velocity using various methods from input simulation of different sizes

The reconstruction error using encoder-decoder CNN is much lesser than other methods. The difference becomes much clearer as input becomes more coarser.

Visualization

The following tables show the results of u and v at *lid velocity = 0.5m/s*

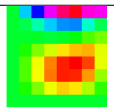
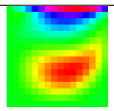
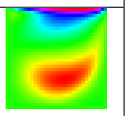
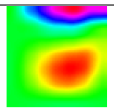
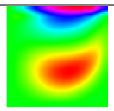
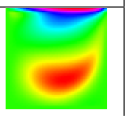
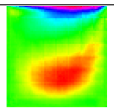
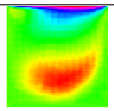
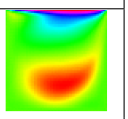
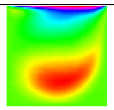
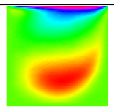
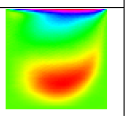
Method	8 x 8	16 x 16	32 x 32
Input			
Reconstruction error	4.0644	2.4236	1.0266
Cubic interpolation			
Reconstruction error	3.9348	2.3003	0.8929
DSC-MS CNN			
Reconstruction error	1.5005	0.5193	0.2537
Encoder-decoder CNN			
Reconstruction error	0.4924	0.2306	0.2212

Table 3: u x-component at lid velocity 0.5 m/s

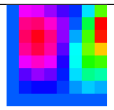
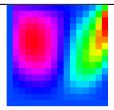
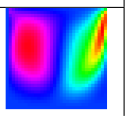
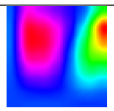
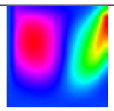
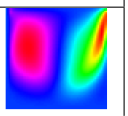
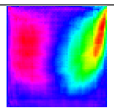
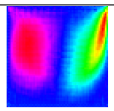
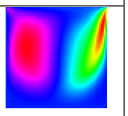
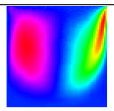
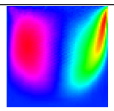
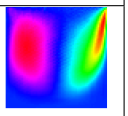
Method	8 x 8	16 x 16	32 x 32
Input			
Reconstruction error	3.8116	2.1482	0.7865
Cubic interpolation			
Reconstruction error	3.7110	2.0481	0.6712
DSC-MS CNN			
Reconstruction error	1.4429	0.4250	0.3054
Encoder-decoder CNN			
Reconstruction error	0.4910	0.2183	0.2308

Table 4: v y-component of velocity at lid velocity 0.5 m/s

Drag force

While, there isn't much difference in loss among different types of padding used in multi-scale model, the difference lies in drag force estimates. The drag force estimates using reflection padding were the most close to actual ones.

$$F = \eta A \frac{dv}{dy} = \sum_{i=1}^{n} \eta \Delta x \frac{\Delta u}{\Delta y} \tag{7}$$

The following graph of drag force on the lid for various experiments with varying lid velocity/ Reynolds number.

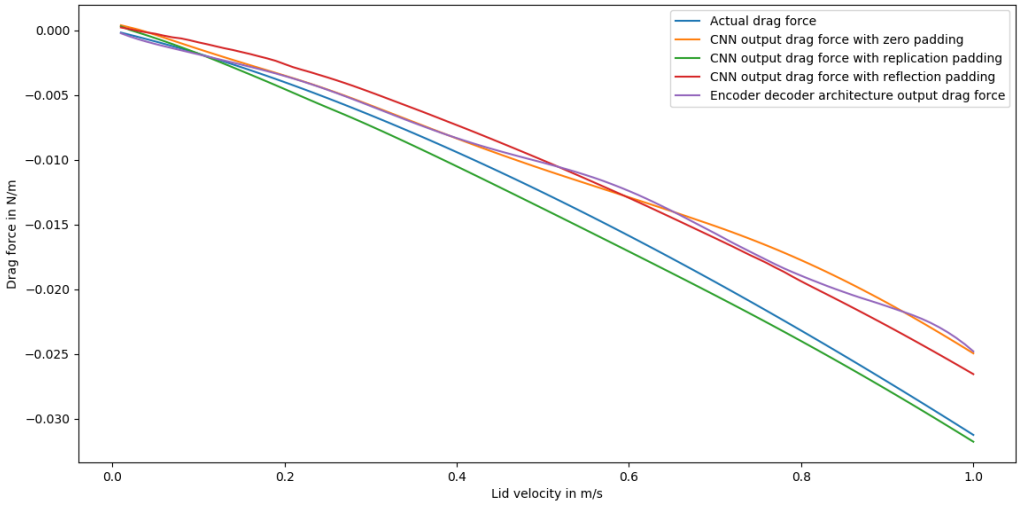


Figure 5: Drag force on lid v/s Reynolds number

Computation time

We give a comparison of time required to compute the fine mesh flow and reconstructing a fine mesh flow when run on a CPU.

DSC-MS CNN

Method	8x8	16x16	32x32	64x64
DNS simulation	12.34s	23.14s	54.28s	112.38s
Iterations	200 it	450 it	900 it	2250 it
DS-MSC CNN (forward pass)	0.1228s	0.1228s	0.1228s	-
Reconstruction time	12.46s	23.27s	54.41s	-
Speed up	9.02x	4.83x	2.06x	-

Table 5: Reconstruction time using down-sampled skip connection multi scale model

Encoder-decoder CNN

Method	8x8	16x16	32x32	64x64
DNS simulation	12.34s	23.14s	54.28s	112.38s
Encoding to get a feature representation	0.00s	0.00s	0.0047s	0.0244s
Fully connected network (forward pass)	0.0037s	0.0052s	0.0058	-
Decoding to get back output	0.0264s	0.0264s	0.0264s	-
Reconstruction time	12.3945s	23.1960	54.3413	-
Speed up	9.06x	4.84x	2.07x	-

Table 6: Reconstruction time using encoder-decoder model

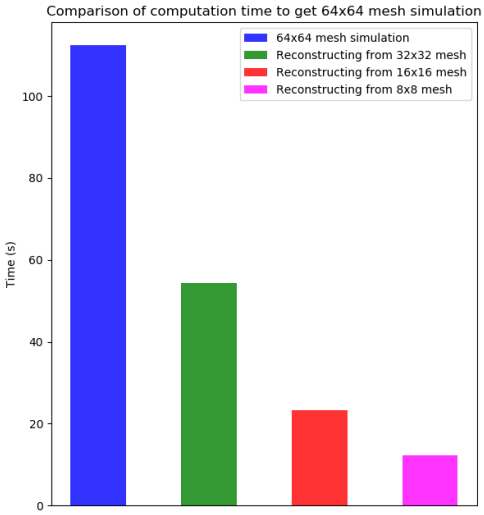


Figure 6: Comparison of computation times for various methods

5 Conclusion

We considered two machine learning models to perform super-resolution reconstruction from coarse flow fields. A Down-sample skip connection multi scale model inspired from *Super-resolution reconstruction of turbulent flows with machine learning* was first studied and used to reconstruct lid driven cavity flows over a wide range of Reynolds numbers. We developed a novel encoder-decoder CNN which produces a feature representation of flows. Reconstruction using feature representation yielded much better results compared to DSC-MS model. We also experimented with various types of padding in convolutions to address the inconsistencies in zero padding. Numerical calculation of drag force was used to verify the boundary conditions. Numerical drag force calculated with reflection padding was found to be close to simulated drag force. Out study is an attempt to show that convolutional neural networks can model the non-linear function existing between coarse and fine mesh flow fields and motivate the development of more generic and flexible models for reconstruction of a variety of flows.

6 Codes

We used PyTorch (an open source machine learning library for python) for developing the machine learning models. The code for all experiments can be found at `https://github.com/harsha070/Reconstruction-of-Flows`.

References

[1] Karthik Duraisamy, Gianluca Iaccarino and Heng Xiao. *Turbulence Modeling in the Age of Data*. Annual Review of Fluid Mechanics 2019, 51:1–23.

[2] Kai Fukami1, Koji Fukagata and Kunihiko Taira. *Super-resolution reconstruction of turbulent flows with machine learning*. arXiv:1811.11328.

[3] Yonggyun Yu, Taeil Hur, Jaeho Jung, and In Gwun Jang. *Deep learning for determining a near-optimal topological design without any iteration*. arXiv:1801.05463.

[4] Carlos Henrique Marchi; Roberta Suero and Luciano Kiyoshi Araki. *The Lid-Driven Square Cavity Flow: Numerical Solution with a 1024 x 1024 Grid*. J. Braz. Soc. Mech. Sci. and Eng. 2009, vol.31, n.3, pp.186-198. ISSN 1678-5878.

[5] Xiaoyi Jia, Xiangmin Xu, Bolun Cai and Kailing Gu. *Single Image Super-Resolution Using Multi-Scale Convolutional Neural Network*. arXiv:1705.05084.