Application of Deep Learning to Computational Fluid Dynamics

Author: Jie Tao

Supervisor: Dr. Greg Wolffe

A Project submitted in partial fulfillment of the requirements for the degree of

Master of Science in

Computer Information Systems

At

Grand Valley State University

April 2019

Greg Wolffe, Professor April 2019

Table of Contents

Table of Figures

Abstract

Computational fluid dynamics (CFD) is a common method with numerical analysis to solve problems involving fluid flows (liquids or gases). It is wildly used in research and engineering fields, including aerodynamics, weather simulation and environmental engineering.

Although the rapid rise of computational power since the utility of supercomputer and cloud computing, CFD is still a time-costing task with complexity and accuracy. In this project, we tried to apply deep learning to the filed of CFD to find the pattern in the specific case to improve the speed of simulation and lower down the complexity as well as keep the accuracy.

1. Introduction

1.1 Background

Computational Fluid Dynamics (CFD) is used to simulate fluid flow quantitively. It specifies a mathematical model of reality with the given assumptions to solve the problems in the real world which is hard or expensive to repeat in the lab. Figure 1 shows a comparison between CFD simulation and the case about weather.

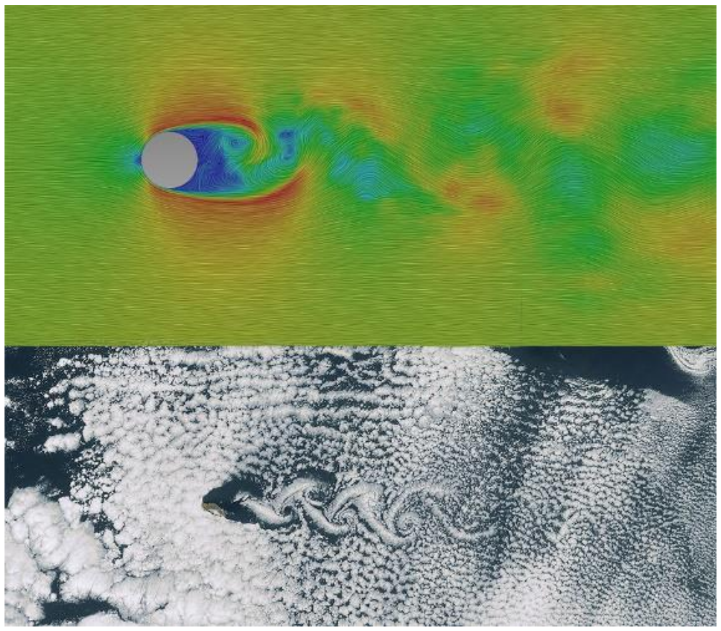


Figure 1: Examples of Karman vortex streets; numerical result (top) and real-life example of clouds^12 (bottom)

Since the complexity of physical or biological system, CFD significantly depends on the computational power and the equation. Although CFD is very mature in the industry, we are thinking: can we use deep learning model to make prediction in CFD? What's the result of deep learning simulation compared with CFD? With these two hypothesizes, we created a simple case and adapted it to the deep learning model.

1.3 Structure of this paper

The structure of this paper is as follows: the next section presents the project approach including project propose and basic tools in CFD and deep learning. We then present the implementation of project and result of this project. A conclusion summarizes the contribution of this project and suggests the directions in the future.

2. Project Approach

2.1 Propose of project

The propose of this project is applying deep learning to simulate fluid flow in the stable statues and compares results with CFD. There are two major steps: generate samples through Openfoam which is a popular CFD tool and create deep learning model trained by the generated samples.

2.2 CFD with OpenFOAM

OpenFOAM is the free, open source CFD software which has an extensive range of features to solve variety CFD problems, like complex fluid flows with turbulence and heat transfer. It also supports parallel computing with multi-core CPU and MPI.

In our project, we create a 2D mesh with size of 64m * 50m. To keep the mesh simple and efficient in the simulation, the mesh is split into 5 blocks with 1200~2000 cells for each. Then we put a circle (radius = 4.24m) inside as an obstacle. See Figure 2.

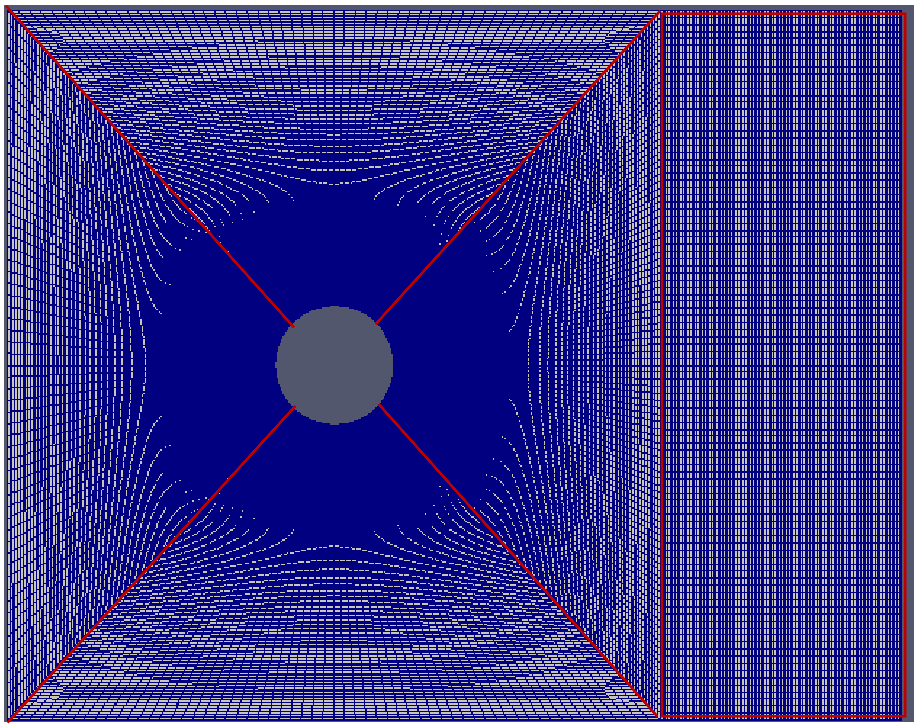
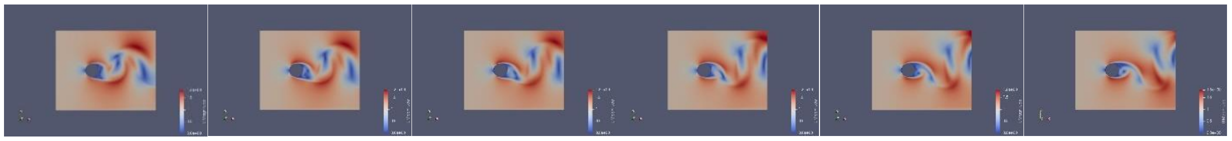


Figure 2: Mesh with 5 blocks (4 trapezoidal on the left and 1 rectangle on the right)

The Mesh has already been filled with water which comes from the left side with velocity = 1m/s and flows out to the right side horizontally. The solver (equation) we used here is **pisoFoam** which is a transient solver for incompressible flow which gives the results mostly like Figure 1. The simulation time takes 250s, we export the visualized outputs every second. After 70s, the simulation reaches a dynamic stable status, see Figure 3. So, we keep the frames from 70 to 250 frames as training samples.



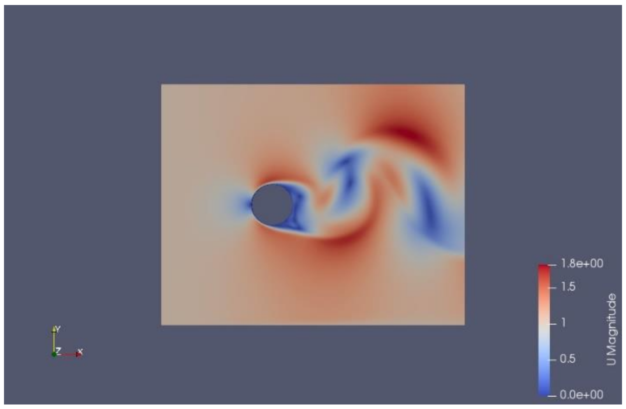


Figure 3: upper side: from left to right, each frame presents the velocity of flow in 70s, 72s, 74s, 76s,78s,80s. down side: the frame at 70s, where we can find a color bar on the right bottom shows how fast of the flow: the warmer color presents the faster velocity; the colder color presents the slower velocity.

2.3 Deep learning with ConvLSTM

Since CFD simulation is based on time series, we can use Long short-term memory (LSTM) network to predict (t+1, t+m) frames according to (t-n, t) frames where m>=1 and n>=0 (Figure 4). LSTM is an artificial recurrent neural network (RNN) introduced by Hochreiter & Schmidhuber (1997), it explicitly designed to avoid the long-term dependency problem in RNN and works well in classifying and predictions based on time series data.

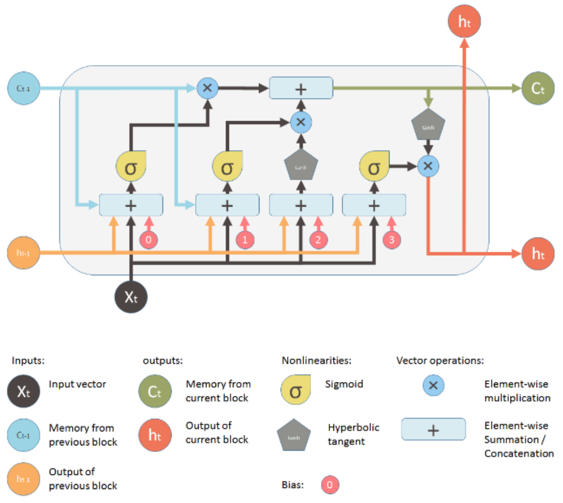


Figure 4: the structure of LSTM. From left to right, we can see forget gate, input gate, input modulation gate and output gate.

In our case, however, the inputs are not only including time series information, but also spatial information. This leads us hard to use LSTM directly. But we can turn picture into matrix. For each frame, it equals to a matrix with size of [m, n, c] where m is the height, n is the width, c is the channel number; each element in this matrix presents a pixel in the correspond picture with value from 0-255. Then the next question is how to use this matrix as input for LSTM? Here we utilized a variant of LSTM called **Convolutional LSTM** (ConvLSTM). It is first introduced in the paper **Convolutional LSTM Network:**

A Machine Learning Approach for Precipitation Nowcasting written by Xingjian Shi, Zhourong Chen, Hao Wang, Dit-Yan Yeung, Wai-kin Wong, Wang-chun Woo. Compared with traditional LSTM, ConvLSTM allow multidimensional data coming with convolutional operations in each gate. See Figure 5.

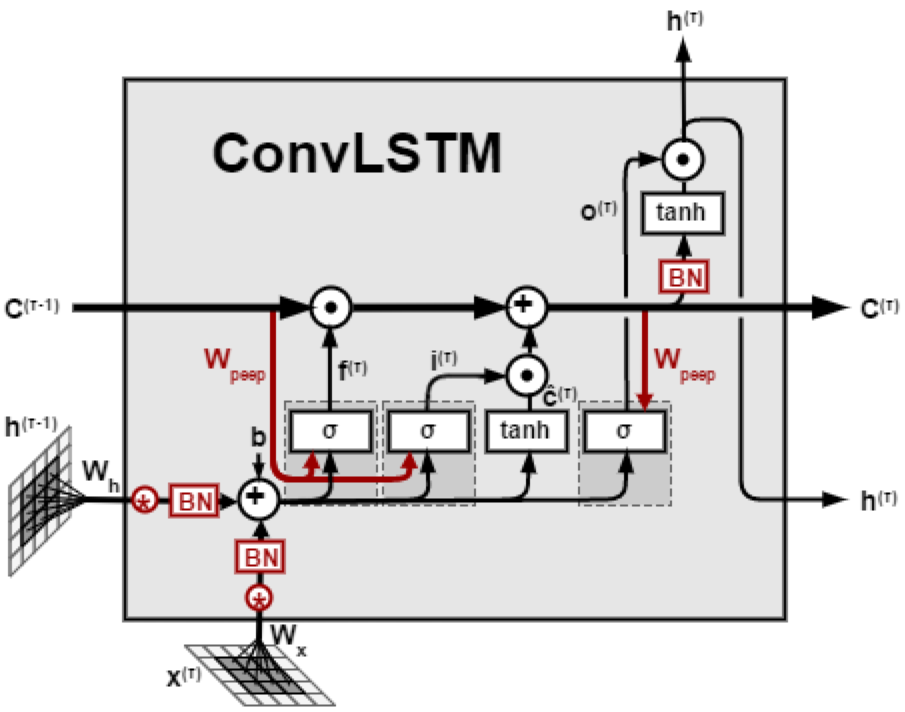


Figure 5: ConvLSTM replaces matrix multiplication with convolution operation at each gate, which leads it can capture spatial features.

In the next section, we will implement a framework called **PredNet** which based on ConvLSTM. And it is used for video predictions in car driving.

3. Implementation

3.1 Model design

PredNet is a predictive neural network for video prediction and unsupervised learning (Figure 6). It is introduced by William *Lotter, Gabriel Kreiman, David Cox* in their paper **Deep Predictive Coding Networks for Video Prediction and Unsupervised Learning.** The authors successfully use it to do the next frame prediction in car driving. In this structure, R_l is a ConvLSTM which gets the inputs from R_{l+1} in the higher layer, E_l^{t-1} and R_l^{t-1} in the previous timestep; $\hat{A}_l$ is a prediction through convolutional and Relu operations to the output of R_l; $A_l$is an input convolutional layer, for $l = 0$, the input is the target image, for $l > 0$, the input is the error E_{l-1} from the previous layer; E_l is the L1 error layer between prediction $\hat{A}_l$ and target A_l. Essentially, it is encoder-decoder structure with recursive ConvLSTM. We can refer to R_l and $\hat{A}_l$ as a decoder, A_l as an encoder. At timestep = 0, the decoder(generator) cannot predict well since no information inputs. As timestep > 0, the decoder can predict $\hat{A}^t$ by the previous inputs A^{t-1} and its error E^{t-1} while R is response for handling spatial and temporal information. The only problem is we can only do t+1 prediction at this time. We will talk about t+n prediction in the next section.

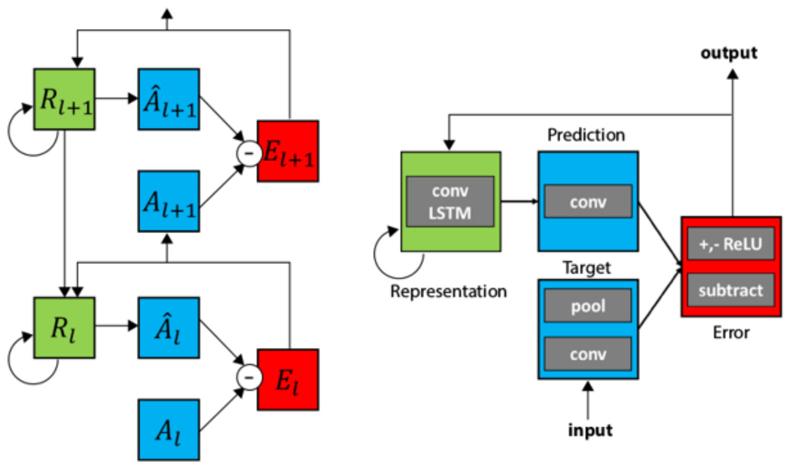


Figure 6 the structure of PreNet. Left: information flow within two layers. Right: Module operation in each layer.

PredNet also provides parameters to configure the model, like convolutional filter size, layer size and channels for each layer, etc. Table 1 shows our configuration in the project.

Parameter	Configure	Desc
Input	(11, 3, 128, 160)	11 frames group to a sample, each frame with size of 160*128 pixels and 3 channels.
Stack size	(3, 48, 96, 192)	There are 4 layers that takes in RGB images and has 48, 96 and 192 channels in the 2nd, 3rd and 4th layers respectively.
A filter size	(3, 3, 3)	The targets for layers 2 and 3 are computed by a 3*3 convolution filter.
Ahat filter size	(3, 3, 3, 3)	The targets for layers 1, 2 and 3 are computed by a 3*3 convolution filter.

Output	• {1} for t+1 training • {11,3,128,160} for prediction and t+n training	For t+1 training, the module will output error, our goal is reduced error to 0. For t+n or prediction stage, the output is prediction image.
Loss function	MAE	Mean absolute error
Optimizer	adam	
Validate ratio	5%	5% of total samples will be used as validation
Total Params	6,915,948	

Table 1: the configuration for PredNet

3.2 Training design

With help of high-performance computing (HPC), we can try more complex models and train it much faster than before. Table 2 shows the hardware and software we use in the training stage.

Item	Specs
HPC Name:	Ghost
Processor	AMD Ryzen ThreadRipper 16-core liquid-cooled CPU
Memory	64GB DDR4 3200MHz Corsair RAM
Hard Drive	1 TB Samsung SSD; 2 TB Seagate HDD
GPU	NVIDIA Titan V w/51`20 cores (640 TensorCores), 12 GB GDDR5
OS	Ubuntu 18.04 LTS
Software	Keras (TensorFlow GPU), Python 3.6.7

Table 2: with the modern GPU's acceleration, we can utilize more complex model and optimate the hyper-parameters easier.

Before starting training, we must do some preprocessing to each frame to reduce unusable information. Firstly, we crop the frame and keep the mesh area only. Then resize it to 160*128 pixel. The small size of sample makes training faster than larger one.

Each inputs sample consists of 11 continuous frames order by time series (the start frame is randomly pick up), let's denote this list as [t, t+11) where t starts with 0. Firstly, we trained t+1 module denoted as **PredNet(t+1)**, our input is [t, t+11), output is sum of error from PredNet which closes to 0 in the training. Second, we construct a new PredNet denoted as **PredNet(t+10)**, which uses same weights as PredNet(t+1) (Copy from trained PredNet(t+1)). The input is [t, t+20), and output is predication [t, t+20]. The difference is we let the pervious prediction as input after t=10, which means [t+10, t+20) is extrapolation. Our goal is making cost between module prediction and ground truth reducing in the training. See Figure 7.

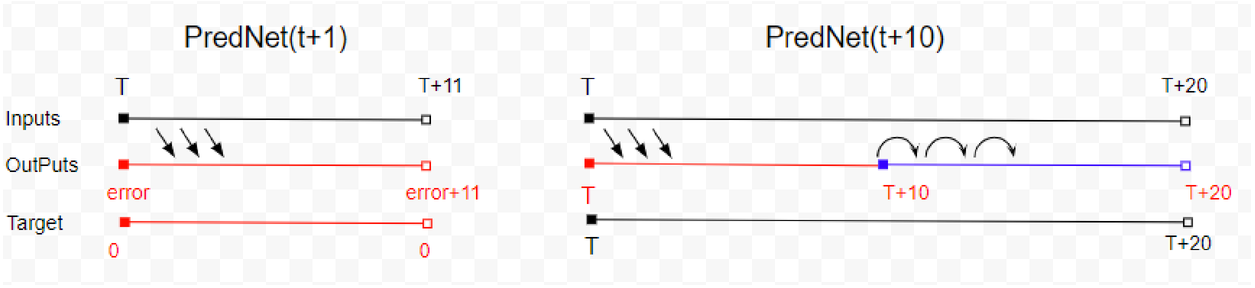


Figure 7: Two training methods. Left: PredNet(t+1), Right: PredNet(t+10). PredNet can be set to output prediction $\hat{A}$ or sum of errors $\sum_{l=0}^{L} E_l$

Our batch size sets as 10, 100 samples for each epoch, and learning rate sets as 0.001 when epoch < 75, otherwise 0.0001. After 100 epoch, loss values of both training methods are lower than 0.002. We saved the models to h5 files for prediction and validation tasks later. As we can see in Figure 8, For PredNet(t+1), after 22 epochs, the loss value cannot reduce any more, the model couldn't be improved by adding more samples. Only 23 mins, we finished a model training with almost 7 million parameters on Ghost. For PredNet(t+10), it seems need more epochs to train the model, after 80 epochs, the loss value cannot reduce any more. It took 100 mins to complete the training process.

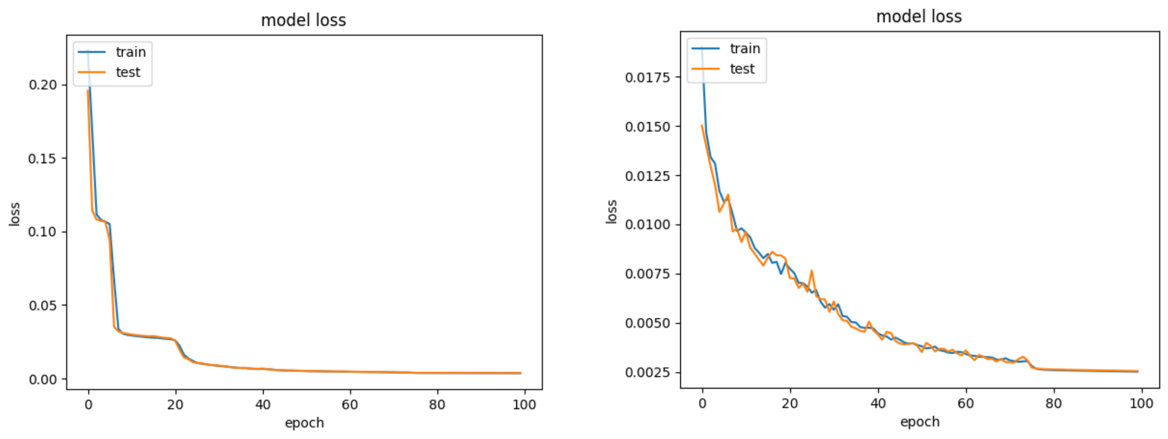


Figure 8: left: PredNet(t+1) loss plot, right: Prednet(t+10) loss plot

3.3 Prediction

We created three prediction manners in term of the two trained models we mentioned previously, see Figure 9. The first manner uses PredNet(t+1) to do t+1 prediction. The input is 11 frames [t, t+11), the output is 10 frames [t+1, t+11). The second manner also use PredNet(t+1) but facilitated the rolling prediction, which means we appended the last frame of predicted output with the original input, then use the new input to do iterative predictions until all frame of input are from prediction. The third manner is as same as the second one but changed the model to PredNet(t+10).

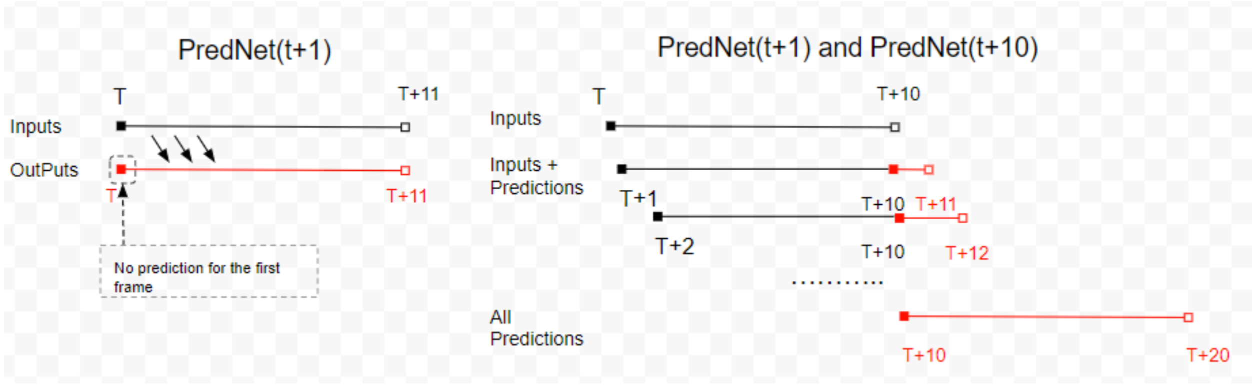


Figure 9: three prediction manners. Left: PredNet(t+1) with t+1 prediction. Right: Prednet(t+1) and PredNet(t+10) with t+10 rolling prediction.

In Figure 10Figure 10 and Figure 11, we visualized the predicted frames to compare with the ground truth so that we can see the results intuitively.

Prediction(t+1) with t+1 prediction:

Prediction(t+1) with t+10 prediction:

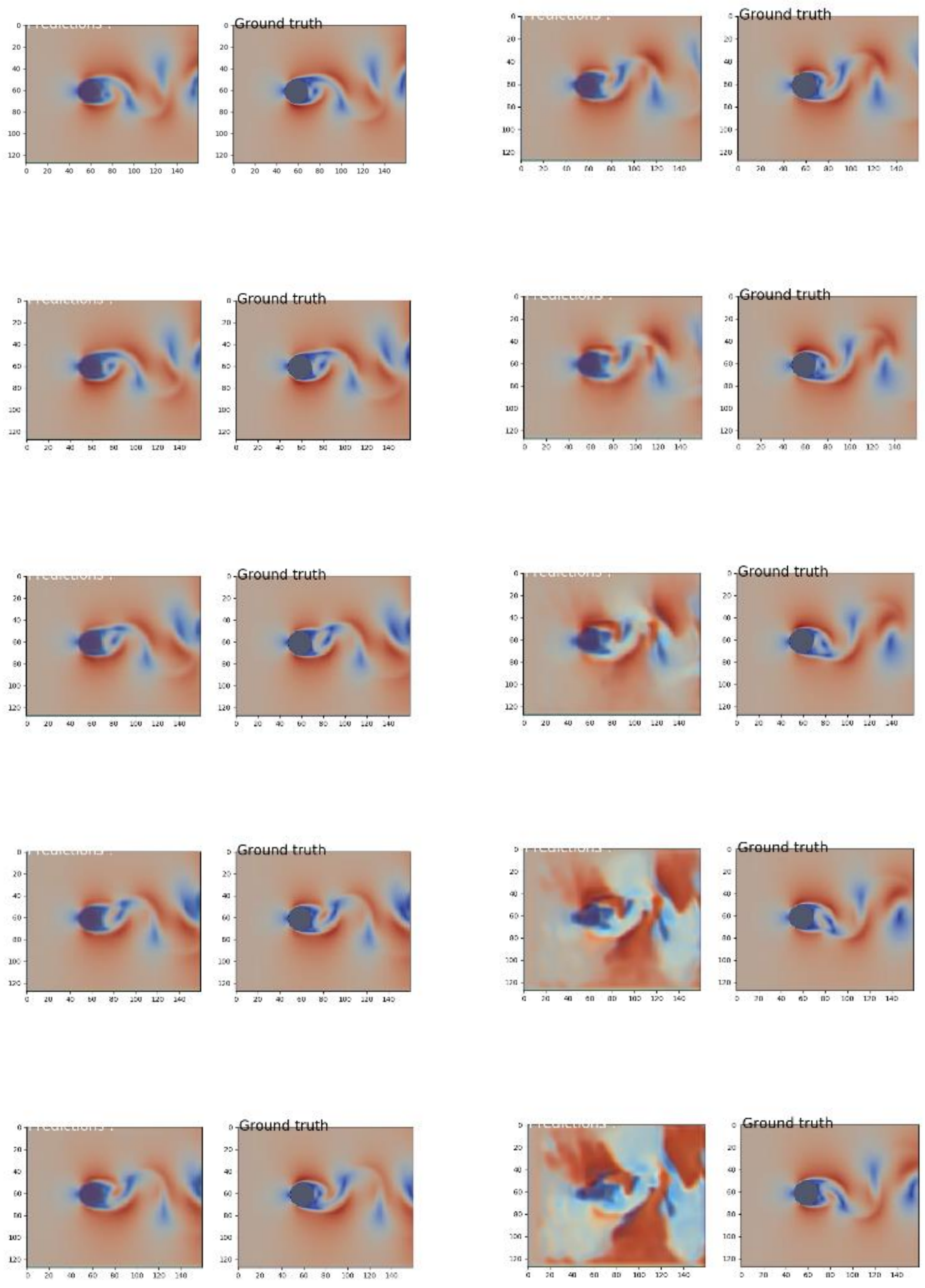


Figure 10: the comparison between prediction and ground truth in PredNet(n+1). We can find PredNet(n+1) didn't work well in the t+10 rolling prediction.

Prediction(t+1) with t+10 prediction:

Prediction(t+10) with t+10 prediction:

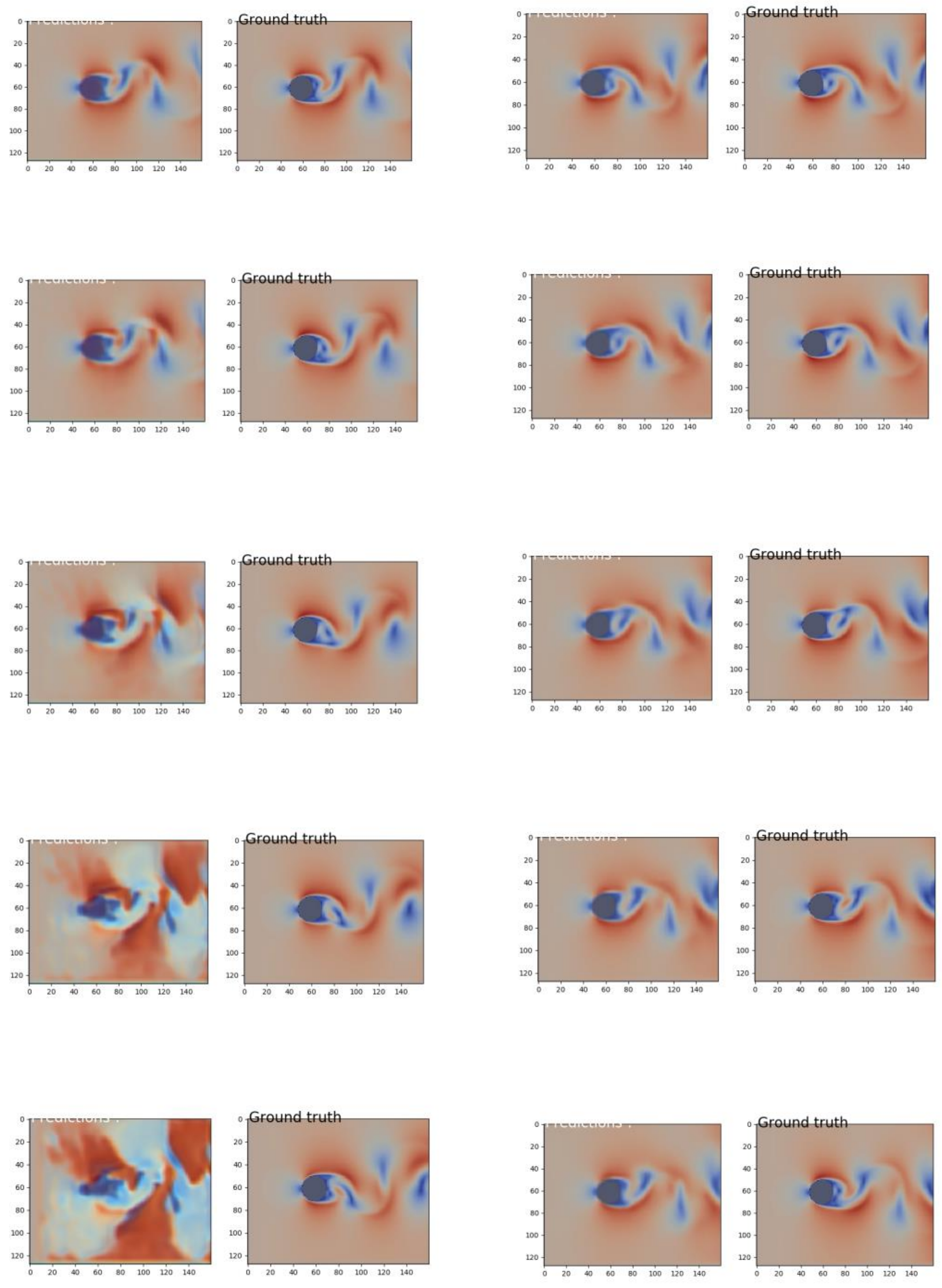


Figure 11: the comparison between prediction and ground truth in PredNet(t+1) and PredNet(t+10). We can find PredNet(t+10) is doing much better in t+10 rolling prediction even they are both inputted with 10 frames. The extrapolation training helps the performance of PredNet(t+10).

3.4 Validation

To validation the result and reduce the deviation, we randomly pick up 100 samples, then calculate the average of PSNR (RGB: 0-255, 3 channels):

$$PSNR = 20log_{10}\left(\frac{Max(f)}{\sqrt{MSE}}\right)$$

$$\text{Where } MSE = \frac{1}{120*160*3*100}\sum_{i=1}^{100}\sum_{j=1}^{10}\left(F_j - G_j\right)^2,$$

where F_J is predicted frame, G_J is corresponding ground truth frame, MAX(f) is the max value of original frame, here is 255. The greater value of PSNR, the better prediction.

And difference of sum of all pixels

$$\text{SumCheck} = \frac{1}{100}\sum_{\text{i}=1}^{100}\left(\left|\left|\Sigma(\text{F})_i - \Sigma(\text{G})_i\right|\right|\right),$$

where $\sum(F)_i$ is sum of all pixel values in the predicted frame i, $\sum(G)_i$ is corresponding sum of all pixel values in the ground truth i.

Before validation, we need compare the different between t and t+10 in the ground truth, which proves the model works rather than the frames similar with each other. See Figure 12.

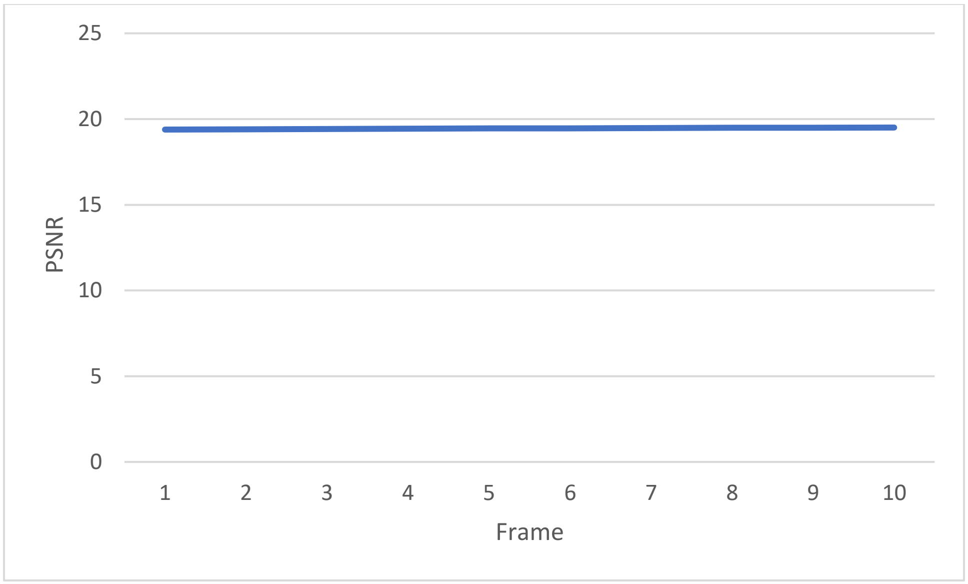
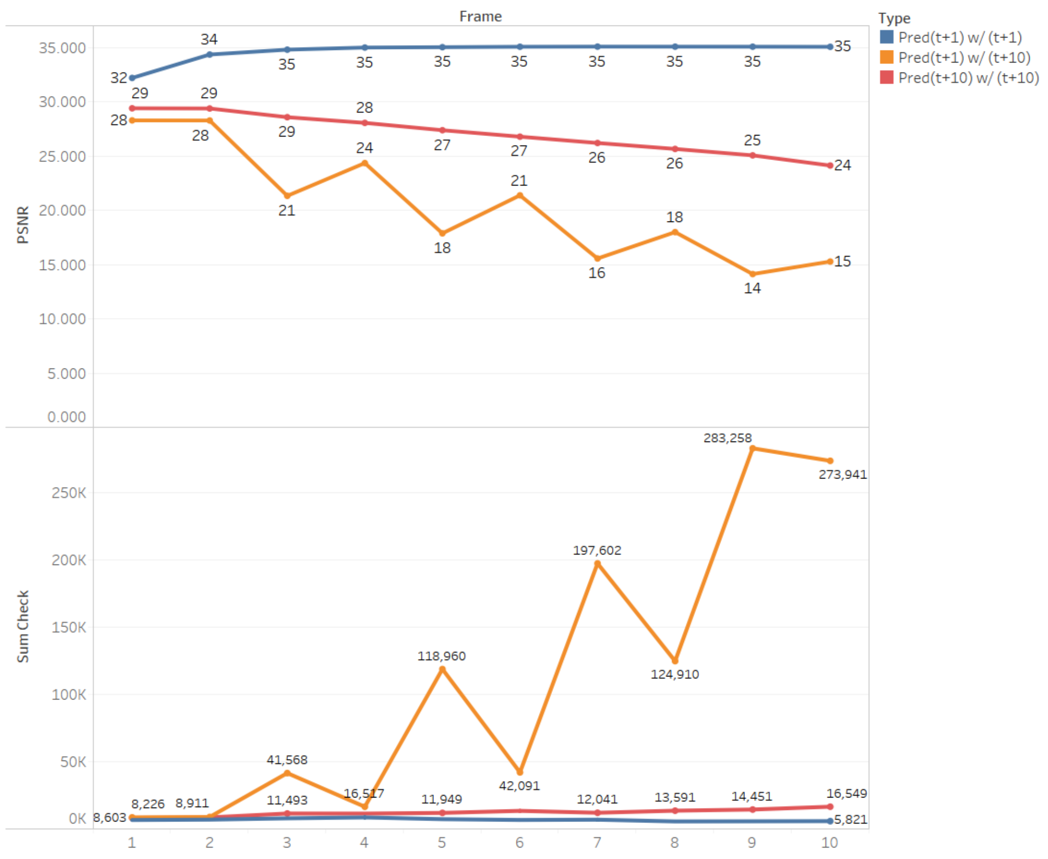


Figure 12: PSNR value is almost fixed on 19, which means the degree of difference between frames is same.



Type	Frame	MSE	Sum Check
Pred(t+1) w/ (t+1)	1	32.204	6,650
	2	34.361	6,951
	3	34.810	7,910
	4	35.006	8,590
	5	35.040	7,273
	6	35.079	6,686
	7	35.091	6,896
	8	35.087	5,604
	9	35.084	5,698
	10	35.073	5,821
Pred(t+1) w/ (t+10)	1	28.291	8,603
	2	28.282	8,911
	3	21.337	41,568
	4	24.358	16,517
	5	17.880	118,960
	6	21.383	42,091
	7	15.562	197,602
	8	17.994	124,910
	9	14.127	283,258
	10	15.282	273,941
Pred(t+10) w/ (t+10)	1	29.406	8,226
	2	29.389	8,588
	3	28.586	11,493
	4	28.061	11,438
	5	27.386	11,949
	6	26.791	13,463
	7	26.207	12,041
	8	25.660	13,591
	9	25.067	14,451
	10	24.131	16,549

Sum Check
5,604 283,258

MSE
14.127 35.091

Figure 13: quantitatively comparison results within three manners based on 100 samples

Through Figure 13, we can easily find out that PredNet(t+1) gives a good performance in t+1 prediction, but rapidly degradation in t+10 prediction. While PredNet(t+10) is relatively better in t+10 prediction. The extrapolation training helps PredNet(t+10) to keep the robust in the rolling prediction.

4. Conclusion

The result of this project supports that deep learning can be applied in the sample case of CFD field. If we control the environment of fluid flow, it can reach a static or dynamic stable status. Then we can use deep learning model to learn the pattern of this stable statues. However, in this project we only applied deep learning model to one strictly controlled scenario and it can only stimulate 10 seconds. In the future, we are planning to create more variety samples in the different velocity, mesh and features of fluid. So that we can figure out if this deep learning model has ability to be used in general situation.

References

n.d. *About OpenFOAM.* https://www.openfoam.com.

Gregory Wolffe, John Oleszkiewicz,David Cherba, Dewei Qi. 2002. "Parallelizing Solid Particles in Lattice-Boltzmann Fluid Dynamics." *Proceedings of PDPTA '02.*

Olah, Christopher. 2015. *Understanding LSTM Networks.* August 27. http://colah.github.io/posts/2015-08-Understanding-LSTMs/.

Shuo Sun, Gaoxiao Liu. 2018. *Air Quality Forcasting Using Convlutional LSTM.* stanford.

n.d. *Standard solvers.* https://www.openfoam.com/documentation/user-guide/standard-solvers.php.

n.d. *What is CFD | Computational Fluid Dynamics.* https://www.simscale.com/docs/content/simwiki/cfd/whatiscfd.html.

William Lotter, Gabriel Kreiman, David Cox. 2016. "Deep Predictive Coding Networks for Video Prediction and Unsupervised Learning." *arxiv.org.*

Xingjian Shi, Zhourong Chen, Hao Wang, Dit-Yan Yeung, Wai-kin Wong, Wang-chun Woo. 2015. "Convolutional LSTM Network: A Machine Learning Approach for Precipitation Nowcasting." *arxiv.org.*