DEVELOPMENT OF A TOOL BASED ON MACHINE LEARNING FOR IDENTIFYING FLOW STRUCTURES

DISSERTATION REPORT

SUBMITTED IN PARTIAL FULFILLMENT OF THE REQUIREMENTS FOR THE AWARD OF THE DEGREE OF

Master of Technology

In

Mechanical Engineering

(THERMAL SCIENCE)

SUBMITTED BY

ANIRUDH PRATAP SINGH

GI9444

19METM714

UNDER THE SUPERVISION OF

PROF. SYED FAHAD ANWER

DEPARTMENT OF MECHANICAL ENGINEERING
ZAKIR HUSAIN COLLEGE OF ENGINEERING &TECHNOLOGY
ALIGARH MUSLIM UNIVERSITY
ALIGARH, UP, (INDIA)

2021

CERTIFICATE

CANDIDATE'S DECLARATION

I hereby declare that this dissertation, titled "**Development of a Tool Based on Machine Learning for Identifying Flow Structures**" is being submitted in partial fulfillment of the requirement for the award of the degree of **Master of Technology** in **Mechanical Engineering** with specialization in **Thermal Engineering** in the **Department of Mechanical Engineering, Aligarh Muslim University, Aligarh,** is an authentic record of my work under the supervision of **Prof. Syed Fahad Anwer**. The matter embodied in this work has not been submitted for the award of any other degree or diploma in any institution.

Dated: **ANIRUDH PRATAP SINGH**

SUPERVISOR'S CERTIFICATE

This is to certify that the above statement made by the candidate is correct to the best of my knowledge.

Dated:

Prof. SYED FAHAD ANWER

Professor, Dept. of Mech. Engg.,

Z.H.C.E.T, Aligarh Muslim University

ACKNOWLEGDEMENT

In the name of Allah, Most Gracious, Most Merciful

I feel great pleasure in expressing deep sense of gratitude and sincere thanks to the venerable supervisor Prof. Syed Fahad Anwer, Professor, Department of Mechanical Engineering, Zakir Husain College of Engineering and Technology, AMU, Aligarh, for his illuminating and precious guidance, invaluable suggestions, constant encouragement and support throughout the course of the project work. His patience, vision and understanding during the numerous crises in this study were invaluable. His conceptual skills and expertise are unmatched. It has been a privilege to gain experience , learn and acquire the values of research, time management and independent thinking, under his esteemed supervision. Working with him, has been very exciting and enriching.

I would also like to take the opportunity to thank all those who helped make this project a success, especially Prof. Nadeem Hasan, Mr. Athar for the invaluable guidance and support.

I would also like to take this opportunity to thank my friends Mr. Issam Wajih, Mr. Faizan Hamid, Mr. Ashish Garg and Mr. Puneet Panwar for the support. Last but not the least, I would like to thank my parents for the love and affection, the never ending support during trying times. Finally, before concluding, I remember once again *Allah* who gave me power, energy and enthusiasm to accomplish this work.

ANIRUDH PRATAP SINGH
A.M.U, ALIGARH
October 24, 2021

ABSTRACT

A very powerful tool of Artificial Intelligence and machine learning is the Neural Network deemed as a universal approximator for any function. The work presents the development of the tool specifically for the case of fluid dynamics. Fluid dynamics conventionally employed POD to extract spatial modes of some fluid flow times series data. These modes can be used to find the most important and dominant features of the flow. With these flow features or modes the flow problem can be modeled to a much simpler version which reduces the computational cost. The extraction of modes requires efficient low dimensional compression. For this the tool of neural network (NN) can be used. A NN, although deemed powerful needs much optimization as it has numerous parameters.The NN structure that deals with dimensionality reduction is the autoencoder. The deep autoencoder is preliminary analyzed using different datasets of 1-D and 2-D (including lid driven flow steady state problem). First the similarity between POD and dense autoencoder with linear activation function is drawn, proving the linear autoencoder being an iterative POD. Then, comparison is made when the autoencoder is incorporated with non-linear activation function with the proper orthogonal decomposition (POD). Along the way the dense autoencoder is optimized for the number of hidden layers and the nodes in the hidden layers. Subsequently, the inadequacy of dense autoencoder grasped when dealing with fluid flow data time series data and is sorted out with the use of conventional-convolutional neural network (C-CNN). The C-CNN being a different autoencoder structure is optimized in regards to the different activation function choices present with the use of Keras API making use of Vortex Induced Vibration (VIV) flow problem as data basis. The pre-processing of data necessary for best possible learning is found out, as a raw data cannot be freely used with the C-CNN. Along the way few observations are also made regarding the sample size that is relevant for good learning when dealing with complex flows. The C-CNN optimized thus, is modified to the MD-CNN. The MD-CNN constructed is used to extract and visualize the modes present in the latent layer of the C-CNN. Based on all the optimizations performed a string of recommendations are presented for the future efficient use of the tool of NN for fluid dynamics. The POD and NN codes are developed using MATLAB and python with keras which in-turn uses tensorflow as backend respectively. The POD uses SVD as the method for performing the mode decomposition.

CONTENTS

LIST OF FIGURES

LIST OF TABLES

NOMENCLATURE

AI Artificial Intelligence

ROM Reduced order modeling

POD Proper Orthogonal Decomposition

PCA Principal component analysis

DNS Direct numerical simulation

ML Machine Learning

NN Neural Network

ANN Artificial neural network

AE Autoencoder

C-CNN Conventional type convolutional neural network

CNN-AE Convolutional neural network autoencoder

MD-CNN Mode decomposition convolutional neural network

EPOCH refers to one cycle through the full training dataset

Re Reynolds number

Ri Richardson number

1. INTRODUCTION

1.1 Reduced Order modeling and Proper Orthogonal Decomposition

Fluid Mechanics is the field of science that deals with an fluid flow governed by an unsolvable equation, the Navier-stokes equation. The NS equation belongs to one of the seven millennium problems as it can't be proven that a smooth solution always exists for a full 3D equation. This problem of not able to find a closed form solution called for an approximate approach for finding the desired solution. The discretization of differential equation gave one such solution. The drawback with this method is that to approximate the differential equation solution a very refined mesh is required. This is called Direct numerical simulation (DNS). DNS of even a simple problem is very data intensive and as such, since the inception of DNS, fluid mechanics has generated massive bytes of data. To generate such vast amount of data a large computational power is required. To reduce this computational load reduced order modeling was thought of that reduced the number of points of computation by extracting the most important feature of fluid flow. The most important features carry the majority information about the flow and helps in discarding the redundant dormant features that carry very little information about the flow. The first method developed in lieu for this was Proper Orthogonal Decomposition (POD)[1]. This method is based on eigen value mode decomposition. The reduced features, also called the modes help construct the reduced order model by greatly reducing the number of degrees of freedom. The reduced order model is realized by mapping the problem on reduced coordinates immensely helping with the computation time.

1.2 Artificial Intelligence(AI)

Simulating Bio-chemical intelligence into another media such as electronic is known as Artificial Intelligence. Artificial Intelligence is generally attributed to machines or inanimate objects. The beginning of modern AI can be traced back to classical Philosophers. But, field of AI wasn't formally introduced till 1956. In 2016 just 60 years ahead Google's DeepMind program [1], AlphaGo beat the 18-time World Go champion Lee Sedol, a game that has $10^{7}68$ possible games in a 361-move

[1] For details on POD refer to appendix

game. This goes to show the level of critical thinking achieved by machine-based intelligence.

Today AI is at the forefront of the technology spreading to nearly all aspects of life from self-driving cars [2] to solving problems to making intricate melodies. And with the advent of fast speed internet AI is getting the resources to accelerate to even higher standards. It is estimated that today humans generate 2.5 Quintillion bytes of data each day. With so much data in hand a well-trained AI can perform almost every task with near and sometimes more accuracy as achieved by humans.

With this vast amount of untapped data great strides can be made in understanding the secrets of the Universe and at the same time help humankind.

1.3 AI and Science

Science is one such area where great advancements have been made due to AI in recent years. Fluid Mechanics being one part of this is where AI is coming up. This field generates intensive amount of data by running simulations which are to be properly interpreted and analyzed. One such area of application is Reduced Order Modelling (ROM) [3, 4] by extracting the important flow features. The reduced model greatly enhances the computation time by allowing computations to be performed on reduced coordinates. A novel method at achieving this is Neural Networks which unlike linear eigen mode decomposition can learn the non-linear features making the ROM even more efficient.

1.4 Artificial Neural network

An artificial neural network is a system of hardware or software that is patterned after the working of neurons in the human brain and nervous system. Artificial neural networks are a variety of deep learning technology which comes under the broad domain of Artificial Intelligence.

Deep learning is a branch of ML which uses different types of neural networks. These algorithms are inspired by the way our brain functions and therefore many experts believe they are our best shot to moving towards real AI (Artificial Intelligence).

There are many kinds of Neural Network structure that make use one of the three kinds of learning mode being:

1. Supervised Learning: A ML task that maps the input to a known output. E.g., Learning to recognize hand written digits from a data set of labelled data

set

2. Semi-Supervised Learning: Combination of labelled and un-labelled data which helps in increasing the learning accuracy

3. Unsupervised Learning: A ML task that draws inferences from the dataset. In some cases, the input acts as the output. E.g., Image compression makes use of same input and output and tries to compress the image into a smaller size containing the maximum details.

Work in ANN or rather AI started when in 1943 McCulloch and Pitts [5,6] made the first mathematical model of single unit of the brain called the neuron. It was very rudimentary and worked on the threshold logic.

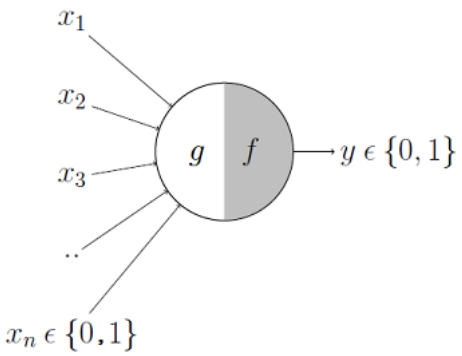


Fig. 1.1: McCulloch-Pitts Mathematical Model of Neuron

Mathematically, the McCulloch and Pitts model is surmised as follows:

$$y = f(g(X)) = \begin{cases} 1 & \text{if } g(X) \geq \theta \\ 0 & \text{if } g(X) < \theta \end{cases} \tag{1.1}$$

where,

$$g(x_1, x_2, x_3, ...x_n) = g(X) = \sum_{i=1}^{n} x_i$$

and θ is the threshold value

The figure 1.1 represents the single unit of an ANN [7]. At the time it was thought to be the universal approximator but this could not have been farther from the truth. The inadequacy of the single McCulloch and Pitts model or the single perceptron came out when the logic gate functions were being studied. It was found that the XOR and the XNOR functions could not be simulated using this model as they don't have linearly separable output(or rather they had non-linear decision boundary). Due to this work on single perceptron got delayed, but after many decades it was picked up once more to be developed into the complex

structures of ANN that we now use. The single layer perceptron is the basic building block of the ANN (shown in figure 1.2)

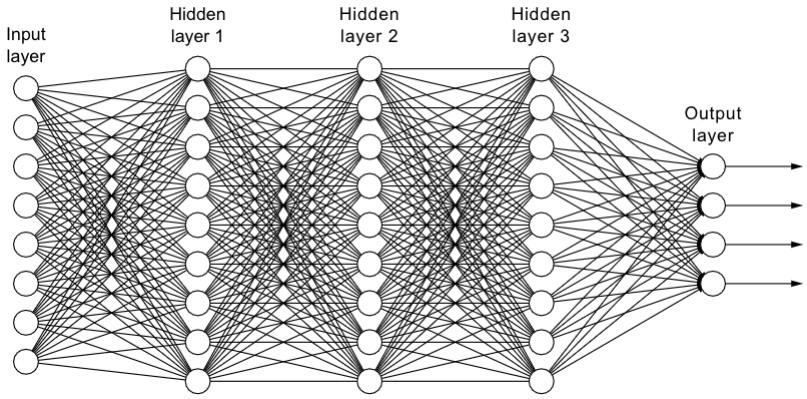


Fig. 1.2: Schematic diagram of an ANN

1.5 Autoencoder

One of the many ANN structures used throughout this thesis is the utoencoder structure. An autoencoder with the simplest structure is shown in figure 1.3

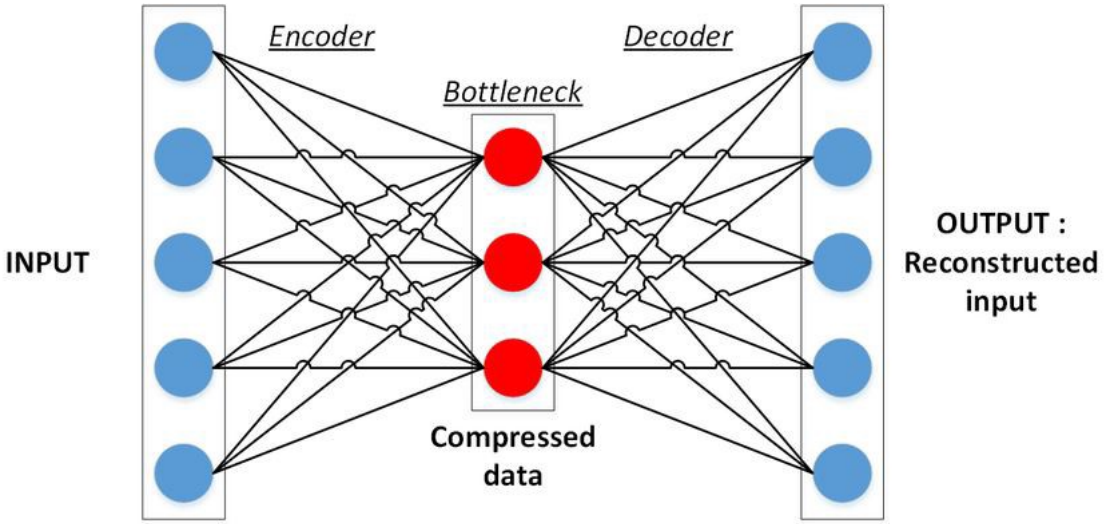


Fig. 1.3: Sketch of simplest Autoencoder with one hidden layer

Autoencoders employ unsupervised learning that leverage NN's for the task of representation learning. They map input to the output which is same as the input i.e. $f : X \to X$.
Autoencoders are widely used for following tasks:

1. Dimensionality reduction

2. Image compression
3. Image denoising
4. Feature extraction
5. Image generation
6. Sequence to sequence prediction
7. Recommendation system

Consequently there are 7 kinds of autoencoder structures available

1. Denoising autoencoder
2. Sparse autoencoder
3. dense autoencoder
4. Contractive autoencoder
5. Undercomplete autoencoder
6. Convolutional autoencoder
7. Variational autoencoder

The application of autoencoder that is most useful for the cause of engineering, specifically in Fluid mechanics is dimensionality reduction and feature extraction. It is very helpful in building a reduced order model(ROM).Reduced order models (ROMs) are simplifications of high-fidelity, complex models. They capture the behavior of these source models so that engineers can quickly study a system's dominant effects using minimal computational resources. For this the *dense* and the *Convolutional* type autoencoder structure are used. In the following sections we briefly discuss both.

1.5.1 Dense Autoencoder

A dense autoencoder is just the complex counterpart of the structure shown in figure 1.3 on page 5. The complex counterpart has many more layers and each layer has numerous perceptrons.

Suppose that X is the input of the autoencoder:

- Encoder : - a function f that compresses the input into a latent-space representation.$f(X) = h$
- Decoder : - a function g that reconstruct the input from the latent space representation.$g(h) \sim X$

The function f *and* g are known as activations[2]

The training is done based on the loss between the input and the output.

1.5.2 Convolutional Autoencoder

To emphasize, a CNN-AE(Convolutional neural network -autoencoder) is just a special class of CNN with the basic structure similar to that of a dense autoencoder. The core math behind the CNN-AE is also similar to that of dense autoencoder. The encoder part of which is shown in figure 1.4, where the decoder part is just the mirror image

The CNN-AE is far superior to the dense autoencoder for images, videos and audio signal inputs. The structure of CNN has three main layers:

- Convolutional layer
- Pooling layer
- Fully connected layer (for the autoencoder structure)

[2] Refer to appendix for details activation function and kinds used

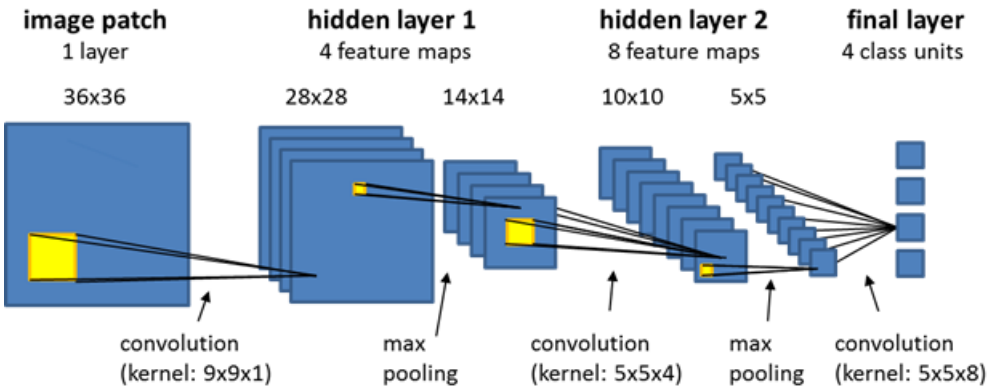


Fig. 1.4: A Basic CNN-AE Encoder

Convolutional Layer

The main layer of the CNN is the convolutional layer. A convolutional layer has filter of some size that contains weights to convolve the first layer as shown in figure 1.5

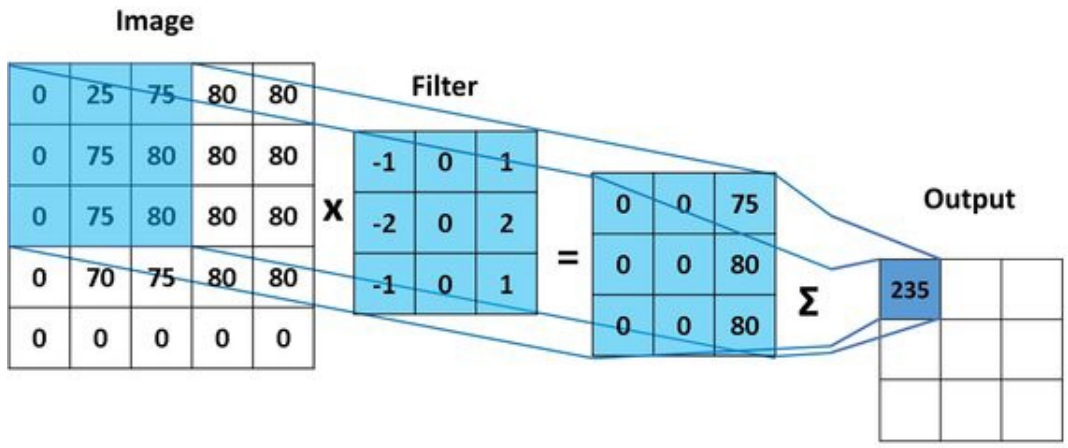


Fig. 1.5: Convolution layer of the CNN

It is weight values in kernel that adjust during training through the process of backpropagation and gradient descent. However, there are three hyper parameters which affect the volume size of the output that need to be set before the training of the neural network begins. These include:

- The **filter/kernel number** affects the depth of the output. For example, three distinct filters would yield three different feature maps, creating a depth of three.
- **Stride** is the distance, or number of pixels, that the kernel moves over the input matrix. While stride values of two or greater is rare, a larger stride yields a smaller output.
- **Padding**, is done to get the desired output after the convolution operation. Padding are of three types
    1. **Valid padding**: This is also known as no padding. In this case, the last convolution is dropped if dimensions do not align.
    2. **Same padding**: This padding ensures that the output layer has the same size as the input layer
    3. **Full padding**: This type of padding increases the size of the output by adding zeros to the border of the input. Figure 1.6 shows the valid and the same padding.

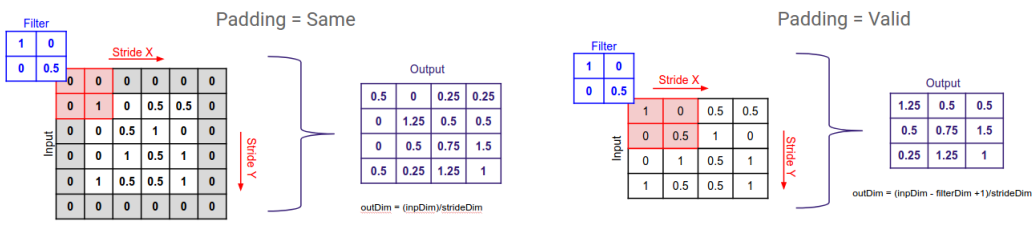
Pooling Layer

Pooling or downsampling layer does the dimensionality reduction, reducing the number of parameters in the input. It does the same process as the convolutional layer except for the fact that the filters contain no weights to be trained. It performs the aggregation or maximum of function on the convolutional filters and this populates the output.There are two main types of pooling (refer figure 1.7) :

- **Maxpooling**: As the filter moves across the input, it selects the pixel with the maximum value to send to the output array
- **Average pooling**:As the filter moves across the input, it calculates the average value within the receptive field to send to the output array.

Fully-connected Layer

This layer essentially converts a CNN to an ordinary NN (like the dense AE).



(a) Same padding *(b)* Valid padding

Fig. 1.6: Padding in CNN

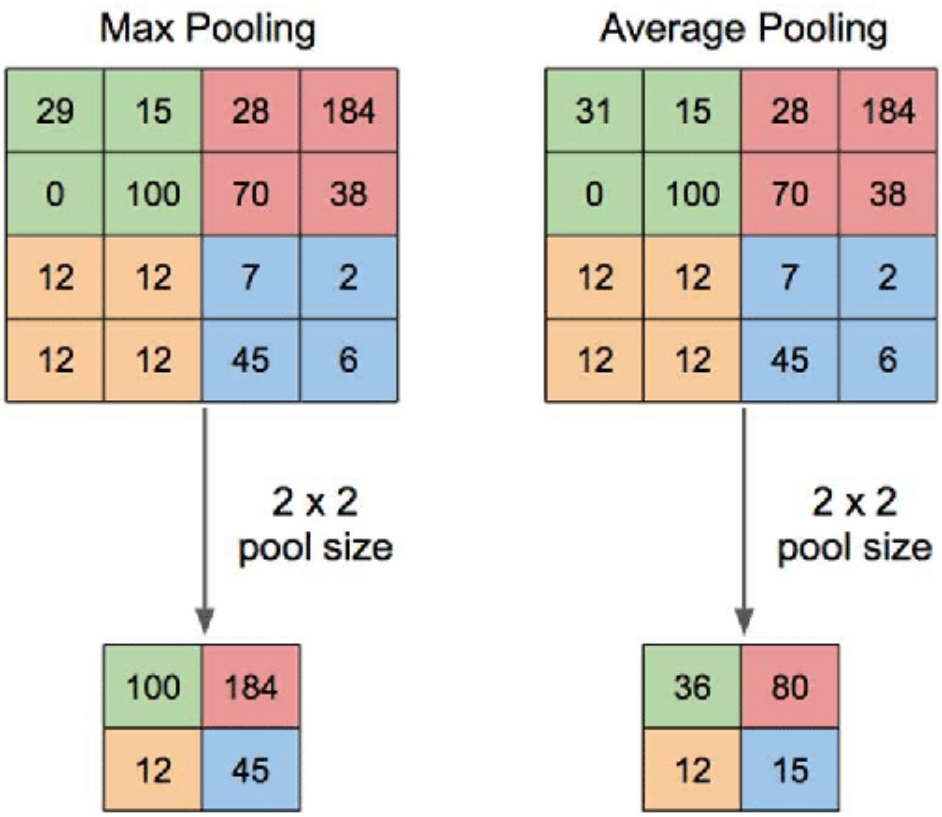


Fig. 1.7: Max and Avg pooling

1.6 Literature review

The first resource to consider when looking for fluid mechanics and machine learning is authored by Brunton et. al. [8].Fluid dynamics/mechanics and AI or machine learning have a astonishingly very long history. During 1940s, Kolmogorov considered turbulent fluid flow as the prime domain for application of machine learning [9]. When first method was developed emulate the brain [10], the hype was very quickly extinguished when it was found that the perceptron model was very insufficient even to approximate XOR or XNOR logic functions that had non-linear decision boundaries [11]. But, it was once again revived and now we have a myriad of tools and methods in AI. The paper showcases these methods available in the broad domain of AI and machine learning for fluid mechanics [12]. It discusses the history of AI and explains the daunting task of using it for fluid mechanics. The three modes of learning used in AI are supervised, semi-supervised and unsupervised. Figure 1.8 shows the learning modes with their method

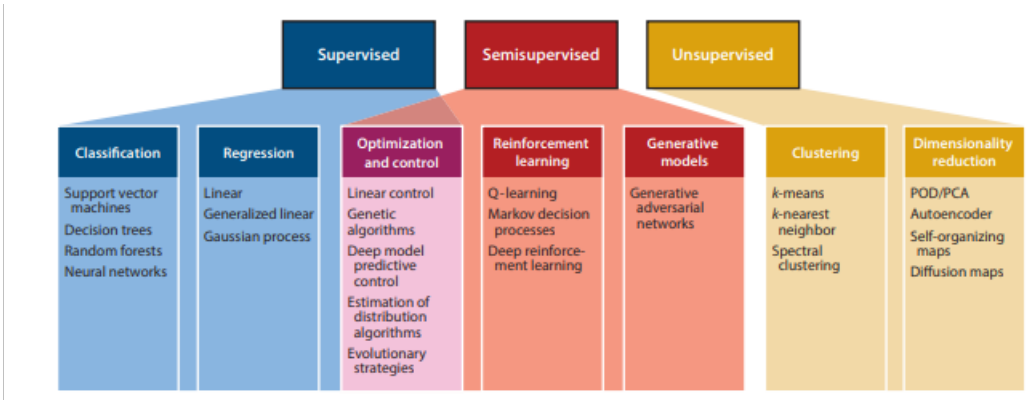


Fig. 1.8: Modes of learning borrowed from published work of Brunton et. al.[8]

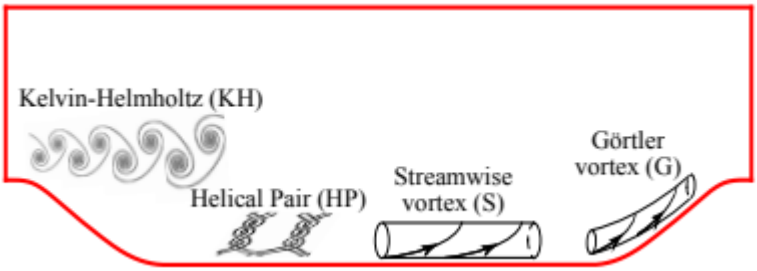
Supervised learning is used a myriad of tasks regarding fluid mechanics. Solving of differential equations using physics informed neural networks (PINNs) [13, 14] and feature identification using CNN.
Physics informed neural networks are neural networks trained to solve tasks using supervised learning constrained by laws of physics which are in turn generally represented by differential equations. The constraints are as simple as momentumm and energy conservation. It is aimed to learn about a given data under the constraints to ultimately develop a mathematical model. With PINNs another objective that can be tackled is of finding state of system defined by the chosen variables given the mathematical model. The two objectives given above are explored in the paper by Raissi et. al. [15]. The equation 1.2 is used though out the paper for exploring the said tasks

$$u_t + \aleph[u; \lambda] = 0, x \in \Omega, t \in [0, T] \tag{1.2}$$

The equation is modified according to the objective required to explore. The equation has a well defined parameter λ when working on the problem of finding

the state under which the system will be according to the given equation. On the other hand the parameter λ is left as a variable for the NN to find given a data set for an arbitrary λ. The profound power of NN to map the unknowns is why they are called the universal approximators [16].

In fluid mechanics identifying coherent flow structures is very important for analysis of flow. A turbulent flow is a treasure trove for these coherent flow structures [17–19], with structures present at nearly every spatial and temporal scales, that can effect how the flow reacts to certain control techniques. Conventionally, statistics has been widely used o identify these structures, e.g. in identification of vortex Q-criterion is used. Over time numerous coherent structures have been identified that impart significant characteristics to the flow. Few coherent structures are shown in figure 1.9.



.

Fig. 1.9: Different vortices (Image courtesy Strofer et. al. [20])

The method of manually identifying the given flow structures in new flow is time taking and cumbersome. Also, the identification process is subjective. The paper presents a novel method of identifying flow structures pictorially. The flow with different classes of flow structures is fed into the NN. The NN after training can identify flow structures without the requisite physics taking off burden of identifying flow structures manually. Thus, this method acts as a complimentary tool to extract the features at increased pace removing the hindrance of ongoing effort to establish a link between coherent structures and turbulence statistics.

The NN method usage given by preceding papers uses supervised learning mechanism. However, these are limited and most works are being carried out in unsupervised learning domain. An unsupervised learning uses a special kind of NN structure, called the autoencoder structure, mentioned in figure 1.8 on page11. Under the same category, we can also the conventional tool, POD. The conjunction between two is presented in the work by Milano et. al. [21]. POD is given by the equation 1.3

$$a(t) = \phi_g^T . u_g^T(t) \tag{1.3a}$$

$$\hat{u}_g^{'}(t) = \phi_g . a(t) \tag{1.3b}$$

The $u_g(t)$ and $\hat{u}_g^{'}(t)$ are the input and output vectors respectively. The vector $\hat{u}_g^{'}(t)$ is the reconstructed vector. The equation shows a linear transformation of the input vector. The vector $a(t)$ has the eigenvalues for the input data. A linear neural network or precisely the autoencoder performing an identity mapping for some input field/vector is given by equation 1.4. The vectors W_1 and W_2 are the weight vectors that are trained, and x_1 is the latent space vector. It is seen quiet clear that replacing these vectors with the ϕ counterpart, we recover the equation 1.3. The linear AE yields the same result as that given by POD and thus the latent space vector contain the POD modes. Building on this a non-linear POD is developed that has two additional layers that introduce non-linearity and also provides with more degrees of freedom. Although, the computation cost increase with increase in number of trainable parameters, this is mitigated by the effective compression of real data. The linear and non-linear principal component analysis (PCA) are compared via Burger's equation and the superiority of the non-linear PCA is showcased. Furthermore, the comparison is carried out for many other flow, including homogenous flow.

$$x_1 = W_1.u_g^T(t) \tag{1.4a}$$

$$\hat{u}_g^{'}(t) = W_2.x_1 \tag{1.4b}$$

The paper by Baldi et. al. [22] explores the quadratic error function surface. The paper shows the existence of a unique minimum. This unique or global minima is the corresponding to the reconstruction generated by the first principal eigen vectors. The other critical points or the saddle points are minima corresponding to the reconstruction generated by the higher order latent space vectors.

It has been shown till now by the works in papers that dense neural networks with non-linearity are superior to the conventional POD/PCA. Still, it has the drawback of becoming unmanageable when the input vector is very high dimensional. To remedy this problem, a convolutional neural network (CNN) is used [23]. The CNN is used to train and explore the heat transport properties of turbulent Rayleigh-Benard convection.

Recent work Kochkov et. al. in machine learning showcases the help that it can render to the computational model in speeding it up. A DNS for turbulent flow is computationally very heavy since it has a myriad of spatio-temporal scales. Some modeling techniques have developed like RANS and LES [24] etc. to make the life simpler. But, they are not that efficient. The modeling can also be achieved with the use of POD. All these modeling approximate the physics very quickly, but at the loss of considerable accuracy. The paper exhibits the that use of machine learning in the DNS can improve both accuracy and much more accurate lower degree freedom models containing much more information in comparison to the

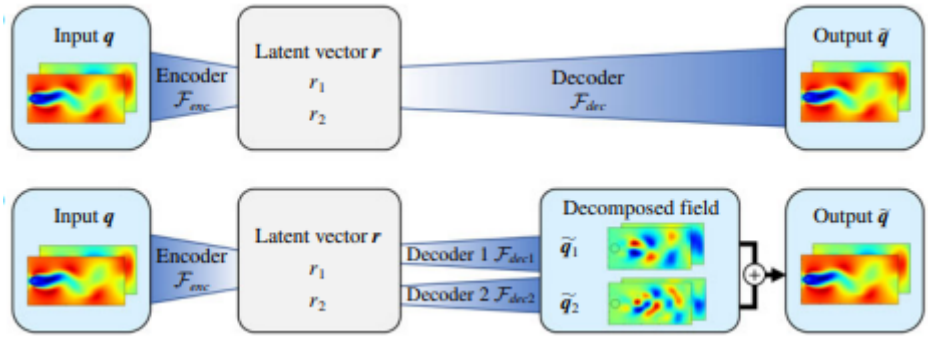


Fig. 1.10: The subtle difference betweeen C-CNN and MD-CNN (Image courtsey Murata et.al. [25])

conventional modeling techniques. The machine learning method achieves a speed up of about 86 times for a turbulent flow problem.

All the papers review till now exhibit the superiority of AI and machine learning over the conventional methods used for data compression. But, efficient compression is just half the battle. What a POD can achieve other than compression is the visualization of the modes that it gets as a product. This key feature is missing in the AI tool of neural network. To resolve this issue Murata et. al. [25] present a novel yet very simplistic way of getting the modes, i.e. the latent space vectors and visualizing them in the original grid size. Modified form of the conventional neural network, mode decomposition-CNN, acronym-ed as MD-CNN is presented. This basically attains the modes in re presentable form as achieved using POD. The figure 1.10 shows the structural difference between the the conventional and its modified counter- part. There is clearly very less difference between the two, some would argue, the difference being none at all.

Mathematically, the C-CNN-AE can be represented by the given equation:

$$r = f_{enc}(q) \tag{1.5a}$$
$$\hat{q} = f_{dec}(r) \tag{1.5b}$$

Consecutively, the modified CNN, MD-CNN is given as:

$$r = f_{enc}(q) \tag{1.6a}$$
$$\hat{q}_1 = f_{dec1}(r_1) \tag{1.6b}$$
$$\hat{q}_2 = f_{dec2}(r_2) \tag{1.6c}$$
$$\hat{q} = \hat{q}_1 + \hat{q}_2 \tag{1.6d}$$

The modification is in the decoder part, being split into the number of modes, then added to attain the reconstructed output data. The method is used for a cylinder wake problem, where two POD modes are sufficient to capture 99% of the flow energy. The MD-CNN is used with the latent vector size 2. The results are compared between POD and MD-CNN with use of 5 different activation functions including linear activation function which approximates the POD. The loss values were compared and clearly, MD-CNN with tanh activation function gave the least or the best result. Furthermore, it was shown that the AE modes contained within them the POD orthogonal modes being consistent with the result obtained by Loiseau et. al. [26]. A more complex problem of transient cylinder wake problem was also considered wherein the two modes obtained by MD-CNN contain more orthogonal POD modes suggesting that more than 2 POD modes are required to have 99% efficiency. A crude result of adding more modes to the latent vector gives the expected result of further decrease in reconstruction error. Although, it wasn't conclusively given, so more analysis was deemed necessary. Also, improvement has to be made in order to work with turbulent flows.

Two problems that arise with the aforementioned MD-CNN is that it doesn't give any information in regards to the energy contained of the real flow in the respective modes which is explicitly given when using POD. Also, like mentioned, well designed version of the MD-CNN will be required to work with flows that require quiet a lot of modes to effectively reconstruct the real flow. These two problems are addressed by Fukami et. al. [27] in their paper. The paper showcases a hierarchical autoencoder for non-linear decomposition of fluid field data. This is an extension to the work by Saegusa et. al. [28] on preserving the order of the principal components of the non-linear PCA which in turn is an extension of the work by Milano et. al. The motive of preserving order of modes is applied to the CNN autoencoder and analysis is done for the same two flows as taken up by Murata et. al. in thier paper to contrast between the two. It is also to be noted that any modified form of CNN when used with linear activation function yields the result same as that by POD. Building on the new mode decomposition CNN utilizing the hierarchical AE a problem of y-z cross-section of turbulent channel flow was considered. At last it was concluded that, even the HE-CNN had some flaws that needed taking care of.

1.7 Conclusion from literature review

From the extensive research done regarding AI and neural network, it can infer that great strides are being made in the field of AI and fluid mechanics. Although, the tools based on AI and machine learning promise great things for reduced order modeling, it has a large degree of arbitrariness inherent to it. A large number of parameters define the structure of the NN and it becomes quiet impossible to reason out the perfect NN structure for the given flow problems. Among the many parameters that make up the NN structure, a few are worth exploring and

calls for in-depth analysis. Following are the points that come up as not being fully ascertained or lack proper investigation:

1. Among myriad of activation functions, which can prove to be the most suitable for the fluid flow data.
2. When and under what conditions is the use of AI superior to the conventional method of POD.
3. Among the many types of neural network structure, which is best suited for the cause of fluid flow problems
4. The pre-processing required for the input data

1.8 Problem formulation

Based on the conclusion of literature review, it can be gathered that development of AI as a tool for dimensionality reduction needs much attention. On careful examination of these gaps in knowledge regarding the NN and and its use for dimensionality reduction of fluid flow data, following objectives have been structured. It is aimed to answer or achieve these objectives to the best of ability. Consequently, the data used to achieve these objectives are given.

1. **Objective I:** Comparing and contrasting the conventional tool of dimensionality reduction with the general and simplest autoencoder structure, the dense autoencoder with linear and non-linear activation function.
2. **Objective II:** Checking the viability and feasibility of the dense autoencoder for future use with times series fluid flow data
3. **Objective III:** Is use of full raw data from the DNS (or for that matter eperimental) possible? If no, what remedy actions can be taken to make it fit for AI learning
4. **Objective IV:** From the plethora of activation function available in the keras API, which one would prove the best in dimensionality reduction for the specific data of fluid flow
5. **Objective V:** The most important objective of all— Extraction of modes of a new periodic flow data of Vortex induced vibration (VIV)
6. **Objective VI:** As is always said for the case of AI and machine learning, it is to be explored and hence to verify that a large datasets is minimum requirement when (atleast) dealing with a complex flow.

For each of these objectives the datasets are given in the following subsections.

1.8.1 Objective I and II:Comparison of dense autoencoder with POD and its viability check for further use

Geometrical data

For the objective of comparison with POD and development of dense autoencoder 1-D and 2-D datasets are used. The 1-D dataset considered are by the equations 1.7-4.1 with their corresponding figure.

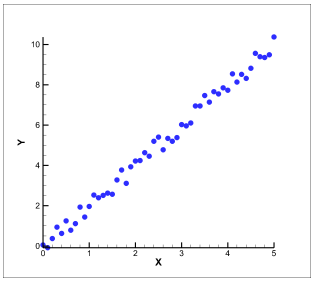
$$y = mx + \delta \tag{1.7}$$

$$y = mx^4 + \delta \tag{1.8}$$

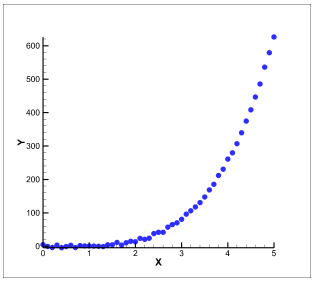
$$y = mx^8 + \delta \tag{1.9}$$

$$y = 0.1x(0.75x + cos(2x)) \tag{1.10}$$

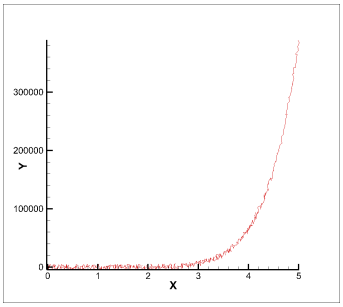
The δ in each case is the noise added to provide another dimension to the data, especially for the case of linear data set given by equation 1.7



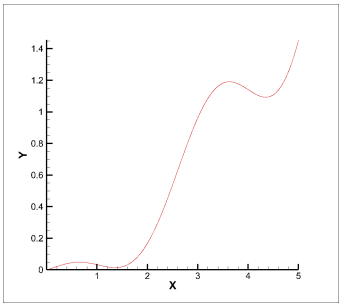
(a) 1-D linear data



(b) Non-linear data I



(c) Non-linear data II



(d) Non-linear data III

Fig. 1.11: Different 1-D dataset used for the analysis and comparison of autoencoder with POD

The data in figure 1.11a and 1.11b will be used to compare the dense autoencoder as an iterative POD with the real POD. The next two figures are used to showcase the optimize the dense autoencoder structure with some non-linear activation function.

The optimized dense autoencoder is next used on 2-D data to compress and consequently reconstruct it. And to create some perspective the POD reconstruction for same is also given using the same number of modes (or for dense autoencoder, the number of nodes in the latent space) as with the dense autoencoder. The first data is an arbitrarily obtained data given by equation 1.11. The arbitrary data is 2-D parabola in I quadrant in essence and is just used as an experimental data. The figure 1.12 gives the contour plot.

$$f(x, y) = x^2 + y^2 \tag{1.11}$$

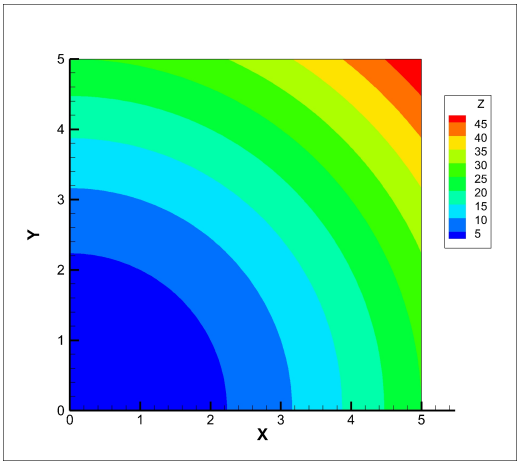


Fig. 1.12: Contour plot of a 3-D parabola in I quadrant

Fluid flow data

For the sake of simplicity first steady state flow is considered. The first of the steady state flow is the lid driven flow [29–32]. For two-dimensional steady incompressible flow in rectangular geometry, the governing equations for stream function and vorticity are:

$$\frac{\partial^2 \Psi}{\partial x^2} + \frac{\partial^2 \Psi}{\partial y^2} = -\omega \tag{1.12a}$$

$$u\frac{\partial \omega}{\partial x} + v\frac{\partial \omega}{\partial y} = \upsilon\left(\frac{\partial^2 \omega}{\partial x^2} + \frac{\partial^2 \omega}{\partial y^2}\right) \tag{1.12b}$$

$$u = \frac{\partial \Psi}{\partial y} \; and \; v = \frac{\partial \Psi}{\partial x} \tag{1.12c}$$

The boundary conditions for the lid driven flow are:

$$u = 0, v = 0 \; at \; x = 0 : u = 0, v = 0 \; at \; y = 0$$

$$u = 0, v = 0 \; at \; x = L; u = U_\circ, v = 0 \; at \; y = H$$

The equation is solved numerically using a simple Gauss-Seidel approach for Re=100 at which the flow is steady. The resulting steady state stream function contour is shown in the figure 1.13.

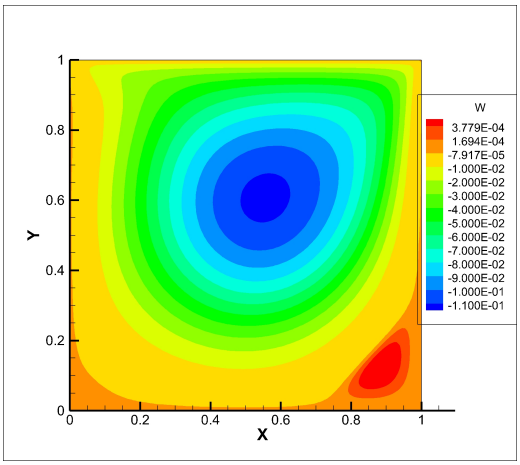


Fig. 1.13: Streamfunction contours of DNS data of Lid driven cavity problem at Re=100

The next dataset for training of dense autoencoder is the inlet/outlet or room flow problem. The governing equation for this flow is the vorticity-stream function formulation of N-S equation given by equation 1.12. The boundary conditions of the flow are given pictorially in the figure 1.14. Similar to the lid driven flow, the GS solver is used. The problem is solved with the the kinematic viscosity as $0.00025m^2/s$, the velocity at the inflow plane is $5m/s$, and dimension of the room is $40cm$ *by* $40cm$.

The final steady state stream function, Ψ, is shown in figure 1.15.

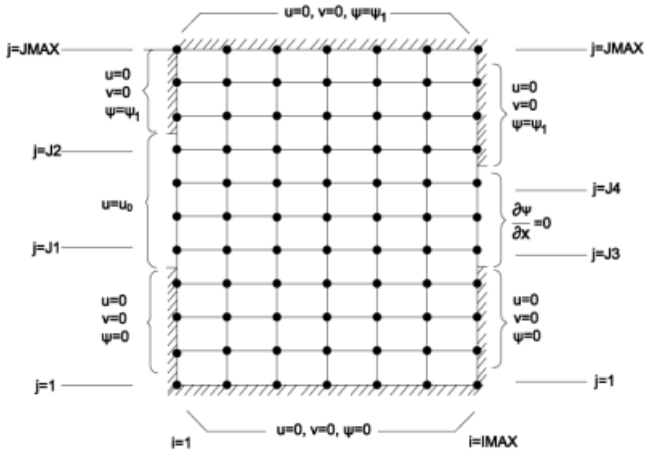


Fig. 1.14: Pictorial and nodal representation of BCs for the inlet/outlet flow problem

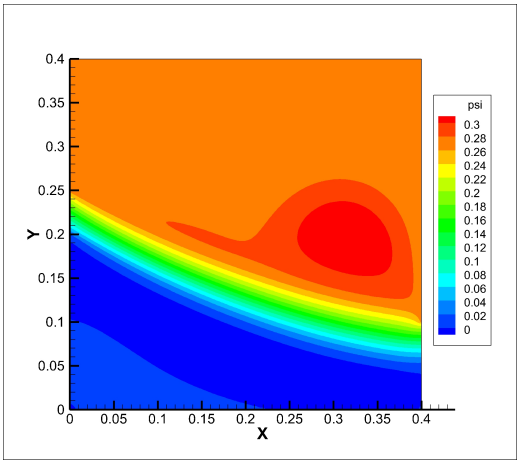


Fig. 1.15: The steady state Ψ contour

1.8.2 Obejctive III: Use of raw DNS data for the training

Vortex Induced Vibration

The flow data of a periodic flow vortex induced vibration is considered. The flow is 2D at a reynolds number of Re=100. From the many cases presented in the paper by Khan et. al. [33] the case with $U_{red} = 3$, $Ri = 0.25$ and angle of approach as 15°. The dimensionless form of Continuity, Navier-Stokes and Energy equation for 2-D incompressible flow are given in equation 1.13.

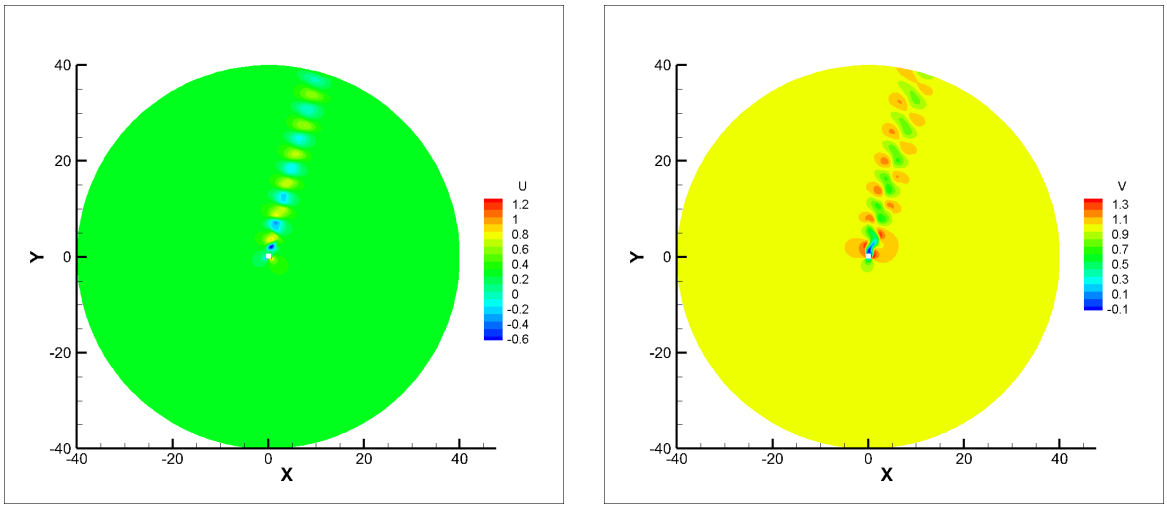
$$\left(\xi_x \frac{\partial}{\partial \xi} + \eta_x \frac{\partial}{\partial \eta}\right) U + \left(\xi_y \frac{\partial}{\partial \xi} + \eta_y \frac{\partial}{\partial \eta}\right) V = 0 \tag{1.13a}$$

$$\left(\frac{\partial U}{\partial \tau} + \left(U^{\xi} - U_g^{\xi}\right)\frac{\partial U}{\partial \xi} + \left(U^{\eta} - U_g^{\eta}\right)\frac{\partial U}{\partial \eta}\right) = -\left(\xi_x \frac{\partial p}{\partial \xi} + \eta_x \frac{\partial p}{\partial \eta}\right) + \frac{1}{Re}\nabla^2 U \tag{1.13b}$$

$$\left(\frac{\partial V}{\partial \tau} + \left(U^{\xi} - U_g^{\xi}\right)\frac{\partial V}{\partial \xi} + \left(U^{\eta} - U_g^{\eta}\right)\frac{\partial V}{\partial \eta}\right) = -\left(\xi_y \frac{\partial p}{\partial \xi} + \eta_x \frac{\partial p}{\partial \eta}\right) + \frac{1}{Re}\nabla^2 U \tag{1.13c}$$

$$\frac{\partial \theta}{\partial \tau} + \left(U^{\xi} - U_g^{\xi}\right)\frac{\partial \theta}{\partial \xi} + \left(U^{\xi} - U_g^{\xi}\right)\frac{\partial \theta}{\partial \eta} = \frac{1}{Re.Pr}\nabla^2 \theta \tag{1.13d}$$

The above equations are solved numerically and the DNS U and V velocity contour are shown in figure 1.16 for some reference snap. The full raw DNS data has a grid size of 375×305 with **180** time snaps available.

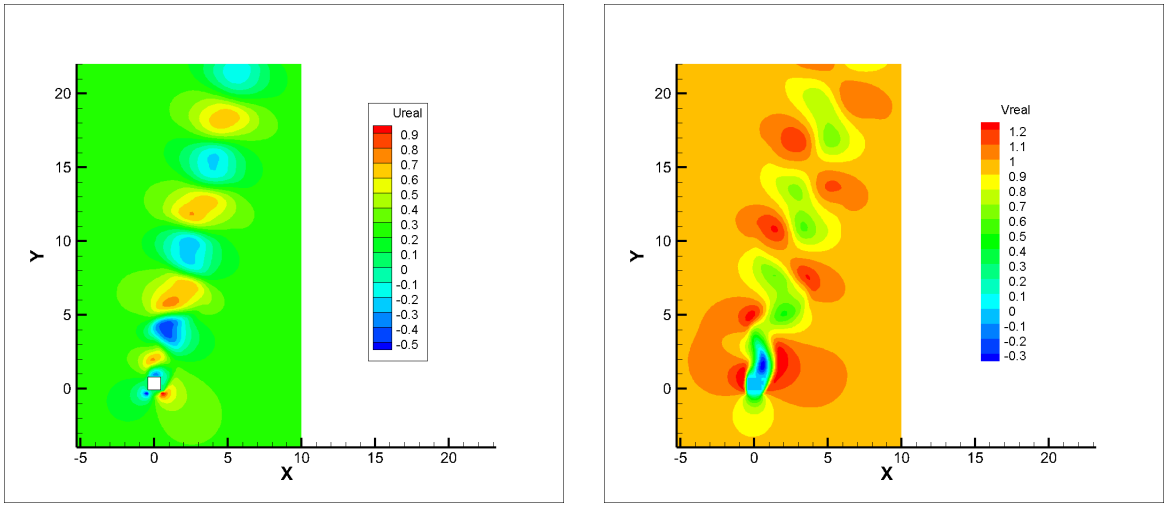


(a) (b)

Fig. 1.16: The U and V velocity contour

1.8.3 Objective IV: What activation function is best suited for fluid flow data

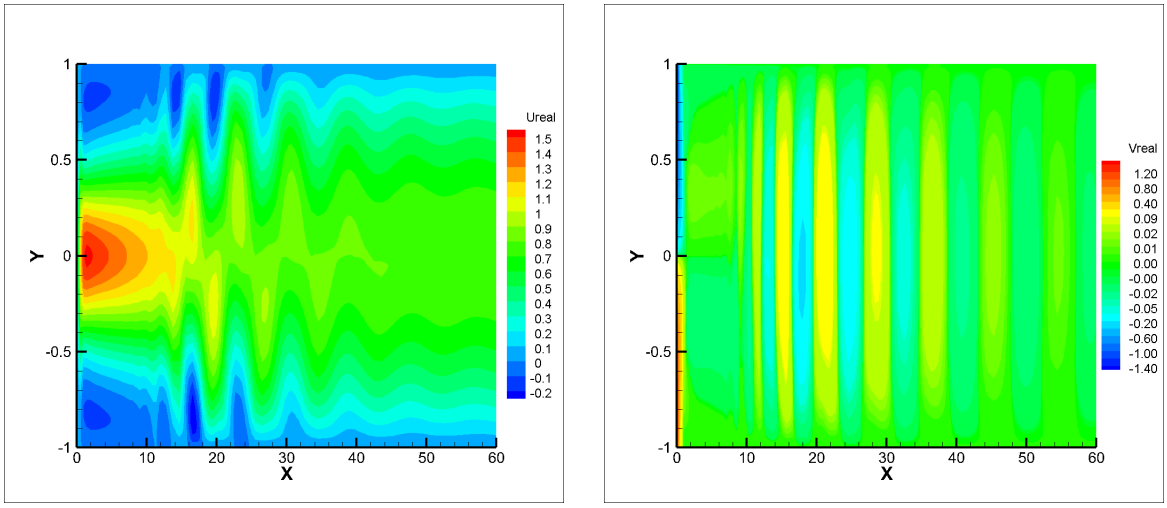
For investigation of the activation function objective, the data shown in figure 1.17 is used. This data is the rectangular extracted zone of the full raw data of VIV flow given in figure 1.16. The extracted zone data has a grid size of 128×128 that contains the crux of the flow discarding the portion where there very small to no flow feature, except for the incoming flow.



(a) (b)

Fig. 1.17: The U and V velocity contour for VIV with $U_{red} = 3$, $Ri = 0.25$ and angle of attack=15°

In addition to this, the need to completely ascertain the best choice of activation function for the input data of fluid flow, one another flow is considered. This flow has been a obtained from the work presented in the paper by Hasan and Khan [34]. The details of which can be found in the paper titled *Two-dimensional interactions of non-isothermal counter-flowing streams in an adiabatic channel with aiding and opposing buoyancy*. Presenting the details here has been omitted as the data used assists in confirming the objective and is beyond the scope of this thesis. The U and V velocity are shown in figure 1.18.



(a) *(b)*

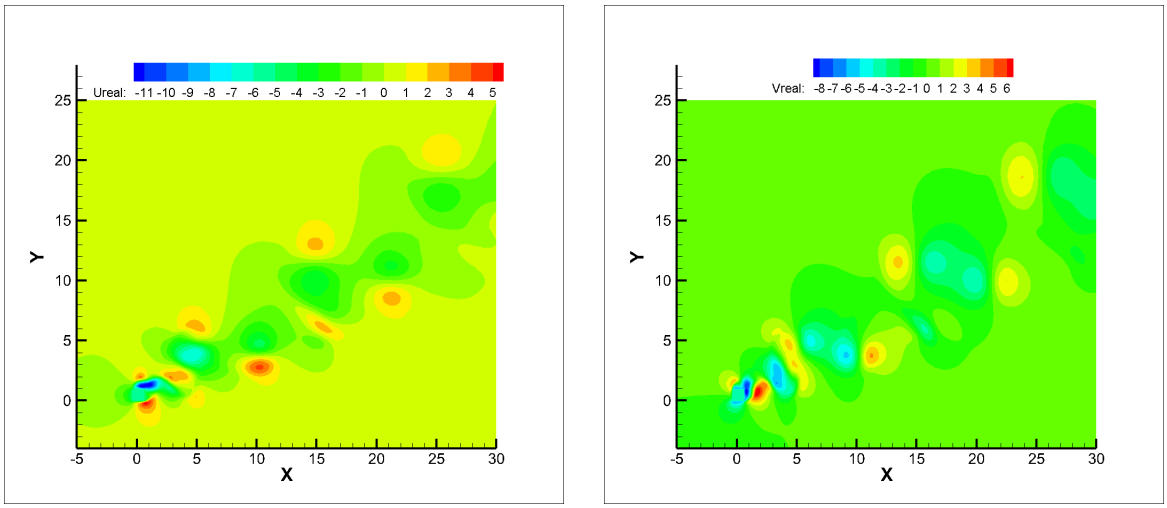
Fig. 1.18: The U and V velocity contour for the 2-D flow of jet impingment

1.8.4 Objective V: Upgrading the C-CNN to MD-CNN for extraction of modes

The most important objective had been to extract the modes of the periodic VIV flow data. The details of data whose modes are extracted is given in figure 1.17. All the other objectives helped to reach the point of extracting the modes of the required flow.

1.8.5 Objective VI: Number of samples that pertain to good learning

The last objective is more of a effect than a cause. It is effect of the cause of using the complex counterpart of the VIV flow data (having again 180 time snaps)shown in figure 1.19. The specifications of the flow are $U_{red} = 6$, $Ri = 0.25$ and angle of approach as 60°.



(a) *(b)*

Fig. 1.19: The U and V velocity contour for VIV with $U_{red} = 6$, $Ri = 0.25$ and angle of attack=60°

2. METHODOLOGY

2.1 Objective I and II: POD and dense autoencoder

To attain the objectives mentioned in the section of problem formulation, two algorithms: POD and dense autoencoder have to be developed.

The POD code is developed using MATLAB which used built in functions of singular value decomposition (SVD). The code is independently developed and the code thus developed is validated which is presented in the validation section.

The dense autoencoder (and for that matter all AI models) are coded with the use of python with keras (that uses Tensorflow as backend AI library). The python version used is 3.7.10 as this was the then, the highest version compatible with the tensorflow version. The tensorflow version is 2.3.0 and the keras version is 2.3.1. The dense autoencoder (and similarily for POD) the input data has 2 degrees of freedom (or modes) and 3 degrees of freedom for the 1-D and 2-D datasets respectively. Since, the datasets used have quite a large range of values, the data needs to pre-processed. From the *sklearn* package the *StandardScaler* object is used to reduce the data's variance yet preserve all the important features of the data. Same pre-processing is done for POD using the in-built functions available in MATLAB.

The simplest of that is the Dense autoencoder with single hidden layer and linear activation function, mimicking the POD. The figure 2.1 is the the model for a single layer dense autoencoder with linear activation.

The summary of the full autoencoder structure with the number of trainable parameters is given in table 2.1

The above structure can be mathematically surmised by equations 2.1:

$$A^2 = f(W^2.X^T) \tag{2.1a}$$

$$A^3 = f(W^3.(A^2)^T) \tag{2.1b}$$

where,

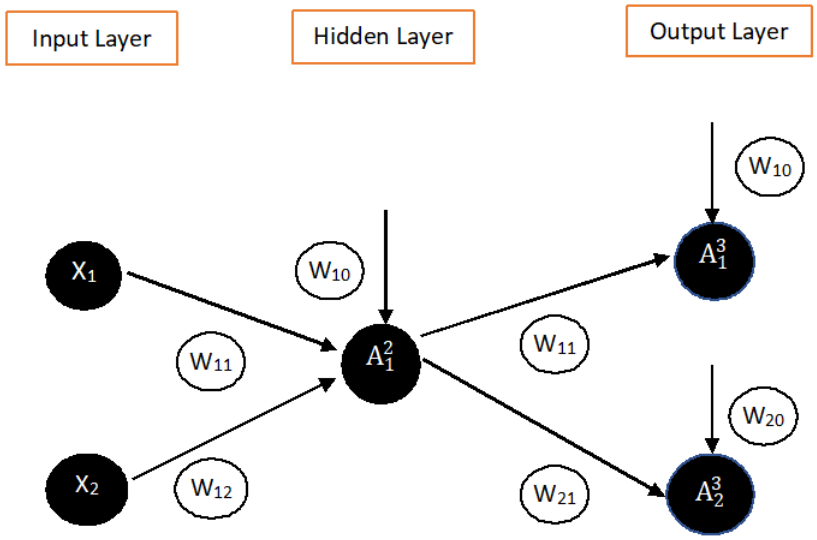


Fig. 2.1: Schematic diagram forDense Autonecoder with single hidden layer

Tab. 2.1: Summary of the AE structure

Layer(type)	Output shape	Number of parameters
input1(input layer)	(None,2)	0
dense 1(dense layer)	(None,1)	3
dense 2(dense layer)	(None,2)	4
Total parameters = 7		
Trainable parameters = 7		
Non-trainable parameters = 0		

f (in this case) is the linear activation fucntion.
X is the input vector being the X and Y coordinates of the data with a bias unit
A^2 and A^3 are the output vectors of the corresponding layers.
W^2 and W^3 are the weight vectors

An AE with non-linear activation function can only capture non-linearity with more than 1 hidden layer as shown in figure 2.2. That is to say that one hidden layer with non-linear activation function reproduces the same result as that done by linear activation function. For the AE to capture non-linearity, it must have more than one hidden layer. Mathematically, the structure is given by equation 2.2

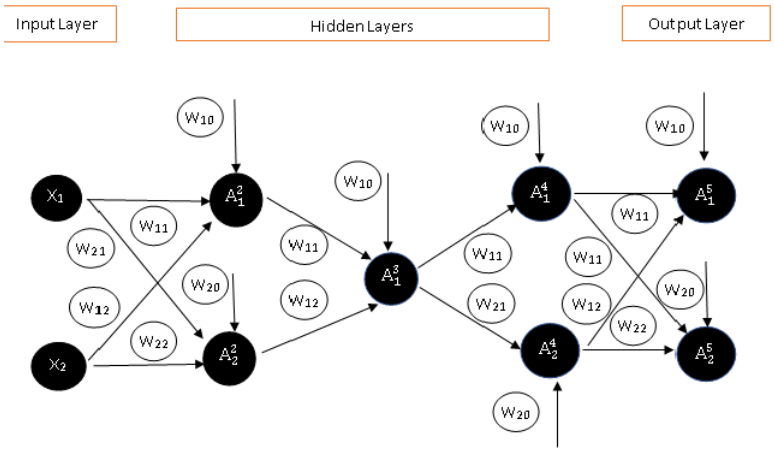


Fig. 2.2: Schematic diagram for a basic AE structure with non-linear activation function

$$A^2 = f(W^2.X^T) \tag{2.2a}$$
$$A^3 = f(W^3.(A^2)^T) \tag{2.2b}$$
$$A^4 = f(W^4.(A^3)^T) \tag{2.2c}$$
$$A^5 = f(W^4.(A^4)^T) \tag{2.2d}$$

where, A^2, A^3, A^4, A^5 are the output vectors for the appropriate layer W^2, W^3, W^4, W^5 are the weight vectors for each layer and X is the input vector to the AE f is the non-linear activation function used which is Leaky-ReLU given by equation 2.3 [1]

$$y = f(X) = \begin{cases} X & \text{if } X > 0 \\ \alpha X & \text{if } X \leq 0 \end{cases} \tag{2.3}$$

Tab. 2.2: Summary of the non-linear AE structure

Layer(type)	Output shape	Number of parameters
input1(input layer)	(None,2)	0
dense 1(dense layer)	(None,1)	9
dense 2(dense layer)	(None,2)	10
Total parameters = 19		
Trainable parameters = 19		
Non-trainable parameters = 0		

The training of dense autoencoder is done using the ADAM or adaptive momentum estimation algorithm. This optimization technique is extension of the Stochastic

[1] For deatils on activation functions used refer to appendix

gradient descent [35], which is the basic algorithm used to train the neural networks. The gradient descent training algorithm is explained in detail in the next section.

2.2 Training of neural networks

The tools or algorithms used for this project are the dense autoencoder and convolutional autoencoder and thereupon the modified CNN. All these are basic neural networks and follow a very basic learning method. The most common of these methods is the Gradient descent. Gradient descent is one of the basic optimization algorithms to find local minima in the functions. The gradient descent algorithm is mathematically given by the equation 2.4

$$x_{n+1} = x_n - \alpha \nabla f(x_n) \tag{2.4}$$

The function f is the cost function in case of machine learning that is to be minimised. The cost function is basically, a relation between the input and desired mapped output. There are in general three basic cost functions:

1. Linear cost function
2. Quadratic cost function
3. Cubic cost function

For the case of unsupervised learning the quadratic type cost function is used, known as mean squared error (MSE) given by following equation

$$\frac{1}{n}\sum_{i=1}^{n}(X_i - \hat{X})^2$$

where, X is the input data and $\hat{X}$ is the reconstructed data.

The minima sought is quiet clearly zero, as we want perfect mapping of input to the output. Since, the gradient descent is an iterative method, the sought after minima is not achieved. In equation 2.4 the parameter α is called the learning rate. This is very crucial in the learning process. A learning rate has to be just right to attain the global minima. IF the learning rate is too high then it can overshoot the minima and miss it completely. If the learning is too low then it takes very large amount of time to attain the global minima or in many cases it gets stuck at the local minima. The figure 2.3 shows the negative effects for both high and low learning rate.

The figure 2.4 shows the loss curve for different learning rates. Thus, it can be inferred that a good choice of learning rate is of paramount importance for good training of the neural network.

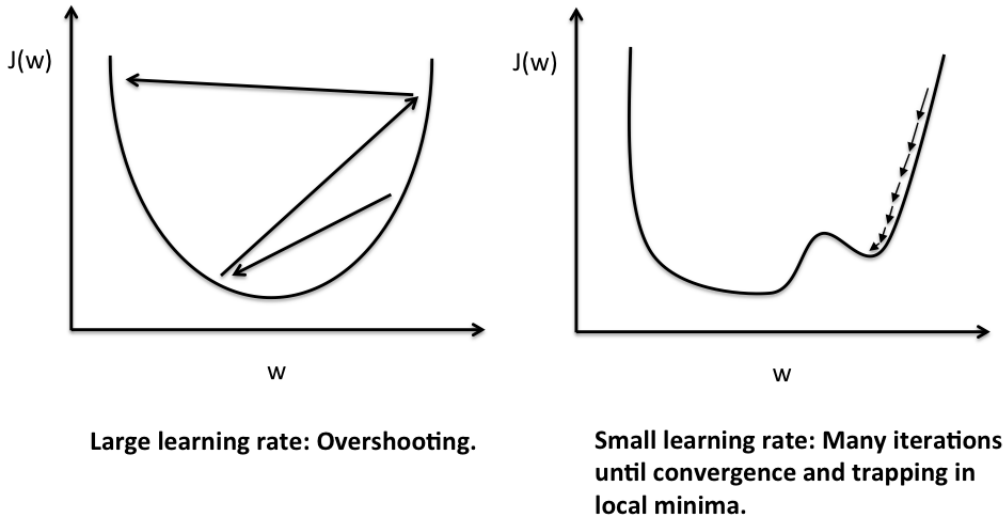


Fig. 2.3: Negative effect of different learning rate

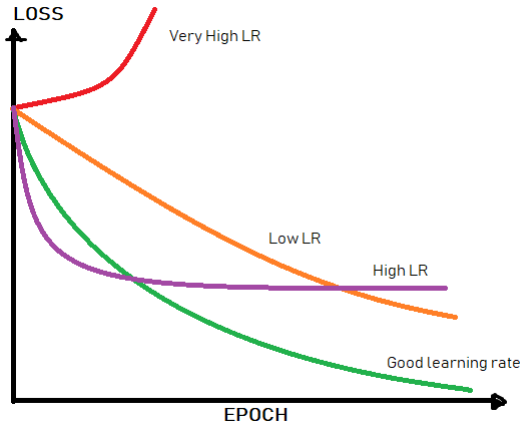


Fig. 2.4: Loss v/s epochs for different learning rate

Optimization of weights is done by the gradient descent as discussed previously. But, for all the weights in a neural network, the calculation of gradients for the purpose of optimizing weights is done using back-propagation. Back-propagation is short for backward propagation of errors, is calculation of errors for each layer. Back propagation aims to minimize the error function given in equation 2.5 with respect to the neural network's weights.

$$E(X,\theta) = \frac{1}{2N}\sum_{i=1}^{N}(X_i - \hat{X}_i)^2 \tag{2.5}$$

where, θ is the weight vector. For minimizing the error, or optimizing the weights, the value of $\frac{\partial E}{\partial \omega_{ij}^k}$ is to be calculated which is given in equation 2.6

$$\frac{\partial E}{\partial \omega_{ij}^k} = \frac{1}{N}\sum_{i=1}^{N}\left(\frac{1}{2}(\hat{X}_d - X_d)\right) = \frac{1}{N}\sum_{i=1}^{N}\frac{\partial E_d}{\partial \omega_{ij}^k} \tag{2.6}$$

.3The gradient for each weight can be calculated as given in equation 2.6. Once calculated the weights can be optimized using the gradient equation given in equation 2.4.

The steps followed in back propagation alogorithm are itemized below.

- The input X is follows a predefined path to get to the output layer
- The inputs on the way are modeled with the real weights or trainable parameters.
- With the trainable parameters the output for every layer is calculated
- The error in outputs is calculated for each layer
- The errors travel back from the output layer hence the name back-propagation.

2.3 Objective III, IV and VI: C-CNN

For time series data a convolutional neural network is used. The convolution operation is an integral that expresses the amount of overlap of one function as it is shifted over other function. Mathematically it is given by equation 2.7.

$$(f * g)(t) = \int_{-\infty}^{\infty} f(\tau)g(t-\tau)\,d\tau \tag{2.7}$$

The input data of time series data is the velocity field $U(X,t)$ where X is space vector of x and y. The fluctuating part of this data is used as the input for the neural network given by equation 2.8

$$u(X,t) = U(X,t) - U_{avg}(X) \tag{2.8}$$

The velocity field sans the avg flow field or mode 0 is the input layer for the C-CNN. The structure of the C-CNN is given in detail. The table 2.4 is the summary of the structure of the encoder part of the C-CNN, while the table 2.5 gives the details for the decoder part of the C-CNN. The autoencoder is the union of the encoder and the decoder part. The basic building block of a CNN is shown in figure 2.5 which is also summarized in the table. Every CNN contains the features extraction part. An AE counterpart of CNN basic building block has a dense layer at the end ending at the latent space. The convolution layer creates the

many feature maps while the pooling layer (max pooling most case)reduces the dimensions of the convolutional layer.

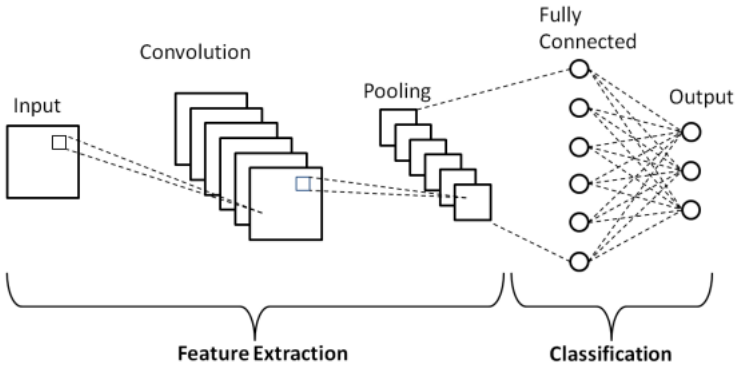


Fig. 2.5: Basic building block of CNN

Tab. 2.3: Summary of basic CNN building block

Layer(type)	Output shape	Number of parameters
input1(input layer)	[(None,X,X,1)]	0
conv2d(Conv2D)	(None, X,X,6)	-
maxpooling2d(Maxpooling2D)	(None, $\frac{X}{2}, \frac{X}{2}$),6	0
reshape (reshape)	(None,6)	0
dense (Dense)	(None, 3)	-

With the basic building block of CNN given, the specific CNN structure used with the objectives is presented with the encoder and decoder parts seperately. It can be seen that both (encoder and decoder) contains the *conv2D* layer's multiple instances. On one hand the encoder part contains the *Maxpooling*, on the other hand the decoder part contains the *Upsampling* layer.

Tab. 2.4: Summary of the Encoder of C-CNN

Layer(type)	Output shape	Number of parameters
input1(input layer)	[(None,128,128,2)]	0
conv2d(Conv2D)	(None, 128,128,16)	304
maxpooling2d(Maxpooling2D)	(None, 64,64,16)	0
conv2d 1(Conv2D)	(None, 64,64,8)	1160
maxpooling2d 1(Maxpooling2D)	(None, 32,32,16)	0
conv2d 2(Conv2D)	(None, 32,32,8)	584
maxpooling2d 2(Maxpooling2D)	(None, 16,16,8)	0
conv2d 3(Conv2D)	(None, 16,16,8)	584
maxpooling2d 3(Maxpooling2D)	(None, 8,8,8)	0
conv2d 4(Conv2D)	(None, 8,8,4)	292
maxpooling2d 4(Maxpooling2D)	(None, 4,4,4)	0
conv2d 5(Conv2D)	(None, 4,4,4)	148
maxpooling2d 5(Maxpooling2D)	(None, 2,2,4)	0
reshape (reshape)	(None,16)	0
dense (Dense)	(None, 2)	34
Total parameters = 3106		
Trainable parameters = 3106		
Non-trainable parameters = 0		

Tab. 2.5: Summary of the Decoder of C-CNN

Layer(type)	Output shape	Number of parameters
decoder input(input layer)	(None, 2)	0
dense 1 (Dense)	(None, 16)	48
reshape 1 (Reshape)	(None, 2, 2, 4)	0
up sampling2d (Upsampling2D)	(None, 4, 4, 4)	0
conv2d 6 (Conv2D)	(None, 4, 4, 4)	148
up sampling2d 1(Upsampling2D)	(None, 8, 8, 8)	0
conv2d 7 (Conv2D)	(None, 8, 8, 8)	296
up sampling2d 2(Upsampling2D)	(None, 16, 16, 8)	0
conv2d 8 (Conv2D)	(None, 16, 16, 8)	584
up sampling2d 3(Upsampling2D)	(None, 32, 32, 8)	0
conv2d 9 (Conv2D)	(None, 32, 32, 8)	584
up sampling2d 4(Upsampling2D)	(None, 64, 64, 16)	0
conv2d 10 (Conv2D)	(None, 64, 64, 16)	1168
up sampling2d 5(Upsampling2D)	(None, 128, 128, 16)	0
conv2d 11(Conv2D)	(None, 128, 128, 2)	290
Total parameters = 3118		
Trainable parameters = 3118		
Non-trainable parameters = 0		

The CNN used for objective VI; for quasi periodic VIV flow has a input size of 256×256, adding another *conv2D* and *maxpooling(or upsampling layer)*

A discussion on the data input to the NN is a must.
Unlike the dense autoencoder the data for C-CNN (or MD-CNN) dosen't go under any pre-processing, only the 0th mode- time average flow field is deducted. Additionally, the sample set is split into two different set being the **Training** and **Validation/test** set. The number of samples available for the VIV flow is 180 and thus a 170/10 split is done for the training and validation set respectively. After this the data is trained and results are inferred.

2.4 Objective V: MD-CNN

The modified form of CNN is the MD-CNN. It is a way developed by Murata et. al. [25] to extract and thus visualize the modes/latent space of the C-CNN structure given in table 2.4 and 2.4. The structure of MD-CNN (pictorially represented in figure1.10) also comprise of the encoder and the decoder part. The encoder part is exactly similar to the structure provided in table 2.4. The difference arise in the decoder part. The decoder part is split depending on the number of modes or latent space dimension and each split is passed through the decoder similar to the one presented in table 2.5. Since, the input dimension of the decoder has to be of shape 2, thus it is required to input 2 dimensional vector. For this the 1 node of latent space is cloned and thus used as a input of shape 2 using he lambda function.

3. VALIDATION

3.1 Validation of POD

The different tools/algorithm developed for the attainment of objectives are validated. The different as mentioned in methodology are POD as a MATLAB code, dense autoencoder and C-CNN autoencoder in python.

The POD code is developed in MATLAB. To check the viability of the developed code the best method is to reconstruct a data with all the modes available. This is done using the data given in 1.11b on page 17. The data has enough non-linearity rendering the POD reconstruction with 1 of the 2 available modes very poor. Thus, a POD reconstruction with 2 modes must give perfect reconstruction. The reconstruction of data with all the available modes i.e. 2 is given in figure 3.1.

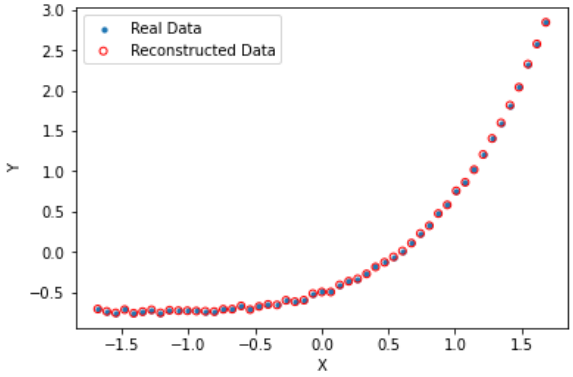


Fig. 3.1: Reconstruction of data using the 2 available modes

The two modes of POD are shown in figure 3.2. The modes are clearly vectors giving the direction and magnitude of variation in data. The blue colored vector (1st mode) given as $0.707\hat{i} + 0.707\hat{j}$ captures the maximum variation of the data. The second vector is orthogonal to the 1st mode $0.707\hat{i} - 0.707\hat{j}$ and captures the lateral variation in the data.

It is noted that similar to the above 1-D data, the POD with all the available

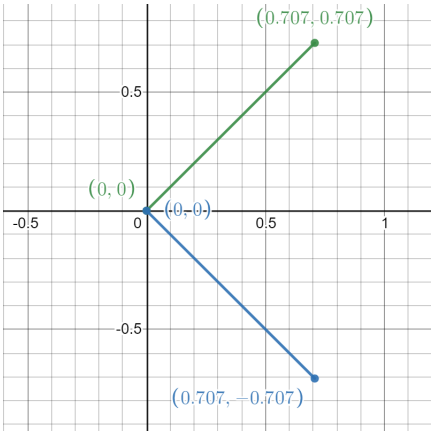


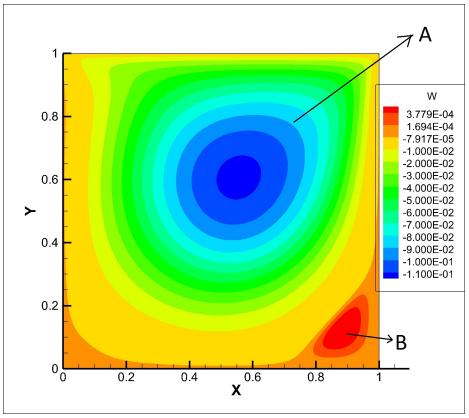
Fig. 3.2: The two modes of POD

modes (3 for the case of 2-D) yields perfect reconstruction for the any of the given 2-D data.

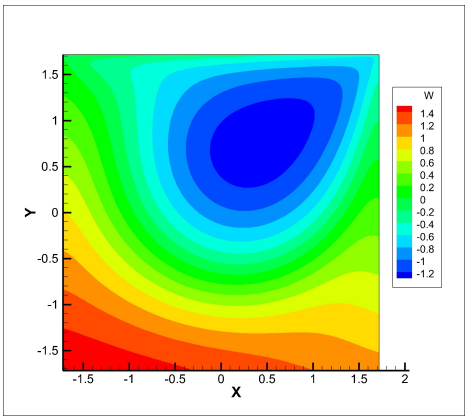
3.2 Validation of dense autoencoder

Next it is required to validate the dense autoencoder that is used for both 1-D and 2-D datasets. The validation is based on the reconstruction efficiency promised of using the dense autoencoder voer POD with the appropriate non-linear activation function. Comparing and pointing the important reconstruction features helps in validating the fact of superior reconstruction of the autoencoder.

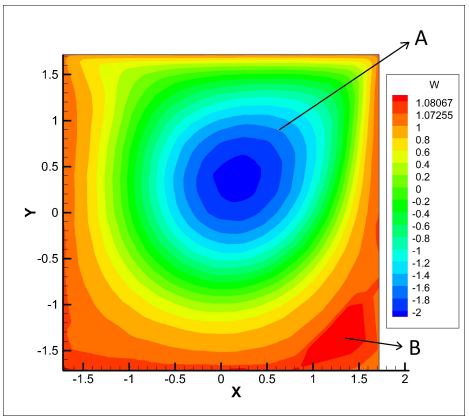
The figure 4.9 shows the original contour. The important discernible features are **A** and **B**. It can be clearly seen that POD can't even reconstruct feature A let alone B. On the other hand the reconstruction using dense autoencoder with non-linear activation function renders both features. Wheres feature A is indistinguishable from that in real data, the feature B is also reconstructed with high level of accuracy. This concludes the validation of dense autoencoder being the superior of the two dimensionality reduction tools, atleast on reconstruction quality.



(a) Original data



(b) POD



(c) Non-linear AE

Fig. 3.3: Reconstruction using POD and non-linear AE for 2D lid driven flow

3.3 Validation of C-CNN

The modal analysis done on fluid data is mostly time series and usually contains a very large number of data points in a single time step. A dense/dense autoencoder becomes very cumbersome to work with as it has very high number of trainable parameters. Also, it is not very efficient in extracting the features. To remedy the above two problems a conventional type convolutional neural network is used(C-CNN). C-CNN is very efficient in extracting features out of a data all the while reducing the number of trainable parameters.

The data used for validation is given in the paper by *Murata et. al.* [25] in pickle file format. The data is fed into the C-CNN with the following autoencoder structure given in table 3.1

The data had 2000 snaps available for training, of which 1500 were used for the purpose of training and remaining were used for validation. A validation set is

Tab. 3.1: Summary of the non-linear AE structure for 2-D

Layer(type)	Output shape	Number of parameters
AE input(input layer)	[(None,384,192,2)]	0
Encoder model (model)	(None,2)	3218
Decoder model (model)	[(None,384,192,2)]	3286
Total parameters = 6504		
Trainable parameters = 6504		
Non-trainable parameters = 0		

given to provide for a general learning of the neural net preventing the learning to be over-fitting. A stopping criterion was employed to enforce early stopping in case of convergence making use of the loss calculated for validation set (also called test set). The reconstructed contours are plotted and shown in figure 3.4. Two snaps are shown(of the 2000 available snaps) being the 1000th and the 1500th snap. Figure 3.4 gives the velocity contour reconstruction for 1000th snap, figure 3.5.

The qualitative reconstruction for the 1000th snap is visibly quiet good with very little distinction. The reconstruction obtained by the paper [25] is similar with a slightly better MSE value, owing to longer training and different dataset split for the training. Because of time crunch a cruder validation was performed which yielded near perfect validation.

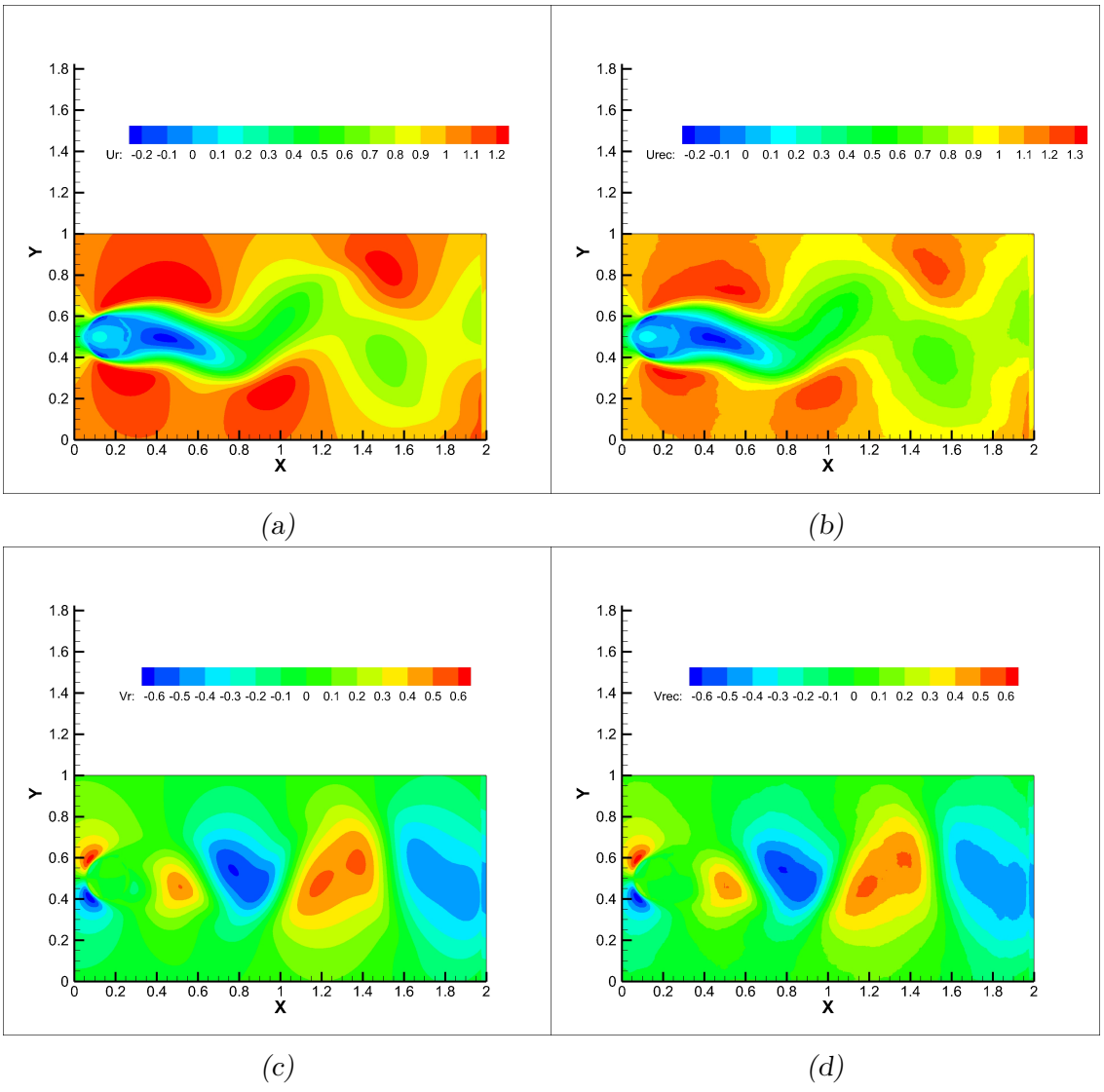
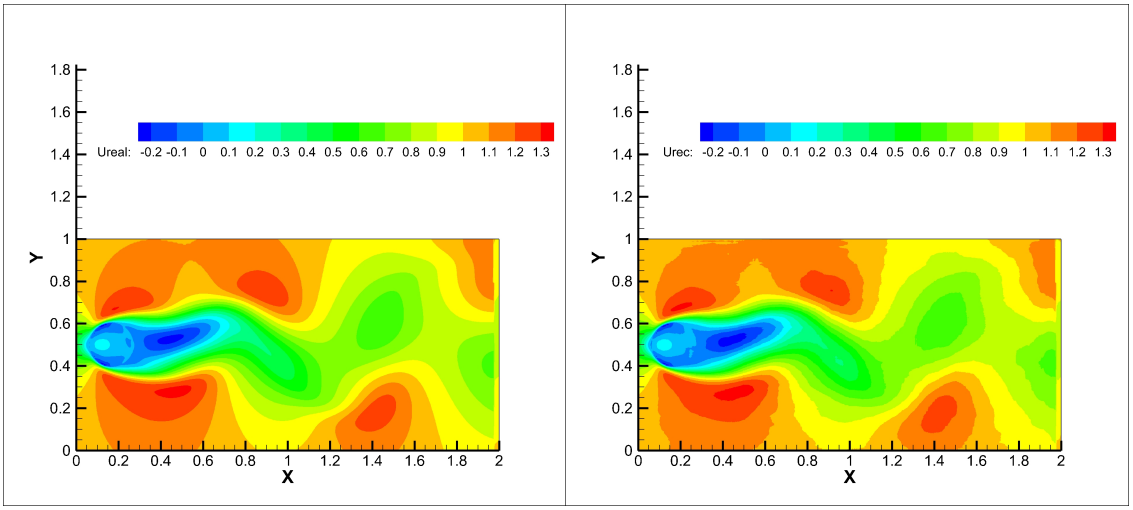


Fig. 3.4: Reconstruction for 1000th snap. (a) and (b) are the real and the reconstructed horizontal velocity contour respectively. (c) and (d) are the real and reconstructed vertical velocity contour respectively.



(a) (b)

(c) (d)

Fig. 3.5: Reconstruction for 1500th snap. (a) and (b) are the real and the reconstructed horizontal velocity contour respectively. (c) and (d) are the real and reconstructed vertical velocity contour respectively.

Quantitatively, the reconstruction error or MSE value obtained is 8.21e-5. The error is quiet good suggesting a very good learning which is also visible from the contour plots.

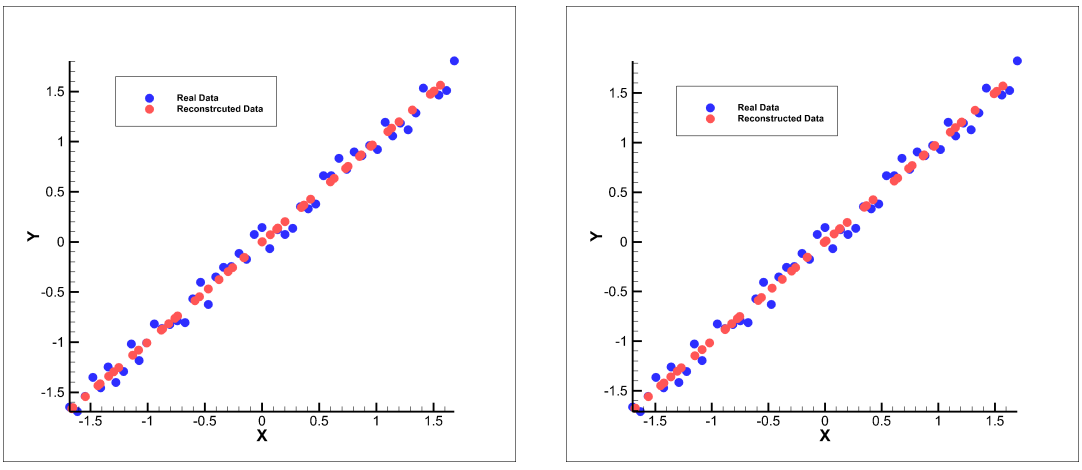
4. RESULTS AND DISCUSSION

4.1 Objective I and II: Comparing POD and dense autoencoder

The autoencoder network structure given in figure 2.1 on page 25 is taken for the linear autoencoder, which is the simplest structure for the autoencoder. The result obtained for this is compared with POD along the way to build compare the two .

4.1.1 Linear autoencoder

First the data given in figure 1.11ais used which has linear characteristics. Both POD and linearly activated autoencoder are trained. For the linear autoencoder the NN structure shown in figure 2.1 on page 25 is used. As already mentioned, this is the simplest structure possible for a NN. Also, a NN with linear mapping makes the used of extra layers redundant.



(a) POD *(b)* Linear AE

Fig. 4.1: Comparison of POD and Linear Autoencoder for straight line data

The quantitative value of POD reconstruction is $2.2e-3$ while the that for linear AE is $2.3e-3$. It is to be noted that the training for the data of the linear

autoencoder is done for 5000 epochs without any convergence criteria using ADAM optimizer [1]. The lack of convergence criteria is clearly visible in the loss (MSE)vs epochs curve shown in figure 4.2. The convergence took place at early epochs, but absence of the said convergence criteria renders the training to not stop.

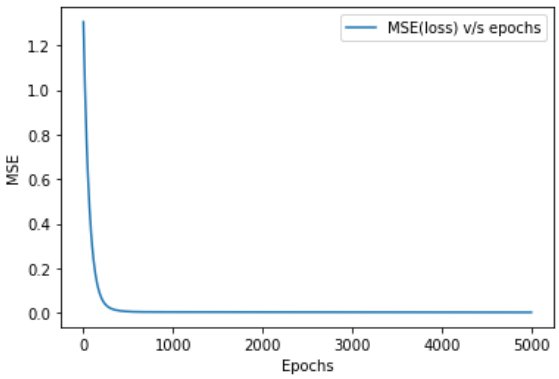


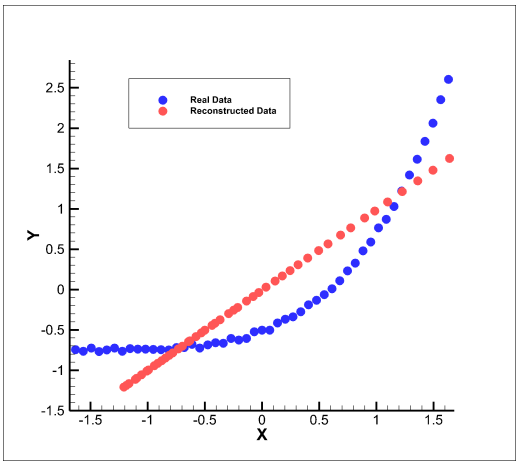
Fig. 4.2: variation of loss with epochs

Next, a data with non-linear inherent characteristics given in figure 1.11b is used. The figure 4.3 gives the reconstruction using the single mode. The same case of training the autoencoder for 5000 epochs was done, but to remove the drawback of previous case, convergence criteria is employed. The data is split into test and training data (or test and training data). This convergence criteria is same as mentioned before in the methodology chapter. Employing the convergence criteria allows for learning to stop at 1800 epochs after which the loss saturates to a certain value and no substantial change is observed.
The quantitative reconstruction for POD and linear autoencoder are $6.8e-2$ and $6.8e-2$ respectively. The figure 4.3a and 4.3b show exactly similar qualitative reconstruction.

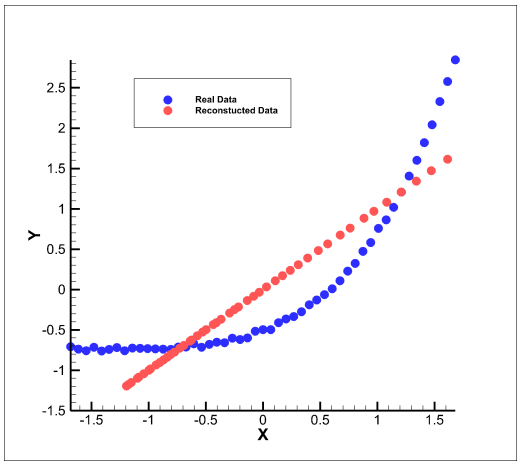
From the two datasets following conclusions can be drawn:

- Based on the results presented for the two different datasets, it can be clearly deduced that a linear autoencoder is nothing but an iterative POD [21] and thus, using linear autoencoder to perform dimensionality reduction is complete waste of computational resources.
- For the case of non-linear data the reconstruction of POD and for the same matter of linear autoencoder is very poor. For autoencoder to be a viable choice, changes to it are must which are presented subsequently.

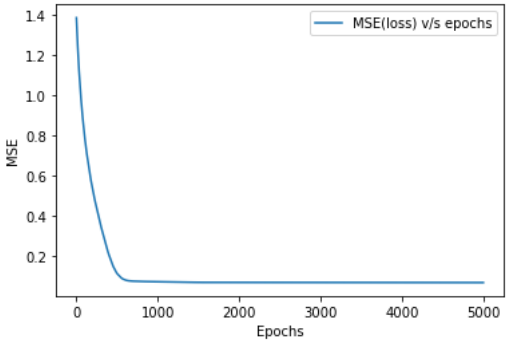
[1] ADAM-Adaptive Moment Estimation. It is an optimized form of the Gradient Descent



(a) POD



(b) Linear AE

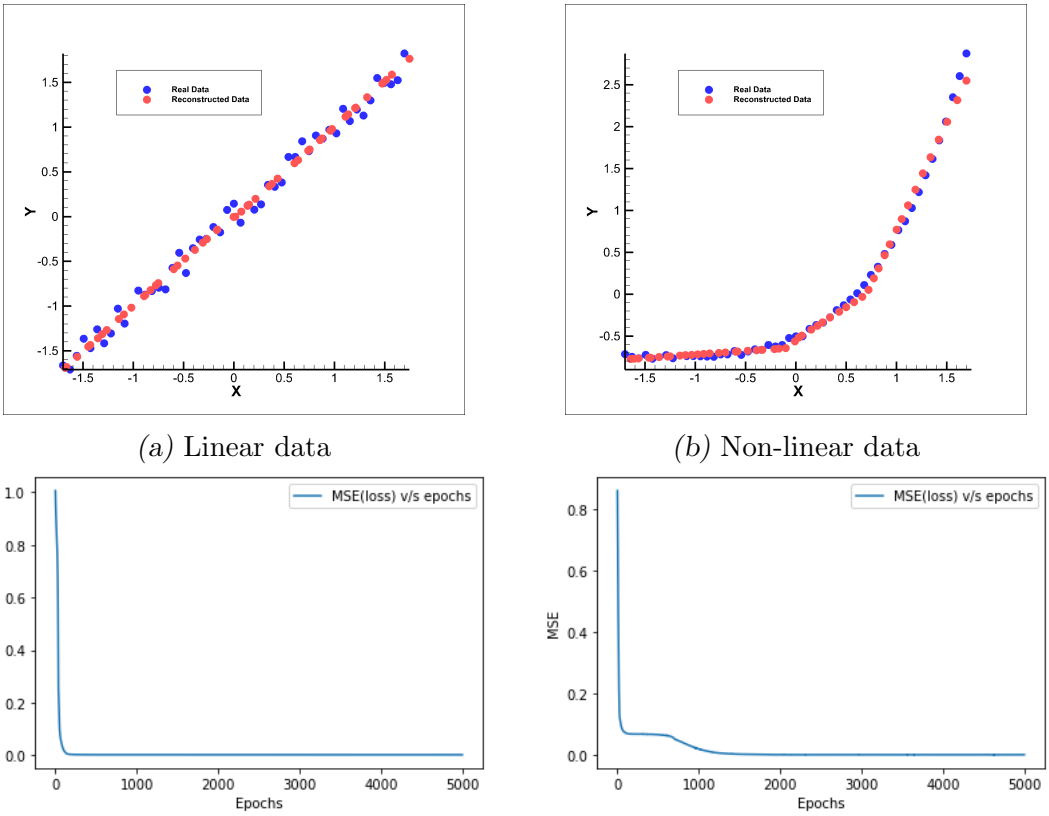


(c) Variation of loss with epochs

Fig. 4.3: Comparison of POD with Linear Autoencoder in (a) and (b)

4.1.2 Non-linear autoencoder

To improve upon the linear autoencoder (iterative POD), activating perceptrons by making use of a non-linear activation functions is the first thing to consider. The sufficient structure required to capture any non-linearity of data is given in figure 2.2. It can be clearly seen that the structure comprises of an additional hidden layer having two nodes per layer. This basic structure is used for both of the previous datasets and results are shown in the figure



(a) Linear data

(b) Non-linear data

(c) Loss curve

(d) Loss curve

Fig. 4.4: Reconstruction for (a)1-D linear data and (b)1-D non-linear data along with the loss curves in (c) and (d) for the linear and non-linear data respectively

A third dataset with higher non-linearity is considered to construct a conclusion. The data is $y = mx^8 + c$. POD, linear AE and the non-linear AE are trained. The reconstruction for non-linear AE is shown in figure 4.5. Also, the reconstruction error for all the dataset considered till now are tabulated in table 4.1 for all three tools.

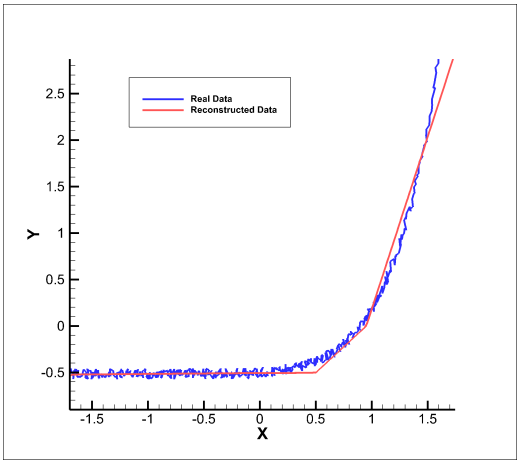


Fig. 4.5: Reconstruction of $y = mx^8$ using the non-linear autoencoder

Tab. 4.1: Comparison between POD, linear AE and non-linear AE

Function	**POD**	**Linear AE**	**Non-linear AE**
	MSE value		
$y = mx + c$	2.2e-3	2.3e-3	2.2e-3
$y = mx^4 + c$	6.6e-2	6.8e-3	8.9e-4
$y = mx^8 + c$	1.9e-1	2.0e-1	2.2e-3

4.1.3 Non-linear AE with 5-hidden layers

Next the optimization of the non-linear autoencoder is considered independent of comparison with the POD. An extra hidden layer is incorporated in the non-linear autoencoder and the parameter considered for optimization for this new hidden layer are the number of nodes in the said layer. First only 2 nodes are used and thereby the optimized value is explored for. The summary of the autoencoder structure thus developed is tabulated in table 4.2.

Tab. 4.2: Summary of the non-linear AE structure

Layer(type)	Output shape	Number of parameters
AE input(input layer)	[(None,2)]	0
Encoder model (model)	(None,1)	15
Decoder model (model)	(None,2)	16
Total parameters = 31		
Trainable parameters = 31		
Non-trainable parameters = 0		

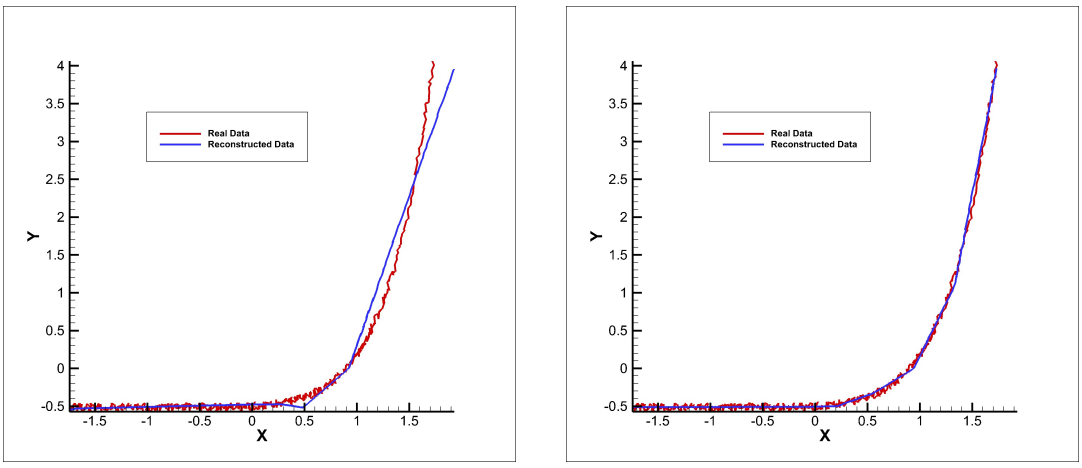
The new hidden layers can have any arbitrary number of nodes(perceptrons). To find out the optimal number of nodes a test analogous to grid in-dependency is done on 2 non-linear datasets shown in figure 4.5 and 1.11d.

The number of nodes optimization is undertaken for the first data set given by equation $y = mx^8 + c$. The optimization test yields the results shown in table 4.3.

Tab. 4.3: MSE value corresponding to the hidden layer size

Dimension of hidden layer	Total trainable parameters	MSE
100	803	4.22E-4
50	403	4.34E-4
10	83	4.85E-4
8	59	4.52E-4
7	51	7.00E-4
5	43	1.10E-3

From the tabulated data we can infer that a hidden layer with 8 nodes is the optimal size for the hidden layer. The figure 4.6 shows the qualitative difference between 2-nodes and 8-nodes in the extra hiiden layers.



(a) 2 nodes *(b)* 8 nodes

Fig. 4.6: 5 hidden layer AE reconstruction

The second dataset considered is an S-curve given by the equation 4.1 and shown in figure 1.11d. As done for the previous dataset, same procedure is applied to this dataset. The results are tabulated and given below.

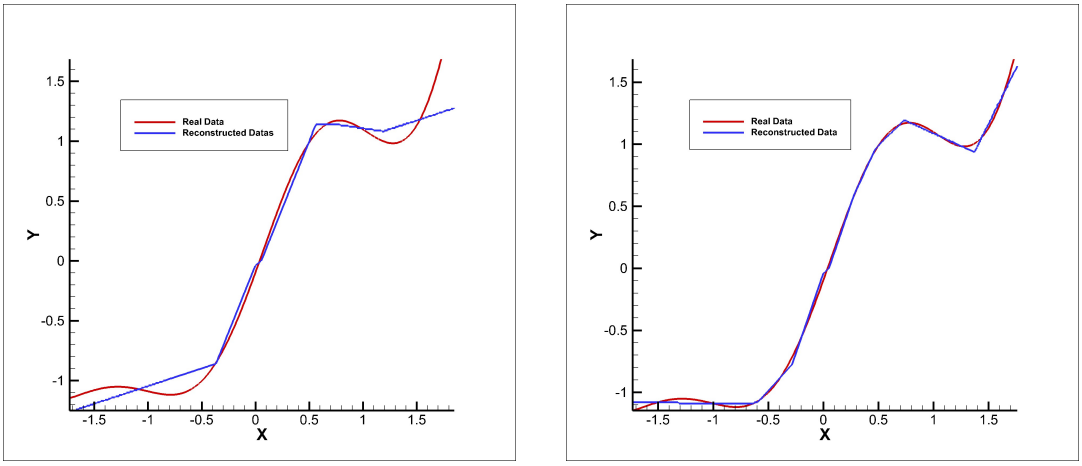
$$0.1x(0.75x + cos(2x)) \tag{4.1}$$

The reconstruction for 2-nodes and the optimal value of nodes taken as 10 is given in figure 4.7.

S

Tab. 4.4: MSE value corresponding to the hidden layer size

Dimension of hidden layer	**Total trainable parameters**	**MSE**
100	803	1.34E-4
50	403	1.80E-4
10	83	3.20E-4
8	59	3.68E-4
7	51	3.40E-2
5	43	3.82E-2



(a) 2 nodes *(b)* 8 nodes

Fig. 4.7: 5 hidden layer AE reconstruction

From the two dataset we can conclude as of yet that selection of right structure of NN is a cumbersome and unrealistic task, especially when realistic dataset are considered. For our case only 1 hidden layer was considered and analyzed. In most cases the NN has a very complex and dense structure and thus the method of brute forcing to find the right structure is not advised. Thus, the decision of choosing a structure of NN is purely intuitive equipped with help from previous works. Still, it can be gathered that similar data have nearly equal optimized parameter.

4.1.4 2-D dataset

From the work done till now it is clear that the hidden layer next to the input can have more nodes than the input nodes, which looks counter intuitive to the concept of autoencoder structure. As a simple and stupid example, imagine an input layer organized as 4 x 4 matrix = 16 nodes. For some reason, the nodes in the hidden layer should detect patterns in the input layer containing straight lines. If learning algorithms are neglected for the moment, and design the hidden layer and

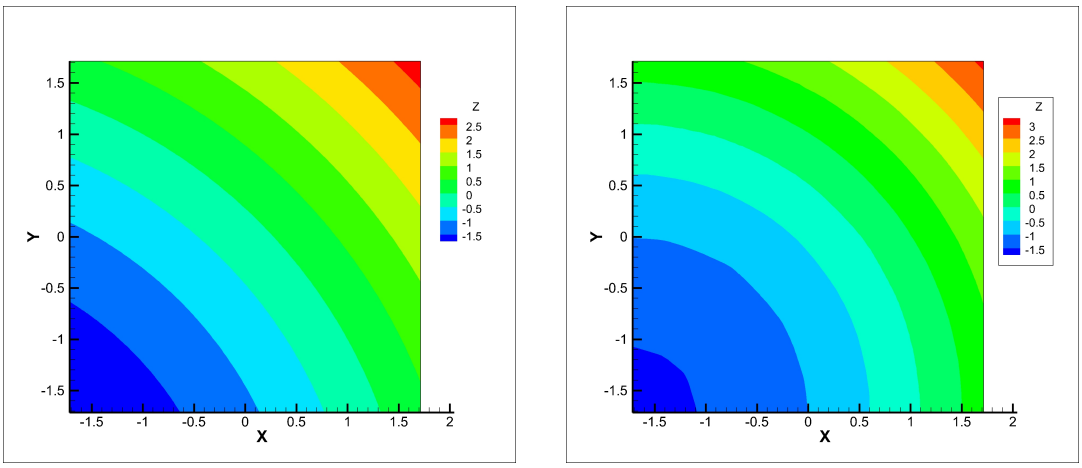
its connections and weights manually, a reasonable approach were to assign one node to each possible straight line. Now there can be 4 horizontal, 4 vertical, and 10 diagonal lines, resulting in 18 nodes in the hidden layer. This simple example answers build the intuition to make use of more nodes in the hidden layer next to the input layer. After this the subsequent layers must decrease in dimension other wise the objective of dimensionality reduction is not achieved.

With the knowledge acquired, the dimension of the input is increased to 2-D. The datasets given in figure 1.12, 1.13 and 1.15 are taken up one-by-one. Bot the POD and autoencoder are trained. For all the three cases the non-linear AE has 5 hidden layers. The layers in encoder part have 100 and 50 nodes, which follows same for the decoder. The latent space has 2 nodes corresponding to the two modes. The structure thus generated for the AE is tabulated below:

Tab. 4.5: Summary of the non-linear AE structure for 2-D

Layer(type)	Output shape	Number of parameters
AE input(input layer)	[(None,3)]	0
Encoder model (model)	(None,2)	5552
Decoder model (model)	(None,3)	5553
Total parameters = 11,105		
Trainable parameters = 11,105		
Non-trainable parameters = 0		

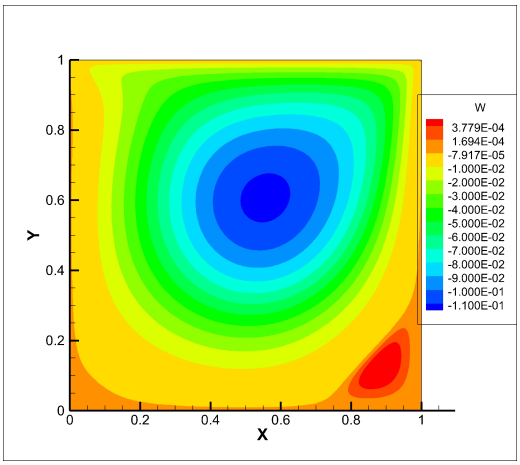
The first 2-D arbitrary data reconstruction using POD and autoencoder with 2 modes out of the 3 available modes gave the MSE or reconstruction value of 1.62e-2 and 2.86e-5 respectively. The same is represented in figure 4.8



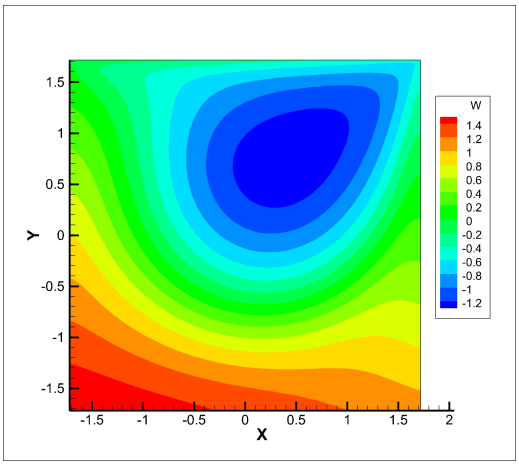
(a) POD *(b)* Non-linear AE

Fig. 4.8: Reconstruction using POD and non-linear AE

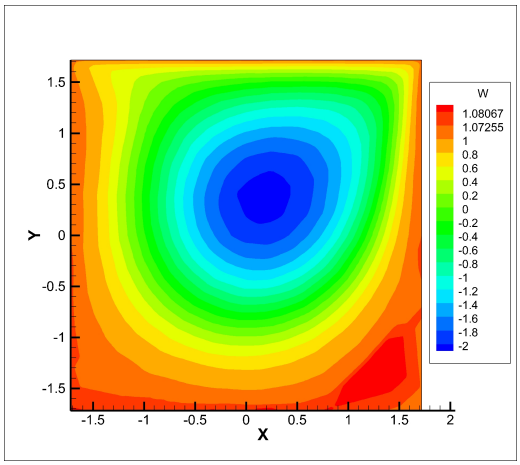
For the case of lid-driven flow the reconstruction value obtained for POD and autoencoder are 2.91e-1 and 6.25e-5 respectively, which again is given in figure 4.9



(a) Original data



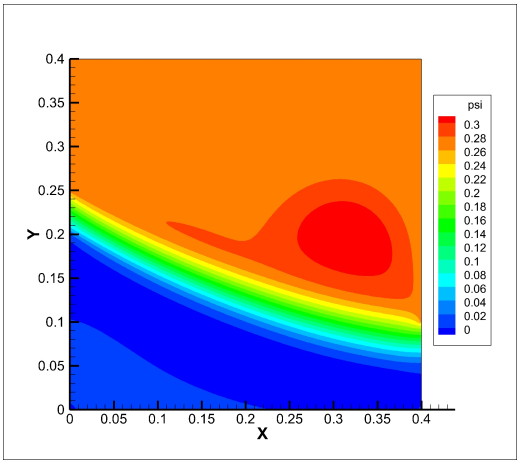
(b) POD



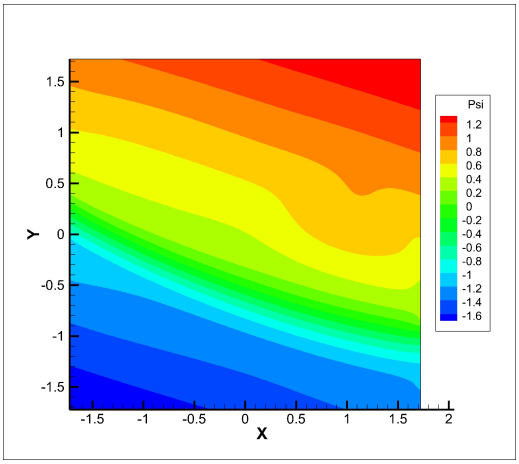
(c) Non-linear AE

Fig. 4.9: Reconstruction using POD and non-linear AE for 2D lid driven flow

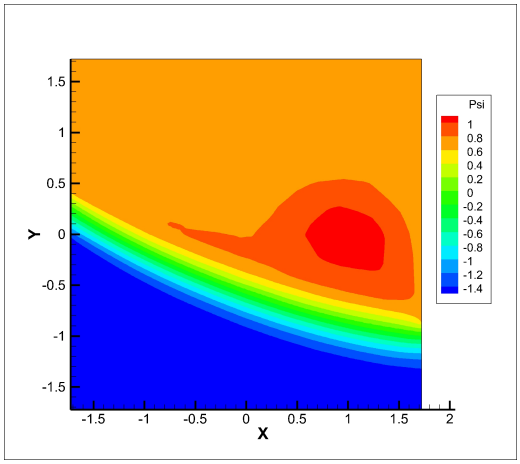
For the room flow or inlet outlet flow problem the MSE values for POD and autoencoder are 8.30e-2 and 5.06e-5 respectively, the qualitative reconstruction of which is given in figure 4.10



(a) Original data



(b) POD



(c) Non-linear AE

Fig. 4.10: Reconstruction using POD and non-linear AE for 2D inlet-outlet flow

Much evidence is now available for safely saying that a non-linear autoencoder has superior reconstruction efficiency in comparison to the POD. Unless, the data has high degree of linearity inherent to it, the POD is bad choice. The real flow data for which ROM is done though has numerous modes available, so the choice of POD over autoencoder can only be made under one circumstance, which is when cost of mode computation has more weightage. For this a feasibility analysis must be done to ensure the best ROM possible.

4.2 Objective III and IV: C-CNN for time series fluid flow data

The data given for autoencoder till now was steady state and thus, the modes were vectors indicating the direction of maximum variation. The use of dimensionality reduction is to extract spatial modes that capture the maximum variation of the data over the time space. This means that the input vector is now the values at all the grid points. Typically, the avgerage grid size for even a very simple flow is 100×100. This makes the input vector of dimension 10,000 at the very least in place of 2 and 3 vector size used till now for the steady state data. And as mentioned earlier, for efficient feature capturing the first hidden layer can have more nodes than that of the input layer. As an example taking number of nodes same as the input vector size, the number of trainable parameters just for the first layer are of the order of 10^8. And this is just for the encoder part. The same structure the decoder has adding another 10^8 trainable parameters. Still, for the structure to capture the non-linearity, the NN must have 3 hidden layers. All-in-all the dense autoencoder becomes impractical even just for simplest of flows. On top of that, the dense autoencoder is very in feature extraction.

4.2.1 Extraction of zone

The solution to this problem is using conventional -convolutional nerual network. A C-CNN-AE is used for the VIV flow data to give conclusion to the objectives. The first objective was faced and achieved as a result of need. The VIV data shown in figure 1.16 on page 21 has a grid size of 375×305. The raw data is not fit for CNN as CNN employs max pooling to reduce the dimension of the data and certain factor number is a must. To remedy this one can add zeros to data necessary to make the data of appropriate dimension.This adding of zeros to conform the grid size as desired by CNN is called padding, and since the zeros are being added, it is called zero padding. In the structure chosen max pooling of halving the original dimension was used. To conform to this max-pooling, the nearest grid size available is 384×320 which upon max pooling yields the smallest grid size to be 6×5. With the subsequent dense layer the final latent space dimension is achieved to be 2 (out of 180). The number of samples available for training is 180, making as many modes available for understanding the data. The training is done and only the qualitative results are presented in figure 4.11.

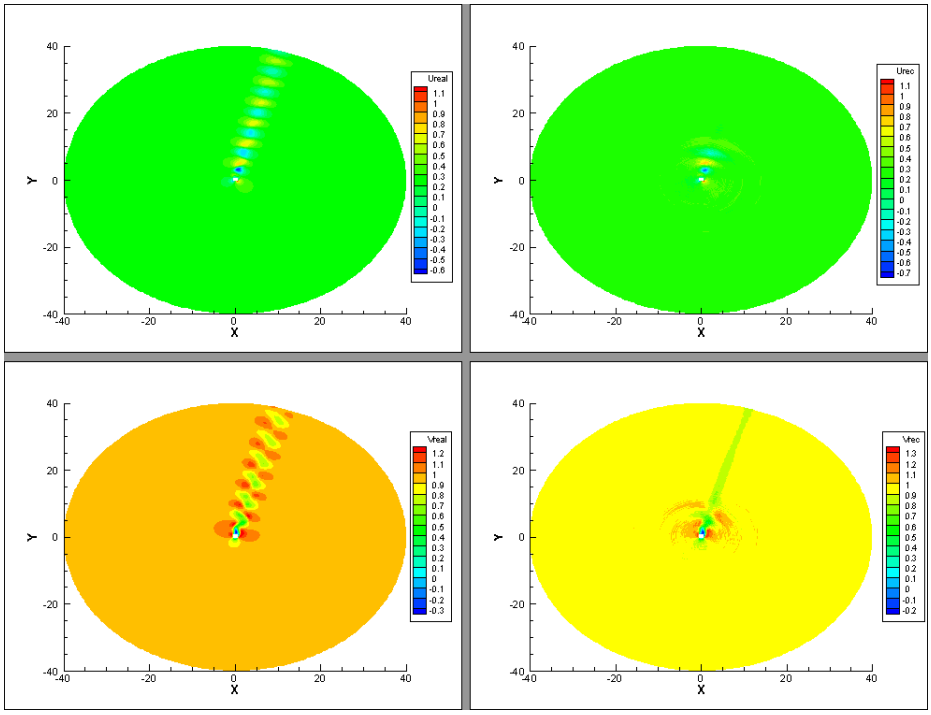


Fig. 4.11: From top left to right is the U velocity contour real and reconstructed respectively, similar case id for V velocity contour

The figure clearly shows that the training is very poor with MSE value of 1.68e-3. To overcome this one can extract the data containing the crux of the data. This extraction of part of the zone meets our objective of zone extraction.

4.2.2 Activation function selection

There are following nine activation functions in keras API for perusal.

1. Linear
2. Softmax
3. Softplus
4. Softsign
5. Sigmoid
6. Elu
7. Selu
8. Relu
9. Tanh
10. Exponential

For each of these activation functions the structure of C-CNN used is given in table 2.4 and 2.5. The training was done on a sample size of 180. Of these 180 snaps 10 were assigned for the the validation set. The training was done for 2000 epochs with a learning rate of 1e-3. The early stopping criteria was kept as 50 iterations with no change in validation loss. The table 4.6 gives the loss and the

respective epochs it took to reach it for each of the activation functions mentioned above.

Tab. 4.6: MSE value and number of epochs for different activation functions

Activation function	Epochs	MSE	**Activation function**	Epochs	MSE
Linear	829	5.22E-4	Elu	1085	1.80E-4
Softmax	127	1.23E-2	Selu	633	2.47E-4
Softplus	308	9.56E-4	Relu	411	7.26E-4
Softsign	838	2.77E-4	Tanh	2000	1.74E-4
Sigmoid	313	1.23E-2	Exponential	-	NaN

For each of these activation functions the vorticity magnitude, U and V velocity contour with their respective L2 norm are given in from figures. Each property is presented for all activation functions for the sake of easy comparison.
The loss with respect to the epochs for each activation functions is shown in figure 4.12.The reconstruction is shown qualitatively for each of the activation functions (except for the exponential which blew up upon learning) in the figures 4.13-4.15. The conclusion are drawn subsequently.

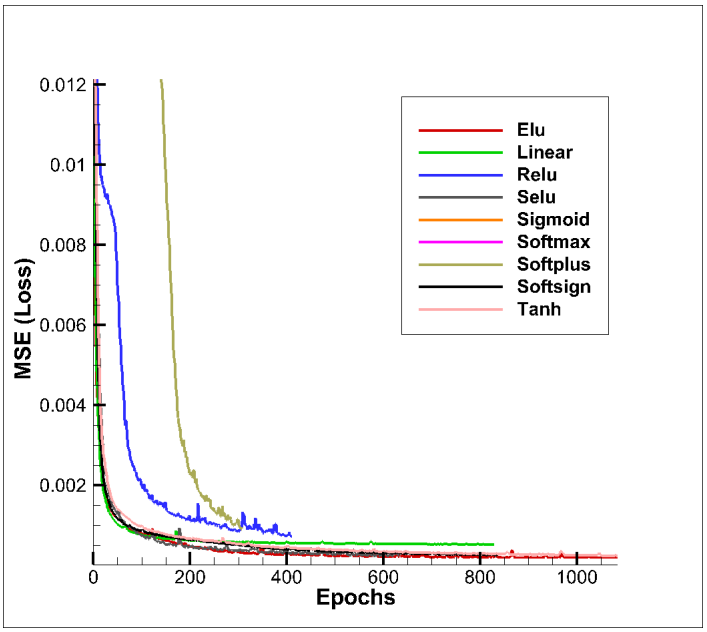


Fig. 4.12: Loss vs Epochs

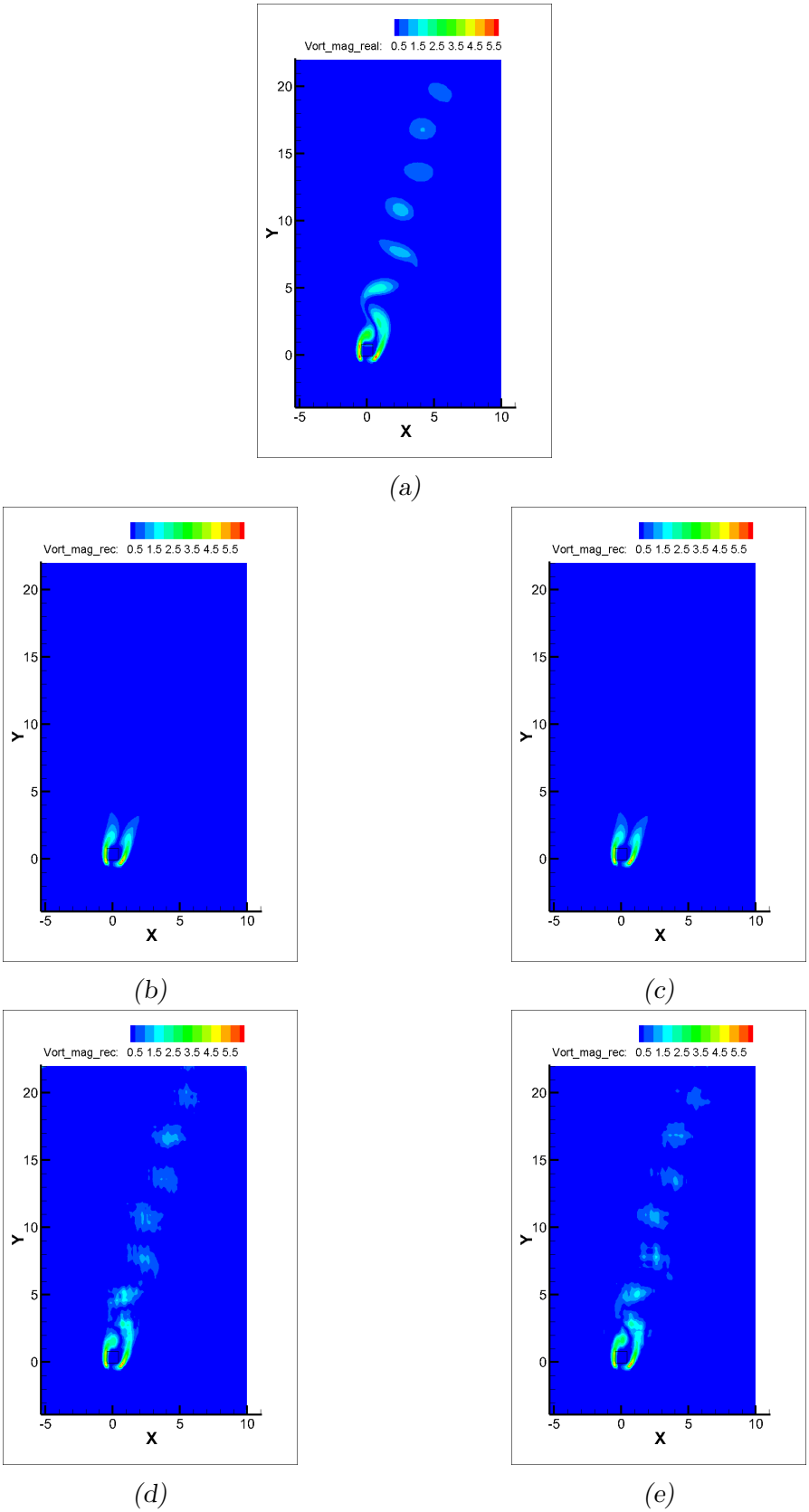
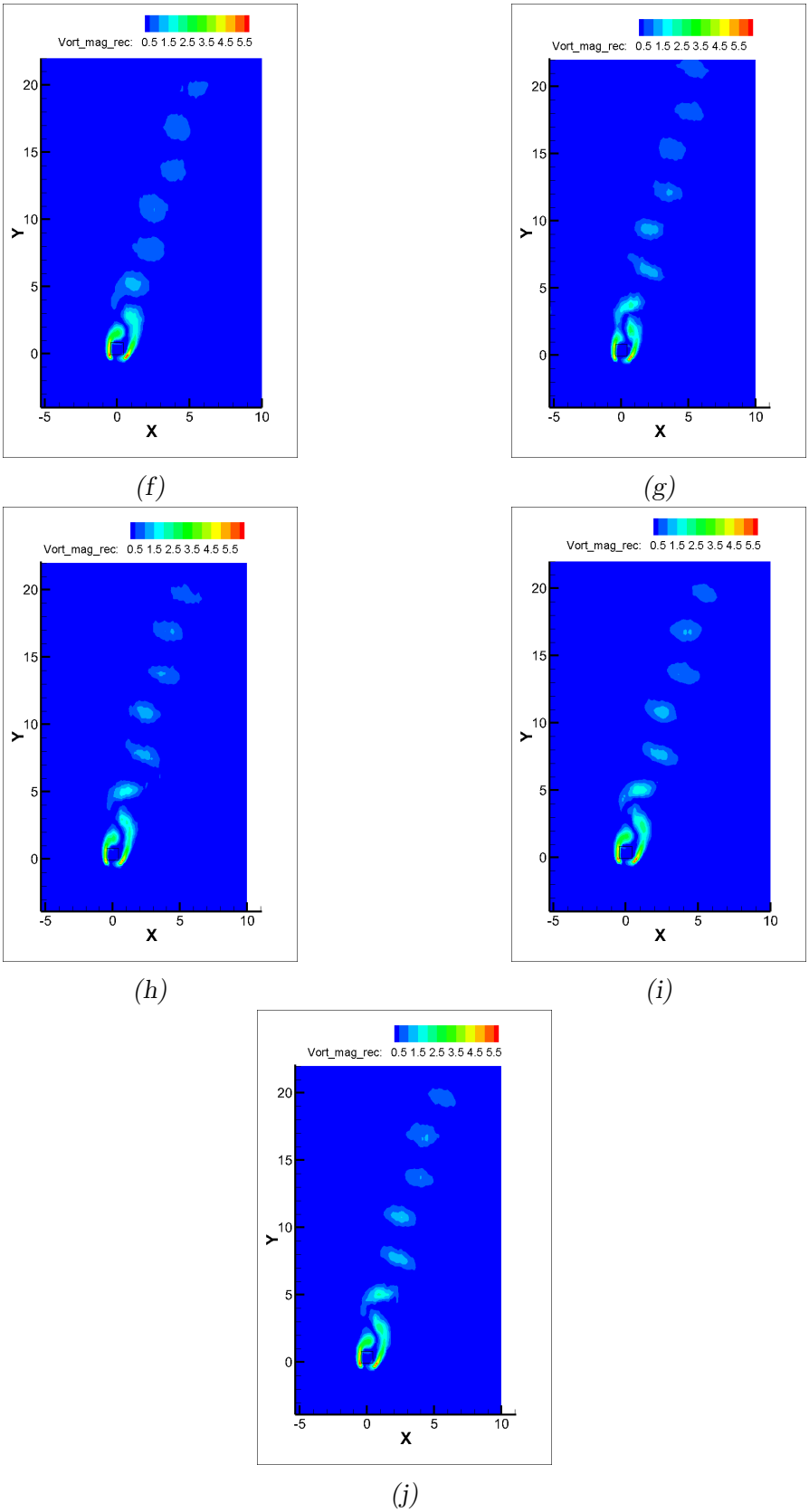


Fig. 4.13: Vorticity magnitude for (a)Reference DNS (b)Sigmoid (c)Softmax (d)Softplus (e) ReLU



(f) (g)

(h) (i)

(j)

Fig. 4.13: cont... Vorticity magnitude for (f)Linear (g)Softsign (h)Selu (i)Elu (e) Tanh

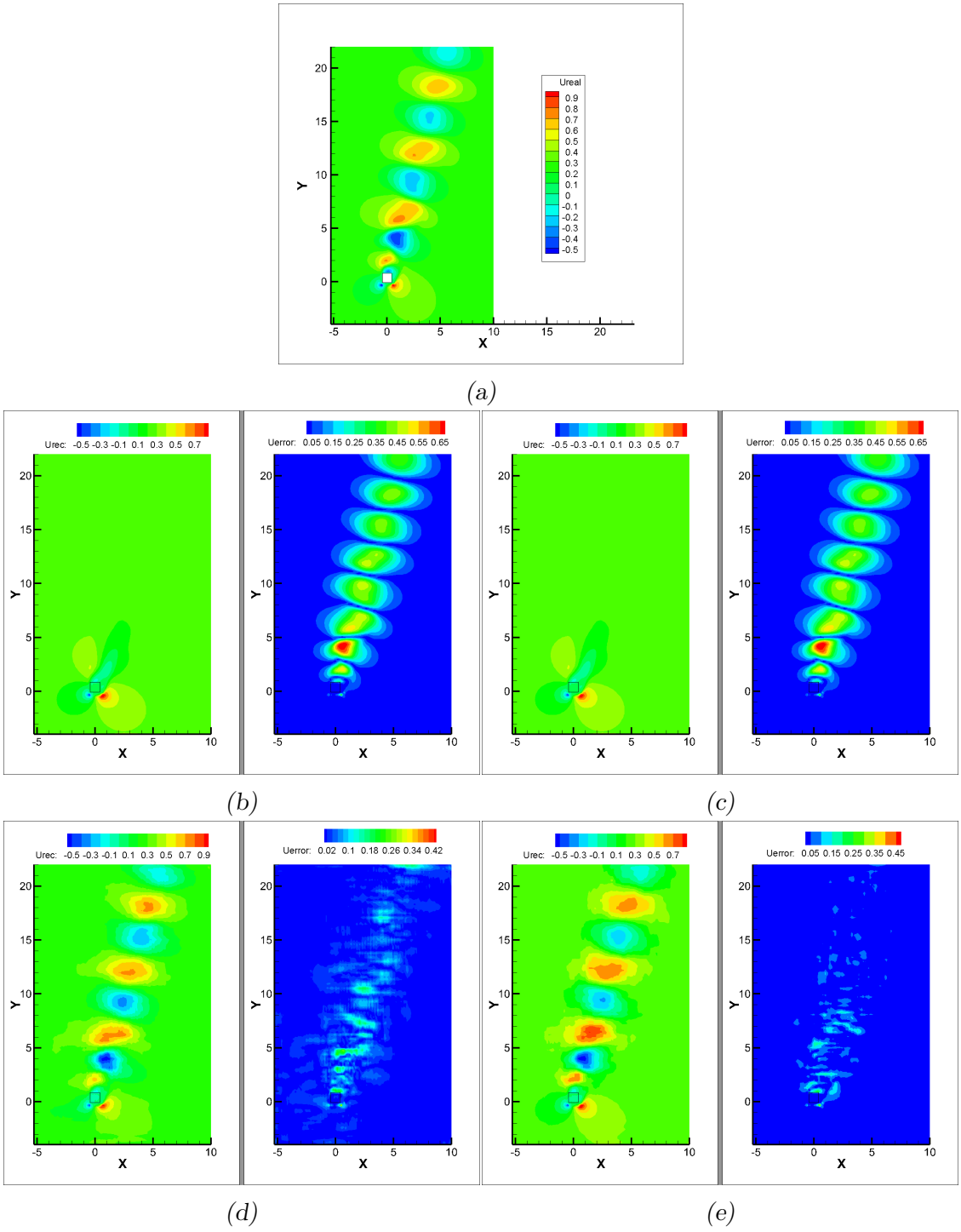


Fig. 4.14: U velocity and it L2 norm with original U for (a)Reference DNS (b)Sigmoid (c)Softmax (d)Softplus (e)ReLU

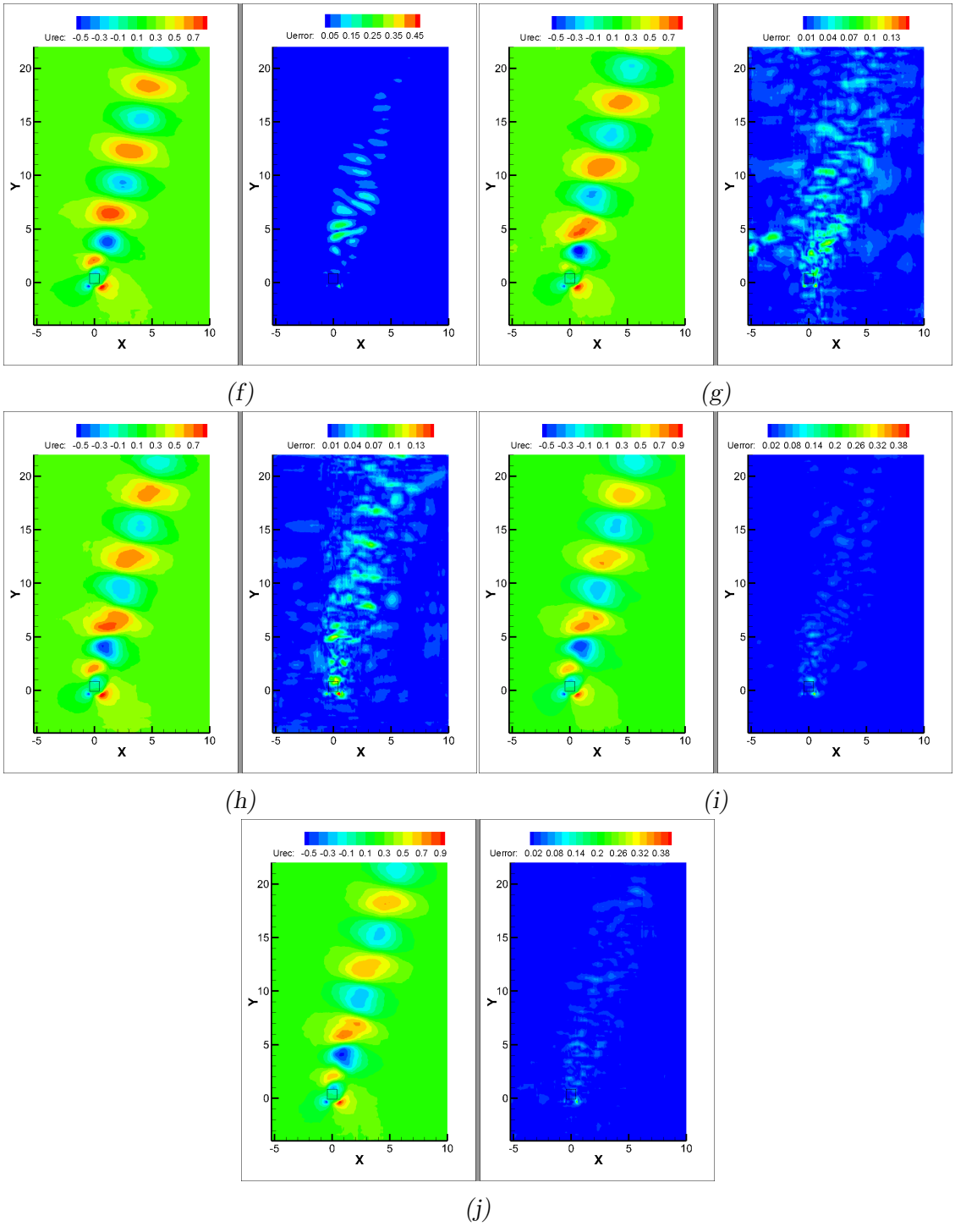


Fig. 4.14: cont... U velocity and it L2 norm with original U for (f)Linear (g)Softsign (h)Selu (i)Elu (e) Tanh

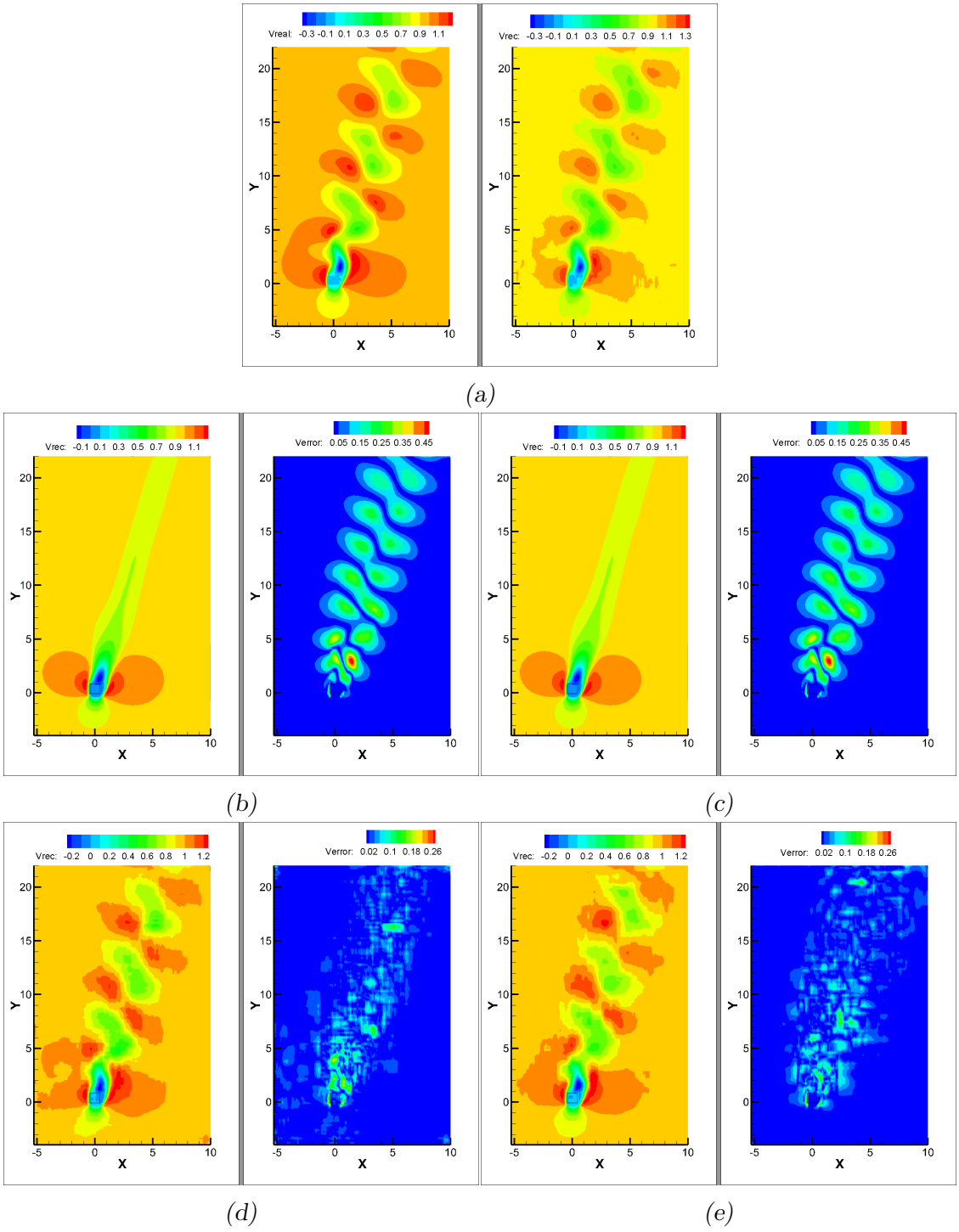


Fig. 4.15: V velocity and it L2 norm with original V for (a)Reference DNS(b)Sigmoid (c)Softmax (d)Softplus (e)ReLU

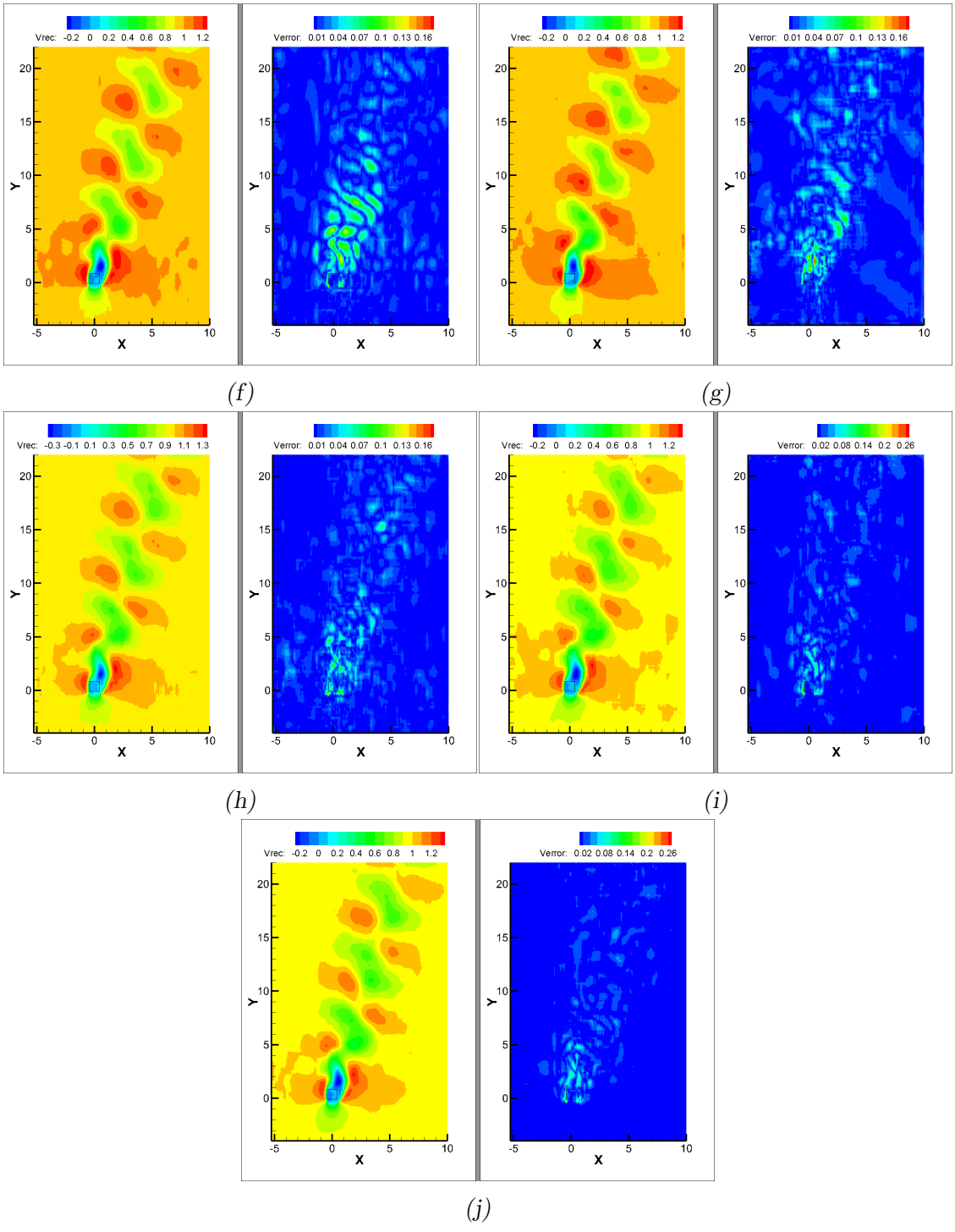
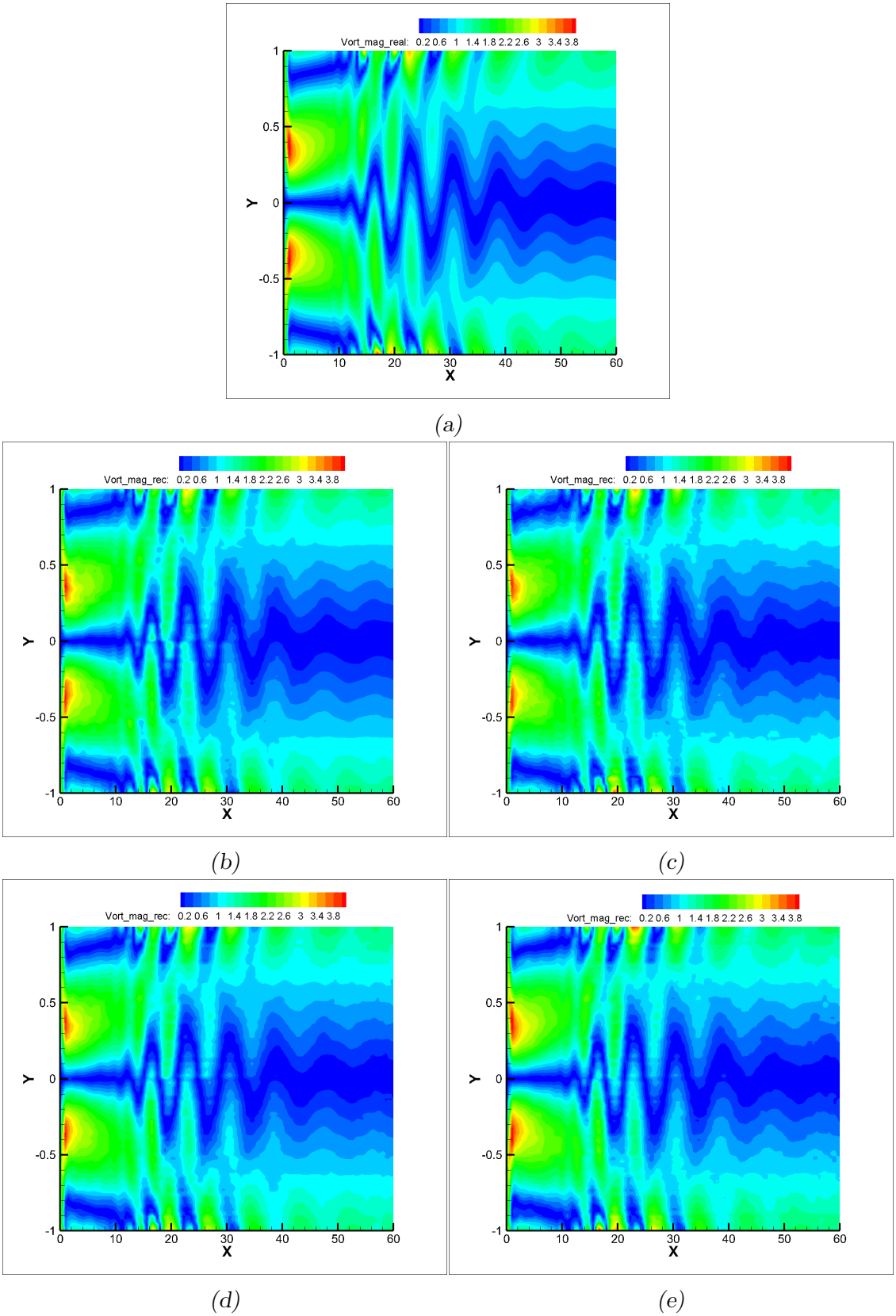


Fig. 4.15: cont... V velocity and it L2 norm with original V for (f)Linear (g)Softsign (h)Selu (i)Elu (e) Tanh

From the tabulated data and the figures, following conclusions can be drawn

- Based on the MSE value the worst performing activation functions are sigmoid and softmax. The sigmoid performs poorly because of its gradient vanishing problem and the fact that it gives probability as output. The softmax being one of its crude modification faces the same problem. The fluid data is not probabilistic in nature which is why the two functions fail. The two functions perform very well for classification problems
- On the same argument of MSE value both softplus and relu can be firmly rejected relative to best performing activation functions. ReLU was able to give better result as it gives linear output, but the lack of no output for negative input value hampers it produce good result. Softplus being the smooth approximation of relu performs worse than relu as the same thing that is used to improve upon relu backfires by giving very large/exploding output values.
- The linear activation proves better than relu as what relu lacks in terms of no output for negative input values, linear function fulfills it. Still, the MSE value and the qualitative reconstruction support its activation function.
- The four remaining activation functions, tenh, selu ,elu and softsign are all viable candidates as per the MSE value. Based on the figure of V velocity L2 norm, the softsign is on the border of being discarded owing to large error values. Still, to maintain generality further investigation is required to reach some solid conclusion.

To clear the ambiguity of right selection of activation function among the remaining viable candidates of activation functions being tanh, selu, elu and softsign, another periodic flow data given in figure 1.18 is trained. The data for this case also had to be extracted from the full raw data confirming the zone extraction objective. The extracted zone has a grid size same as used in VIV of 128X128 for which the same CNN structure is used given in table 2.4 and 2.5. The training is done in the same as done for VIV flow, just for the remaining activation functions. The results are presented in figures



(a)

(b)

(c)

(d)

(e)

Fig. 4.16: Vorticity magnitude (a)Original data (b)Softsign (c)Selu (d)Elu (e)Tanh

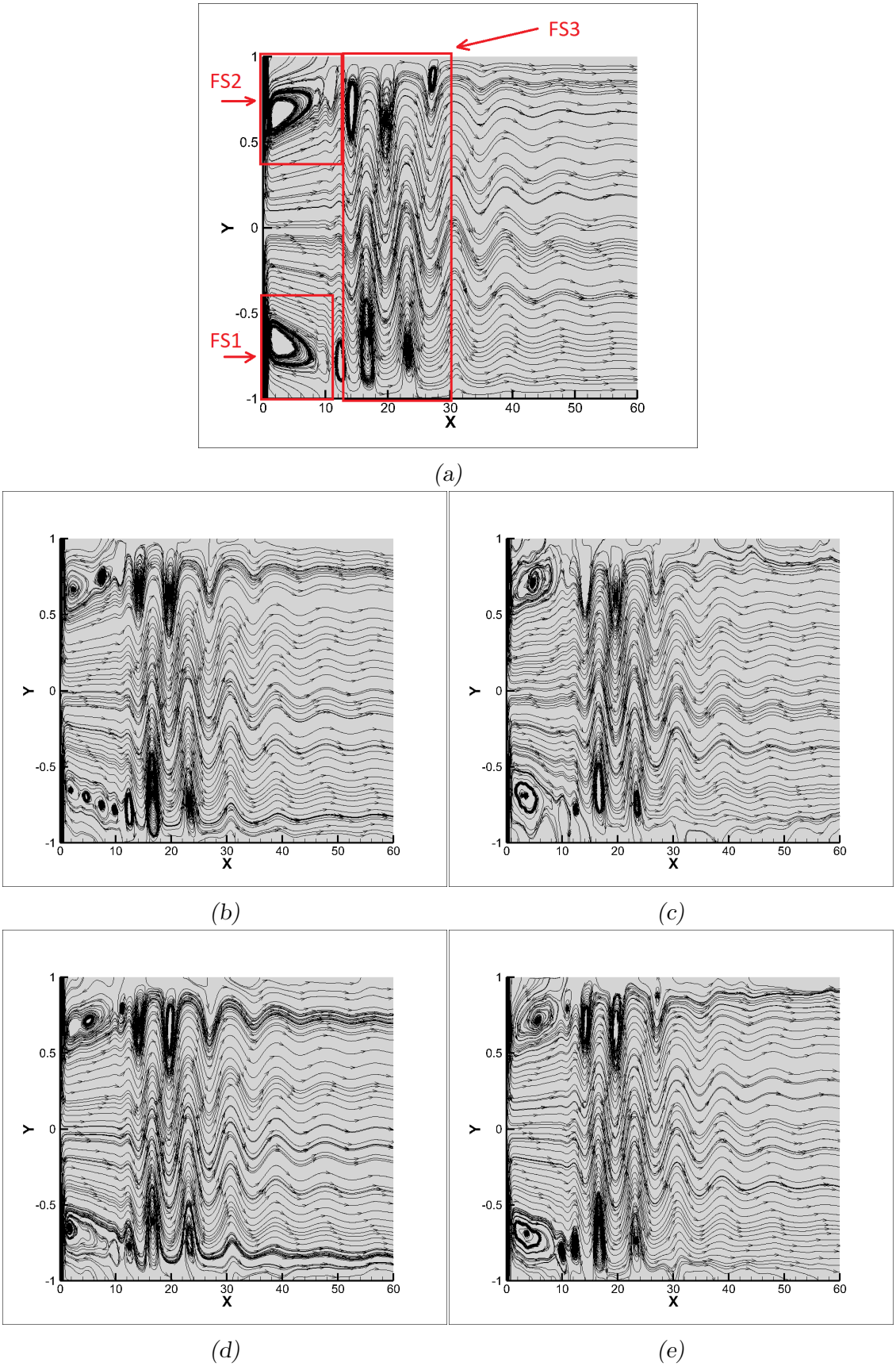


Fig. 4.17: Streamtraces for (a)Reference DNS with important flow features marked (b)Softsign (c)Selu (d)Elu (e)Tanh

The table 4.7 gives the error for the four activation functions. The tabulated data is still not enough to reach at some conclusion.

Tab. 4.7: MSE value and number of epochs for remaining four activation functions

Activation function	**Epochs**	**MSE**
Elu	972	3.37E-5
Selu	1225	5.53E-5
Softsign	1168	3.24E-5
Tanh	1086	3.85E-5

For closer investigation the main flow structures are marked in the streamtrace and presence of them are noted in table corresponding to each of the activation functions.

- FS1: Primary vortex I
- FS2: Primary vortex II
- FS3: Primary flow structure

Tab. 4.8: Important flow structures reconstruction comparison

Activation function	**FS1**	**FS2**	**FS3**
Softsign	×	×	✓
Selu	×	✓	×
Elu	✓	✓	✓
Tanh	✓	✓	✓

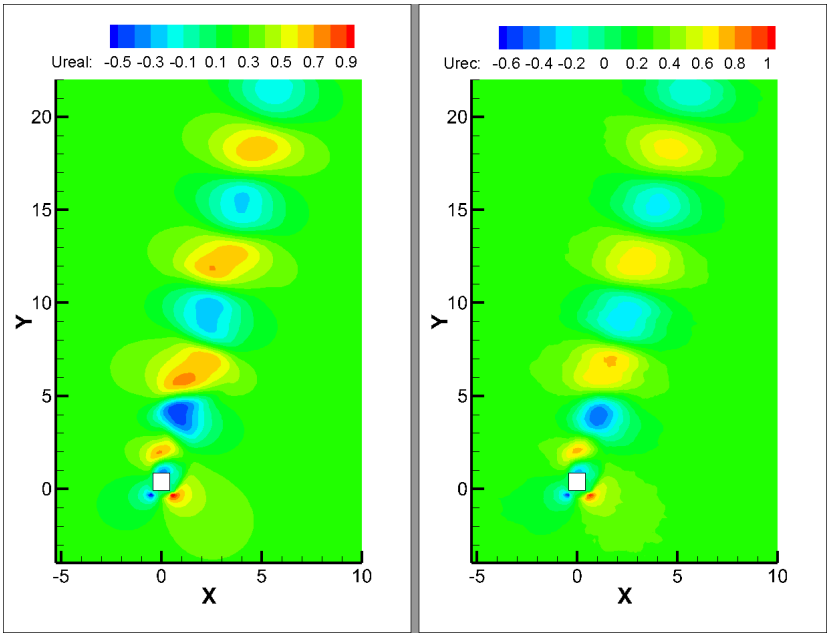
From the data presented so-far for both periodic flows, softsign and selu can be dropped as viable activation function candidate for fluid flow data. Since no such conclusive evidence is found for the Elu and Tanh activation functions, therefore, these both remain a choice for activation function. Although, some would argue that Elu's reconstruction of FS1 is poor, that falls under ones own subjectivity.

4.3 Obejctive V:Mode decomposition of VIV flow

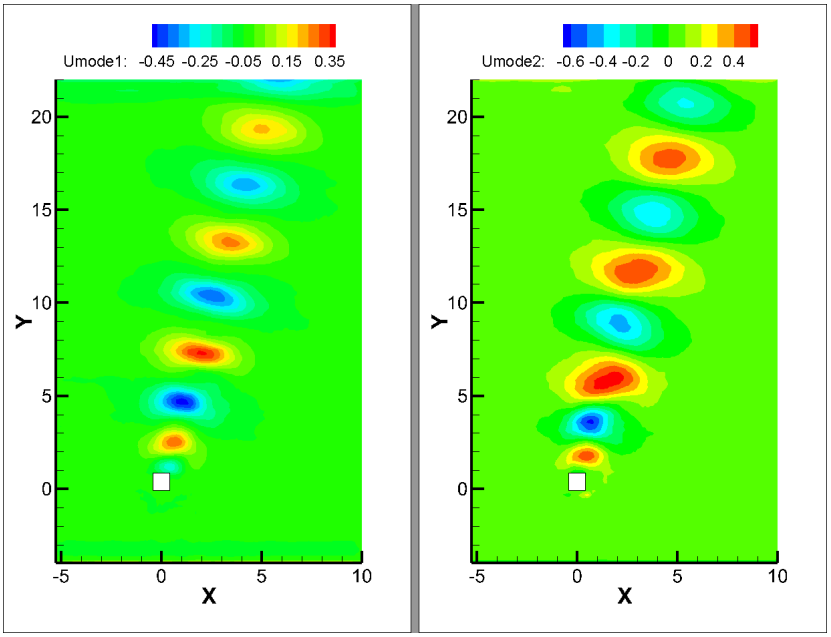
All the work done till is culmination of tool development of efficiently compressing the given data using the AI. The next step which is of real importance is to extract modes of the flow and visualize them. The method developed my Nurata et. al. of MD-CNN [25] will be used to extract the modes of the VIV flow problem. MD-CNN is the modified form of the conventional CNN. What C-CNN lacks in extraction of modes, MD-CNN comes to the rescue of it. All the work done till now helps to work with the optimized MD-CNN, atleast on some parameters that defines the neural network.

The training was done using tanh activation function which yielded an MSE value of 2.29E-4. The MSE value is greater than the C-CNN counterpart, clearly indicating the complexity of the MD-CNN structure and also in conformance with the results of Murata et. al.

The figures 4.18-4.23 show two different reference DNS and the corresponding modes obtained. A naive discussion that can bloom from this is that the first modes for both U and V velocity reference DNS 1 and 2 are nearly same (the difference attributed to computational errors), while the 2nd modes are different clearly suggesting use of different modes from the available 180 modes. This clearly showcases the drawback of using the MD-CNN to extract modes. Although, the modes can be extracted and visualized as can be done with POD, unlike the POD these modes are not arranged in the ascending order of energy content. This calls for further development of the current tool to a hierarchical mode decomposition.

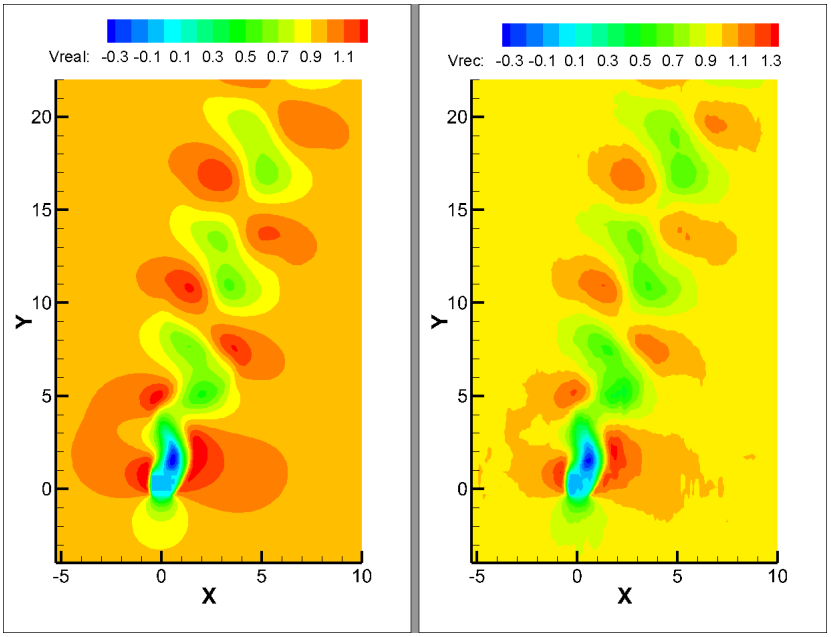


(a) Reference DNS-1 and its reconstructed counterpart

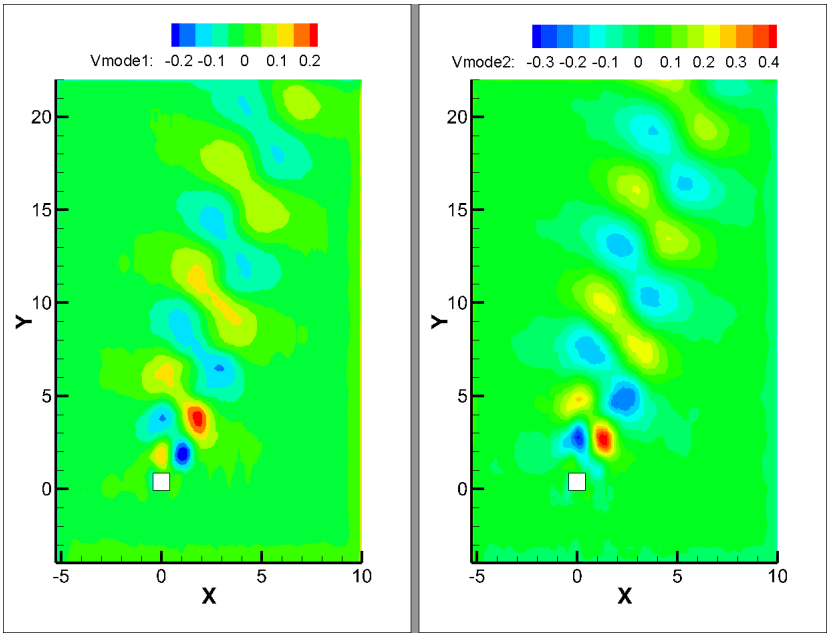


(b) The 2 modes for the given reference DNS-1

Fig. 4.18: (a) U velocity DNS-1 and its reconstruction (b) the two modes extracted using MD-CNN



(a) Reference DNS-1 and its reconstructed counterpart



(b) The 2 modes for the given reference DNS-1

Fig. 4.19: (a) V velocity DNS-1 and its reconstruction (b) the two modes extracted using MD-CNN

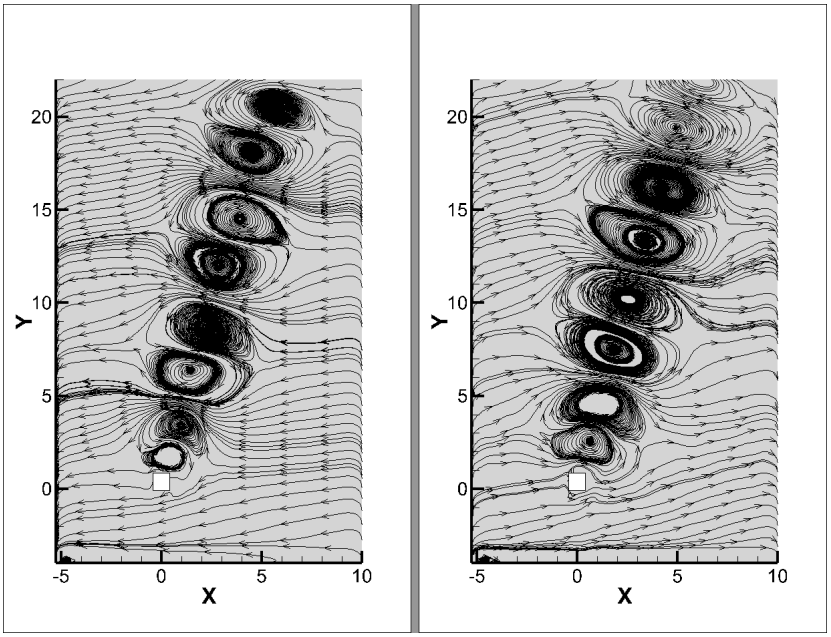
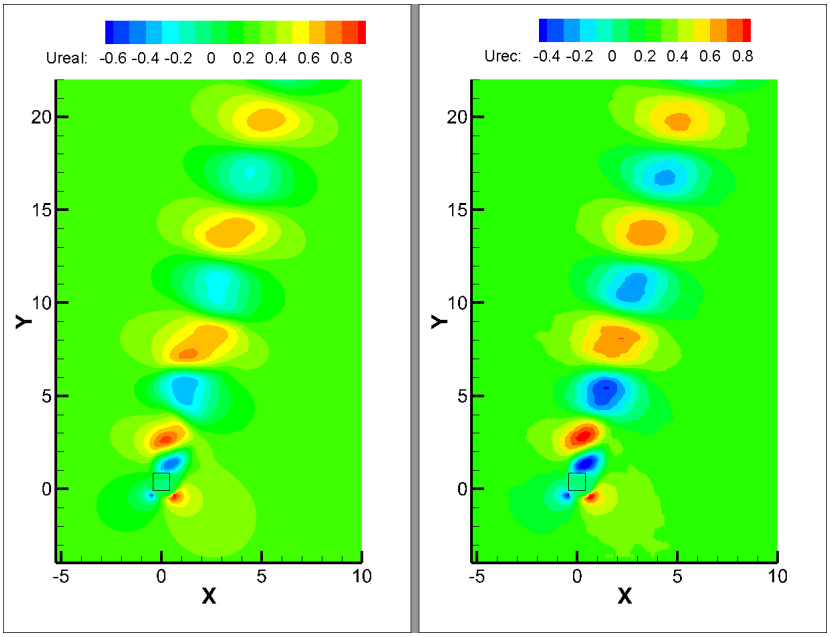
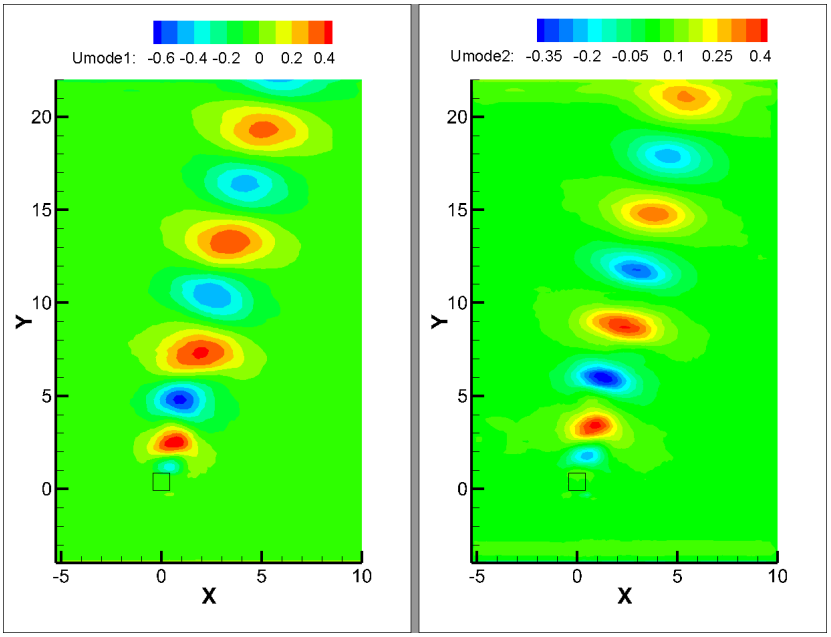


Fig. 4.20: Streamtraces using the two modes obtained

The streamtraces for the two modes are shown in figure 4.20. From the obtained modes in figure 4.19b not much information can be extracted. The amount of information contributed towards the flow reconstruction is unknown, although since the mode number are few one can make an intelligent guess that the first mode (just to be clear here that the talk is about the V velocity modes) contributes more based on the review of few values. But, this method of drawing conclusions is un-scientific and thus, all the prepositions stand null and void unless some solid facts are put forward.

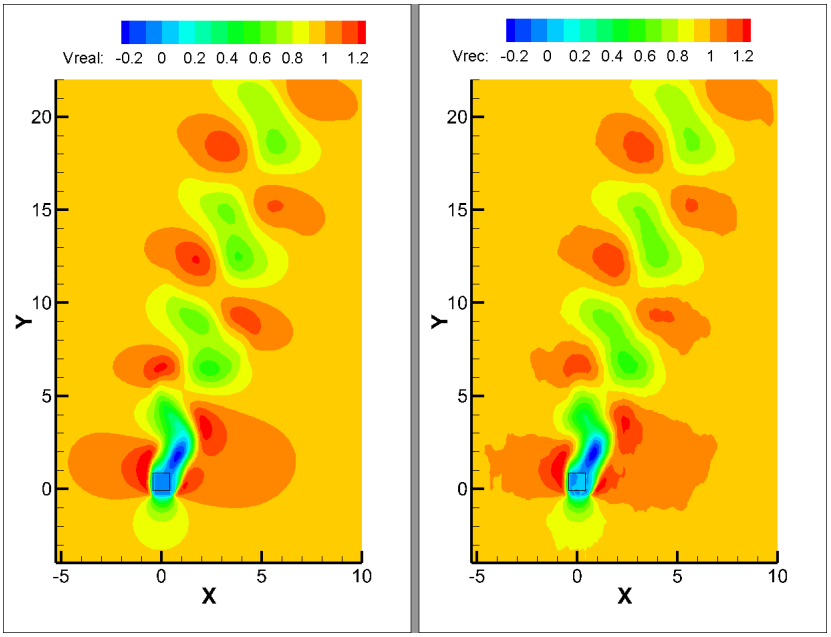


(a) Reference DNS-2 and its reconstructed counterpart

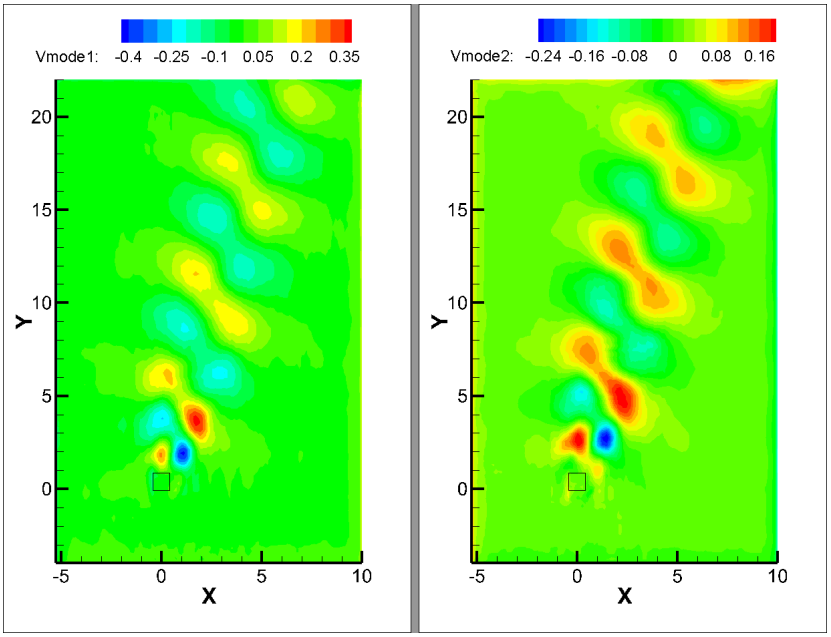


(b) The 2 modes for the given reference DNS-2

Fig. 4.21: (a)U velocity DNS-2 and its reconstruction (b) the two modes extracted using MD-CNN



(a) Reference DNS-2 and its reconstructed counterpart



(b) The 2 modes for the given reference DNS-2

Fig. 4.22: (a) V velocity DNS-2 and its reconstruction (b) the two modes extracted using MD-CNN

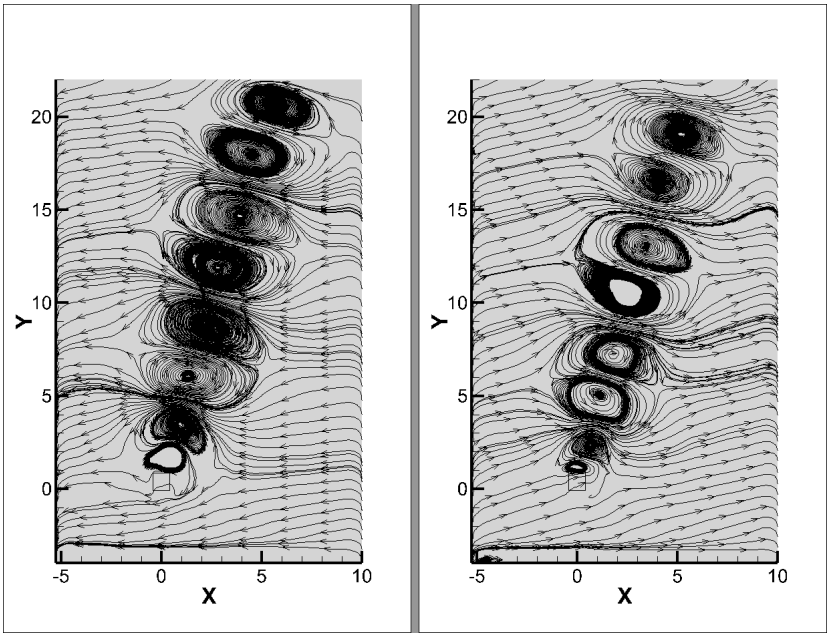


Fig. 4.23: Streamtraces using the two modes obtained

The streamtraces for the two modes are shown in figure 4.23. From the modes shown for two different reference DNS one gets the intuition of the first mode being the dominant mode as it is nearly the same for the two. But, still the basis for conclusion is not sound and scientific and thus, one must refrains from making such remarks.

4.4 Objective VI: Quasi-periodic flow

This section is complimentary as the result or rather the data given here doesn't have any practicality, moreover the so-called "result" is poor. Still, it is presented as some naive conclusion can be drawn.

The training data is a quasi-periodic flow is considered, presenting much more complexity for the AI model to learn. It is to be noted that same 180 snaps were given for training of the AI. The data as followed from the result of zone extraction, is extracted from the raw data with grid size of 256×256.The C-CNN used is same as the one used all this while with only an extra Conv2D and MaxPooling layer. Eventually after training, though, the same result was unfortunately not obtained for the present case. The figure 4.24 show the qualitative reconstruction for a reference DNS. The reconstruction is for using 20 modes or 20 perceptrons in latent space.

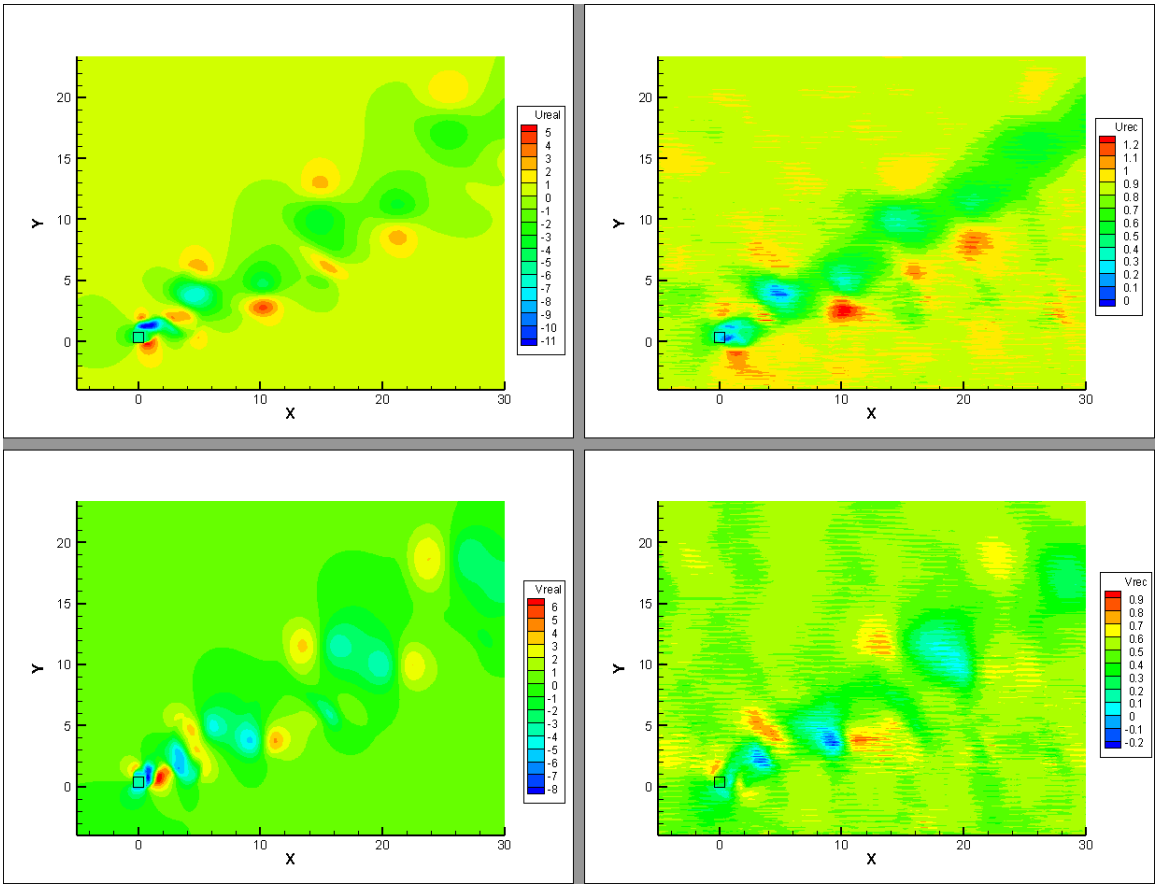


Fig. 4.24: From top left to right is the U velocity real and reconstructed contour respectively while the bottom is for the V velocity contours

From the figure it can be garnered that learning is very poor with a MSE value of 6.33E-3. Although, the model tried to learn, the learning still remains poor. The table4.9 presents the MSE value when using different number of modes to find any connection

Tab. 4.9: Reconstruction error using different number of modes or latent space dimension

Number of modes	Epochs	MSE (Loss)
2	1851	3.58E-1
10	664	7.99E-3
20	896	6.33E-3
100	392	6.00E-3

A naive interpretation and cause of the poor learning can be attributed to the fact that a very small sample size was given for training in the first place. Thus, the first solution that can be thought of is training the model with more snaps. A preferred number of snaps would be more than 1000 based on intuition.

From the above discussion, a naive yet intuitive conclusion can be drawn that a large sample size would provide an even better learning for the VIV periodic flow case, which follows from the works of Murata et. al. that used 2000 snaps for the purpose of training with an input data very similar in characteristics to the periodic flow and obtained an MSE value 10 order better. This hypothesis needs a rigorous work for it to be generally accepted.

5. CONCLUSION AND FUTURE SCOPE OF WORK

The results are concluded and thus the conclusions are itemized below:

1. Use of AI tool - the autoencoder, it can be observed that it is the future of reduced order modeling. Other than when the input data has extremely inherent linear characteristics, the **autoencoder proves to be far more efficient than the conventional tool of POD**. And since, the fluid flow data even in its simplest has non-linearity inherent to it, it is safe to say that an autoencoder should be the first choice unless the computational cost weighs more than the accuracy of the reduced order model. This rejection of AE for the computational cost comes from the fact that an AE is an iterative method while the POD is method of single step.

2. The second observation is a fruit of need. Using a dense autoencoder preliminary lead to a major issue of cumbersomeness and complexity. The number of trainable parameters were too huge for the computation, also the inefficiency to extract features couldn't justify the computational cost. To remedy this issue, the conventional type convolutional neural network is used which is much better in extracting relevant features.

3. With the options of myriad of activation functions, it becomes a hefty task to decide which among them to select. Like mentioned, the Keras API has nine activation functions available for use. For the case of fluid flow data, the two best activation functions that emerged from the analysis are the Elu and the Tanh. Although, not conclusively but the recommendation that can be made is to **use Tanh as the primary activation function**.

4. One other thing that can be concluded is the extraction of zone when dealing with DNS data that has a very large domain and the crux of the flow lies in a very region of that zone. It is thus observed and proven that **extracting the part of the zone** that contains that crux be extracted and used as input for the training purpose.

5. The modes were easily extracted by the use of MD-CNN given by Murata et. al. [25]. Although, these modes were more efficient in compressing the data, they lacked the power possessed by the POD modes- like the energy order and contribution of the modes to the flow. Subsequently, much work has to be done on that.

6. As is the case for AI and machine learning, a large dataset would make for better learning. So, its preferred that a large data set is used for training the AI model.

The conclusions presented above have recommendations that can be offered for future utilization. Thus, the aim of the thesis has been achieved to some degree. The amount of parameters and their combination present for any autoencoder is staggering and cannot be all analyzed, thus the level of degree of achievement.

As concluded, the modes obtained using the MD-CNN are still primitive and lack very useful information like energy content and the order in energy in which the modes are obtained. For this the yet another modification to the MD-CNN called the hierarchical-CNN [27] can be used to obtain the same modes but in the order of their energy content. This order in energy helps in selecting the most dominant modes which further can be used to develop the best desired reduced order model.

Further, a real flow has more important patterns than the spatial ones. These are the temporal patterns that a flow has. As in the case of dynamic mode decomposition (DMD), AI tools are being developed to extract the spatio-temporal modes that helps in understanding and thus, developing a flow model that best captures the physics of the flow. The temporal modes are what helps in understanding the flow in time in the way similar to the spatial modes which helps in understanding the flow physics spatially. A small example will elucidate the importance of temporal coherence; a tornado is a very prominent spatio-temporal structure of a tropical storm weather system. Practically, it is very important for understanding this specific flow structure in space and time. Which is why extraction of spatio-temporal is very important.

Other than the use of AI tool for the extraction and study of modes, development of the AI tool will prove to be very crucial for future as (already mentioned before) it has too much arbitrariness to be handled efficiently. Already, much work is being done and the future is very bright for someone taking up AI and its use in fluid dynamics.

BIBLIOGRAPHY

[1] Elizabeth Gibney. Google ai algorithm masters ancient game of go. *Nature News*, 529(7587):445, 2016.

[2] Mike Daily, Swarup Medasani, Reinhold Behringer, and Mohan Trivedi. Self-driving cars. *Computer*, 50(12):18–23, 2017.

[3] Bernd R Noack, Marek Morzynski, and Gilead Tadmor. *Reduced-order modelling for flow control*, volume 528. Springer Science & Business Media, 2011.

[4] Erwan Liberge and Aziz Hamdouni. Reduced order modelling method via proper orthogonal decomposition (pod) for flow around an oscillating cylinder. *Journal of fluids and structures*, 26(2):292–311, 2010.

[5] Samantha Hayman. The mcculloch-pitts model. In *IJCNN'99. International Joint Conference on Neural Networks. Proceedings (Cat. No. 99CH36339)*, volume 6, pages 4438–4439. IEEE, 1999.

[6] Harold Szu and George Rogers. Generalized mcculloch-pitts neuron model with threshold dynamics. In *Int'l Joint Conf. Neural Networks*, 1992.

[7] Sun-Chong Wang. Artificial neural network. In *Interdisciplinary computing in java programming*, pages 81–100. Springer, 2003.

[8] Steven L Brunton, Bernd R Noack, and Petros Koumoutsakos. Machine learning for fluid mechanics. *Annual Review of Fluid Mechanics*, 52:477–508, 2020.

[9] Andrey Nikolaevich Kolmogorov. A refinement of previous hypotheses concerning the local structure of turbulence in a viscous incompressible fluid at high reynolds number. *Journal of Fluid Mechanics*, 13(1):82–85, 1962.

[10] Frank Rosenblatt. The perceptron: a probabilistic model for information storage and organization in the brain. *Psychological review*, 65(6):386, 1958.

[11] Marvin Minsky and Seymour A Papert. *Perceptrons: An introduction to computational geometry*. MIT press, 2017.

[12] Andrew Pollard, Thomas J Hacker, and Shirley Dyke. Whither turbulence and big data for the twenty-first century. In *Whither Turbulence and Big Data in the 21st Century?*, pages 551–574. Springer, 2017.

[13] Khemraj Shukla, Patricio Clark Di Leoni, James Blackshire, Daniel Sparkman, and George Em Karniadakis. Physics-informed neural network for ultrasound nondestructive quantification of surface breaking cracks. *Journal of Nondestructive Evaluation*, 39(3):1–20, 2020.

[14] Xuhui Meng, Zhen Li, Dongkun Zhang, and George Em Karniadakis. Ppinn: Parareal physics-informed neural network for time-dependent pdes. *Computer Methods in Applied Mechanics and Engineering*, 370:113250, 2020.

[15] Maziar Raissi, Paris Perdikaris, and George E Karniadakis. Physics-informed neural networks: A deep learning framework for solving forward and inverse problems involving nonlinear partial differential equations. *Journal of Computational Physics*, 378:686–707, 2019.

[16] Kurt Hornik, Maxwell Stinchcombe, and Halbert White. Multilayer feedforward networks are universal approximators. *Neural Networks*, 2(5):359–366, 1989.

[17] Lawrence Sirovich. Turbulence and the dynamics of coherent structures. i. coherent structures. *Quarterly of applied mathematics*, 45(3):561–571, 1987.

[18] John L Lumley. Coherent structures in turbulence. In *Transition and turbulence*, pages 215–242. Elsevier, 1981.

[19] AKM Fazle Hussain. Coherent structures and turbulence. *Journal of Fluid Mechanics*, 173:303–356, 1986.

[20] Carlos Michelén Ströfer, Jinlong Wu, Heng Xiao, and Eric Paterson. Data-driven, physics-based feature extraction from fluid flow fields. *arXiv preprint arXiv:1802.00775*, 2018.

[21] Michele Milano and Petros Koumoutsakos. Neural network modeling for near wall turbulent flow. *Journal of Computational Physics*, 182(1):1–26, 2002.

[22] Pierre Baldi and Kurt Hornik. Neural networks and principal component analysis: Learning from examples without local minima. *Neural networks*, 2(1):53–58, 1989.

[23] Enrico Fonda, Ambrish Pandey, Jörg Schumacher, and Katepalli R Sreenivasan. Deep learning in turbulent convection networks. *Proceedings of the National Academy of Sciences*, 116(18):8667–8672, 2019.

[24] W Rodi. Comparison of les and rans calculations of the flow around bluff bodies. *Journal of wind engineering and industrial aerodynamics*, 69:55–75, 1997.

[25] Takaaki Murata, Kai Fukami, and Koji Fukagata. Nonlinear mode decomposition with convolutional neural networks for fluid dynamics. *Journal of Fluid Mechanics*, 882, 2020.

[26] Jean-Christophe Loiseau, Steven L Brunton, and Bernd R Noack. 9 from the pod-galerkin method to sparse manifold models. In *Applications*, pages 279–320. De Gruyter, 2020.

[27] Kai Fukami, Taichi Nakamura, and Koji Fukagata. Convolutional neural network based hierarchical autoencoder for nonlinear mode decomposition of fluid field data. *Physics of Fluids*, 32(9):095110, 2020.

[28] Ryo Saegusa, Hitoshi Sakano, and Shuji Hashimoto. Nonlinear principal component analysis to preserve the order of principal components. *Neurocomputing*, 61:57–70, 2004.

[29] Charles-Henri Bruneau and Mazen Saad. The 2d lid-driven cavity problem revisited. *Computers & fluids*, 35(3):326–348, 2006.

[30] O Botella and R Peyret. Benchmark spectral results on the lid-driven cavity flow. *Computers & Fluids*, 27(4):421–433, 1998.

[31] JR Koseff and RL Street. The lid-driven cavity flow: a synthesis of qualitative and quantitative observations. 1984.

[32] UKNG Ghia, Kirti N Ghia, and CT Shin. High-re solutions for incompressible flow using the navier-stokes equations and a multigrid method. *Journal of computational physics*, 48(3):387–411, 1982.

[33] Mohammad Athar Khan, Syed Fahad Anwer, Saleem Anwar Khan, and Nadeem Hasan. Hydrodynamic and heat transfer characteristics of vortex-induced vibration of square cylinder with various flow approach angle. *International Journal of Thermal Sciences*, 156:106454, 2020.

[34] Nadeem Hasan and Saleem Anwar Khan. Two-dimensional interactions of non-isothermal counter-flowing streams in an adiabatic channel with aiding and opposing buoyancy. *International Journal of Heat and Mass Transfer*, 54(5-6):1150–1167, 2011.

[35] Léon Bottou. Stochastic gradient descent tricks. In *Neural networks: Tricks of the trade*, pages 421–436. Springer, 2012.

[36] YC Liang, HP Lee, SP Lim, WZ Lin, KH Lee, and CG Wu. Proper orthogonal decomposition and its applications—part i: Theory. *Journal of Sound and vibration*, 252(3):527–544, 2002.

[37] Anindya Chatterjee. An introduction to the proper orthogonal decomposition. *Current science*, pages 808–817, 2000.

[38] JL Holmes. Lumley, and g. berkooz. *Turbulence, coherent structures, dynamical systems and symmetry*, 1996.

[39] HM Park and DH Cho. The use of the karhunen-loeve decomposition for the modeling of distributed parameter systems. *Chemical Engineering Science*, 51(1):81–98, 1996.

[40] Kay A Robbins. Visualization of scientific video data using kl decomposition. *IEEE transactions on visualization and computer graphics*, 4(4):330–343, 1998.

[41] BF Feeny and R Kappagantu. On the physical interpretation of proper orthogonal modes in vibrations. *Journal of sound and vibration*, 211(4):607–616, 1998.

[42] B Ravindra. Comments on "on the physical interpretation of proper orthogonal modes in vibrations". *Journal of Sound and Vibration*, 1(219):189–192, 1999.

[43] R Kappagantu and BF Feeny. An" optimal" modal reduction of a system with frictional excitation. *Journal of Sound and vibration*, 224(5):863–877, 1999.

[44] RV Kappagantu and BF Feeny. Part 1: Dynamical characterization of a frictionally excited beam. *Nonlinear Dynamics*, 22(4):317–333, 2000.

[45] Ian T Jolliffe. Principal components in regression analysis. In *Principal component analysis*, pages 129–155. Springer, 1986.

[46] Konstantinos I Diamantaras and Sun Yuan Kung. *Principal component neural networks: theory and applications*. John Wiley & Sons, Inc., 1996.

[47] Virginia Klema and Alan Laub. The singular value decomposition: Its computation and some applications. *IEEE Transactions on automatic control*, 25(2):164–176, 1980.

APPENDIX

A. POD

The proper orthogonal decomposition (POD) [36]is distinguished and at the same time very potent tool used for data analysis whose aim is to reduce the ensuing dimensions. The low-dimensional entities are approximate description of high dimensional process. The POD gives the basis vectors for the decomposition of modes of some ensemble functions. Mostly, these data comprise of either the observations from the experimental data of DNS computation. The basis functions used to compress the data go by many names being proper orthogonal modes, basis vectors, empirical eigenfunctions, or empirical basis functions. The stark feature of POD is its ideality; able to provide with the basis vectors or modes in the most efficient way possible extracted from infinite-dimensional process. The extracted modes number is very low [37, 38]. Universally, there are two different elucidations for the POD. The first elucidation regards the POD as the Karhunen-Loeve decomposition (KLD) [39, 40] and the second one considers that the POD consists of three methods: the KLD, the principal component analysis (PCA), and the singular value decomposition (SVD) [37, 38, 41–44, 44]. The first elucidation presents itself in many engineering literatures related to the POD. It is also used to prove existence of as many spatial modes as there are snaps (and no more)of flow available. Because of the close connections and the equivalence of the three methods, the authors prefer the second elucidation for the POD, that is, the POD includes the KLD, PCA and SVD.

The POD was a thought child of many independent investigators traced by Lumley. The main idea of the POD is to look for a set of ordered orthogonal basis vectors in a subspace (without loss of generality) where a random vector takes its values, such that the samples in the sample space can be expressed optimally using the selected first l basis vectors. The mean square error can be used as a measure for the optimal problem given by equation A.1.

$$E||x - x(l)||) \leq E(||x - \hat{x}(l)||^2 \tag{A.1}$$

where $x(l)$ is the approximate expression of a random vector x using the first l basis vectors of the undetermined set of orthonormal basis vectors, and $\hat{x}_(l)$ is the approximate expression of x using arbitrary l basis vectors

The two methods of PCA [**?**, 45, 46] and KLD as an extension of PCA to infinite-dimensional space are not the matter of discussion here. The SVD or singular-

value decomposition method for POD was used. Klema and Laub [47] showed that the SVD was put forward for real-square matrices in the 1870s by Beltrami and Jordan, for complex square matrices in 1902 by Autonne, and for general rectangular matrices in 1939 by Eckart and Young. SVD is the extension of the eigen value decomposition for the case of non-square matrices which is often the case for real fluid flow data especially.
The final and best factorization of a matrix is given by equation A.2 where, U and V are orthogonal and Σ is diagonal. In the decomoposition of equation A.2, A can be any matrix. It is known that if A is symmetric positive definite, its eigenvectors are orthogonal and it can written that $A = Q \wedge Q^T$. This is a special case of a SVD, with U = V = Q. For more general A, the SVD requires two different matrices U and V. The data in case of dimensionality reduction is made such that the SVD gives the required orthogonal basis vectors or modes. Among the factors of SVD the matrx Σ contains the energy value in ascending order which makes the POD so optimal. The orthogonal U contains the eigen vectors or modes.

$$A = U\Sigma V^T \tag{A.2}$$

B. ACTIVATION FUNCTIONS

The different activation functions used are presented for the perusal of the reader.

B.1 Linear activation function

The most primitive kind of activation function that can be thought of is the linear activation function. The linear activation function was first used when the first perceptron was developed by McCulloch and Pitts for mapping input to output as an identity i.e. as an identity map. Mathematically, linear activation function is given in equation B.1a and its derivative by the equation B.1b. The figure B.1 gives the same graphically.

$$f(x) = x \tag{B.1a}$$

$$f'(x) = 1 \tag{B.1b}$$

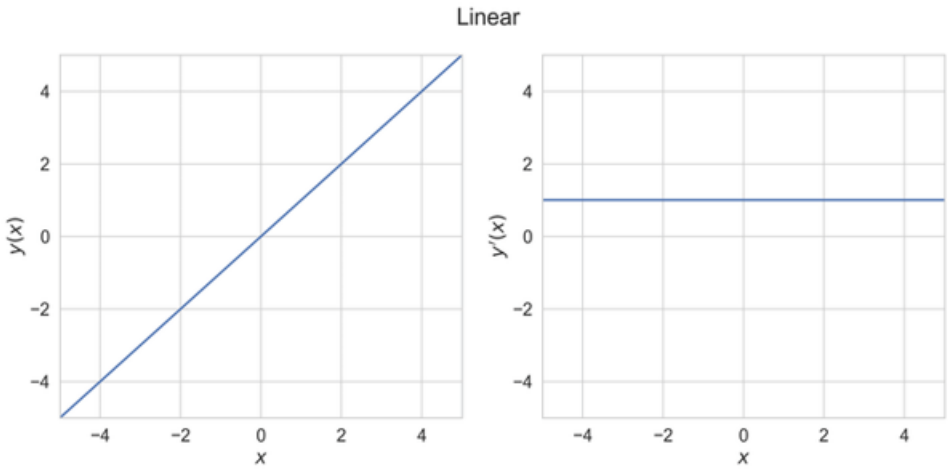


Fig. B.1: Linear activation and its derivative

Two major problems associated with the linear activation functions are:

1. The back propagation algorithm is useless in case of the linear activation function as its derivative is a constant and thus no relation regarding the

output to the input is known which eventually can't help to optimize the weights

2. A full complex neural network with multiple hidden layers collapses to a single hidden layer neural network as the new layers are just the linear combination of the previous layer, giving the same value over and over again.

Basically, the neural network with linear activation function is a linear regression model and as such cannot capture any non-linearity.

B.2 ReLU activation function

A rectified linear unit or ReLU is a piece wise activation function that gives output only when the input is positive, which furthermore is akin to the linear activation function. This activation function is the easiest to train and solves the vanishing gradient problem. It is also the default activation function that is used when developing the multi-layer neural network. Mathematically, ReLU is given by equation B.2 and figure B.2 shows the same graphically.

$$f(x) = max(0, x) \tag{B.2a}$$

$$f'(x) = \begin{cases} 0 & \text{if } x < 0 \\ 1 & \text{if } x > 0 \end{cases} \tag{B.2b}$$

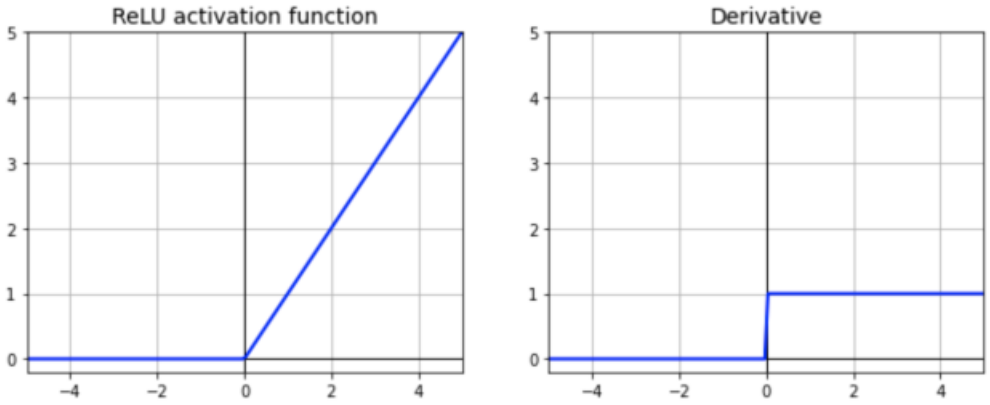


Fig. B.2: ReLU activation function and its derivative

The problem of no learning for negative inputs can be mitigated by allowing for some minimal output value given by equation B.3. The modified relu function if called the leaky relu. The function thus obtained is shown graphically in the figure B.3.

$$f(x) = \begin{cases} x & \text{if } x > 0 \\ \alpha x & \text{if } x \leq 0 \end{cases} \tag{B.3a}$$

$$f'(x) = \begin{cases} 1 & \text{if } x > 0 \\ \alpha & \text{if } x < 0 \end{cases} \tag{B.3b}$$

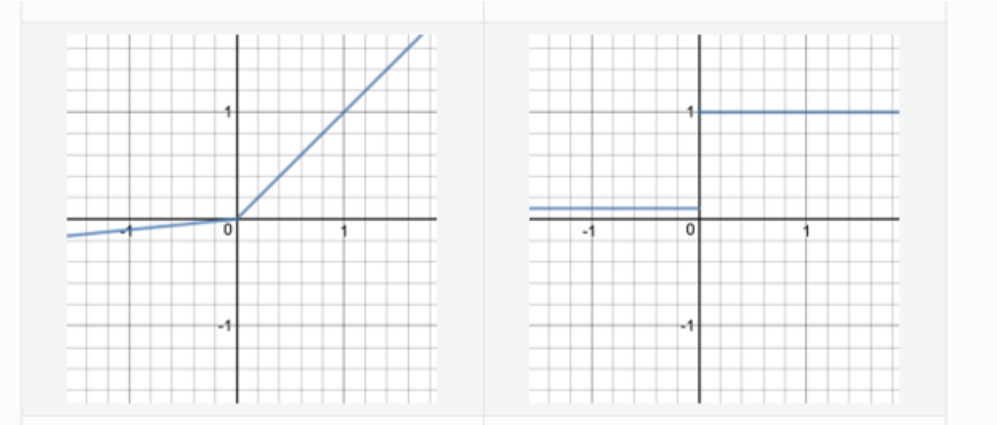


Fig. B.3: Leaky ReLU function and its derivative

B.3 Sigmoid activation function

Sigmoid function is also known as the logistic function. Like ReLU it was widely used during the inception of neural network and mahcine learning. The output of this function lies between 0 and 1. Mathematically, it is given by equation B.4 and represented graphically in figure

$$f(x) = \frac{1}{1 + e^x} \tag{B.4}$$

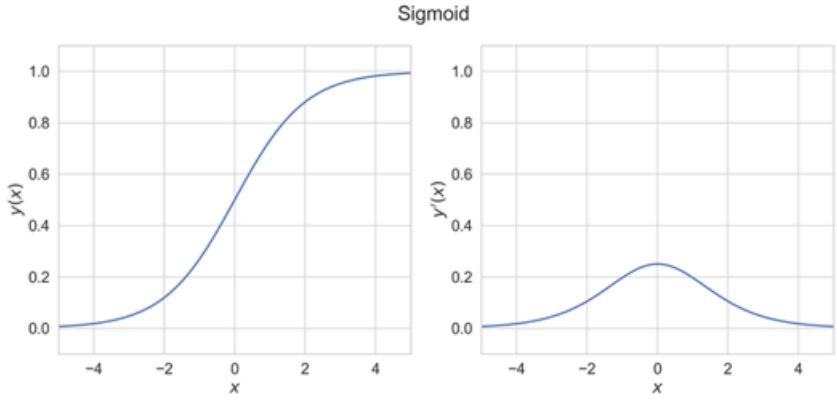


Fig. B.4: Simgoid activation function and its derivative

The sigmoid activation is powerful when used for the case of classification. The con of sigmoid activation is that of gradient vanishing when the input is near to 0. This hampers the learning as it completely switches of the node which in turn is reflected back because of back propagation.

B.4 Softmax activation function

Sigmoid is a probabilistic activation function similar to sigmoid. Mathematically, it is given by equation B.5. The graph is similar to that of sigmoid and thus, it entails the same problem as that of sigmoid.

$$f(x_i) = \frac{e^{x_i}}{\sum_{j=1}^{k} e^{x_j}} \tag{B.5}$$

B.5 Softplus activation function

The softplus activation function is smooth approximation of ReLU. The output is constrained to be always positive no matter the input. The equation gives the function and its derivative.

$$f(x) = \ln(1 + e^x) \tag{B.6a}$$

$$f'(x) = \frac{1}{1 + e^{-x}} \tag{B.6b}$$

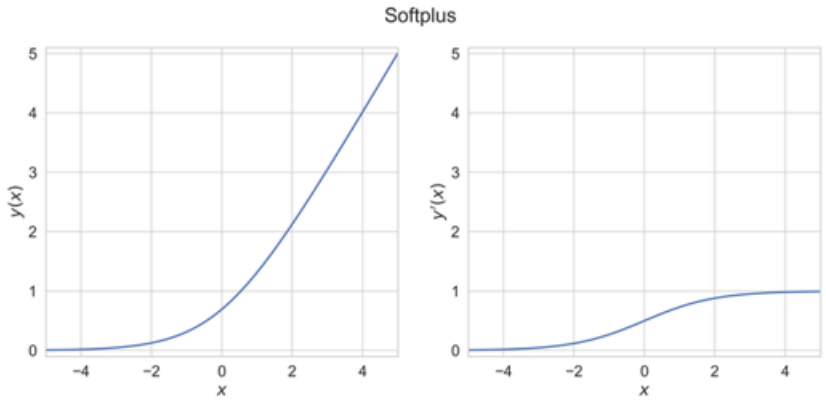


Fig. B.5: Softplus activation function and its derivative

The softplus activation function has a con for when the input is large the output explodes to an even larger value, which adulterates the learning.

B.6 Softsign activation function

It is a neuron activation function that is based on the mathematical function given by equation B.7 which is plotted in figure B.6.

$$f(x) = \frac{x}{1 + |x|} \tag{B.7a}$$

$$f'(x) = \frac{1}{(1 + |x|)^2} \tag{B.7b}$$

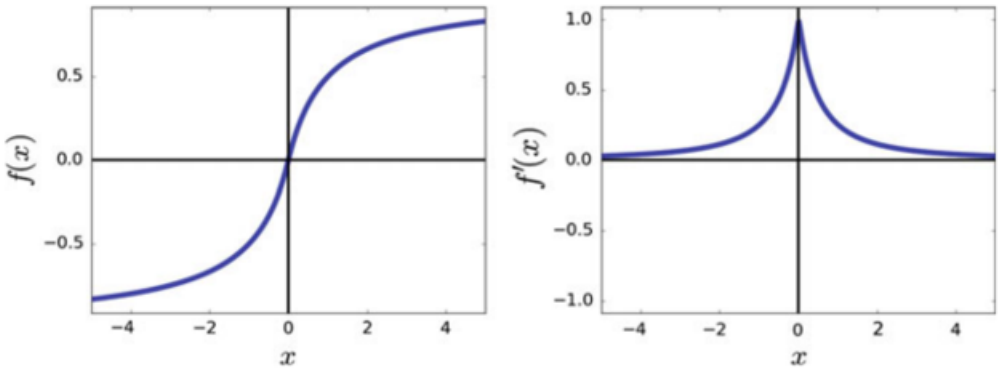


Fig. B.6: Softsign activation function and its derivative

The main advantage of softsign is that its output value is centered at 0 and its output range is [-1,1]. Also, there is no gradient vanishing problem as with the sigmoid. It activates the neuron very effectively making the learning better on expense of time.

B.7 Tanh activation function

Evident from the name it outputs the value as tan hyperbolic value of the input. The output range is same as softsign being [-1,1]. Mathematically, it is given by equation B.8 and shown in figure B.7

$$f(x) = tanh(x) = \frac{e^x + e^- x}{e^x - e^- x} \tag{B.8a}$$

$$f'(x) = 1 - (f(x))^2 \tag{B.8b}$$

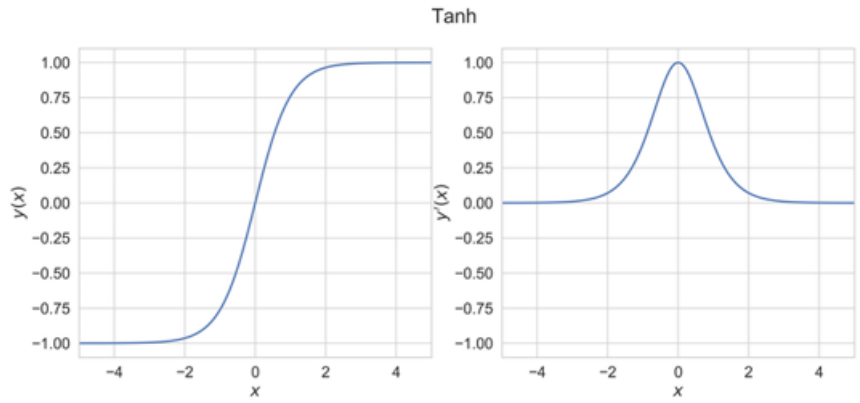


Fig. B.7: Tanh activation function and its derivative

Tanh has improvement over the sigmoid activation functions being the stronger gradient near 0 input values. But it still has the gradient vanishing problem although the range of that is very small. Also, unlike sigmoid its value is centered around 0 giving intensive activation of neurons.

B.8 Elu activation function

Elu or exponential linear unit is mathematically expressed by the equation . It is piece-wise activation function.

$$f(x) = \begin{cases} x & \text{if } x \geq 0 \\ (e^x - 1) & \text{if } x < 0 \end{cases} \tag{B.9a}$$

$$f'(x) = \begin{cases} 1 & \text{if } x < 0 \\ e^x & \text{if } x > 0 \end{cases} \tag{B.9b}$$

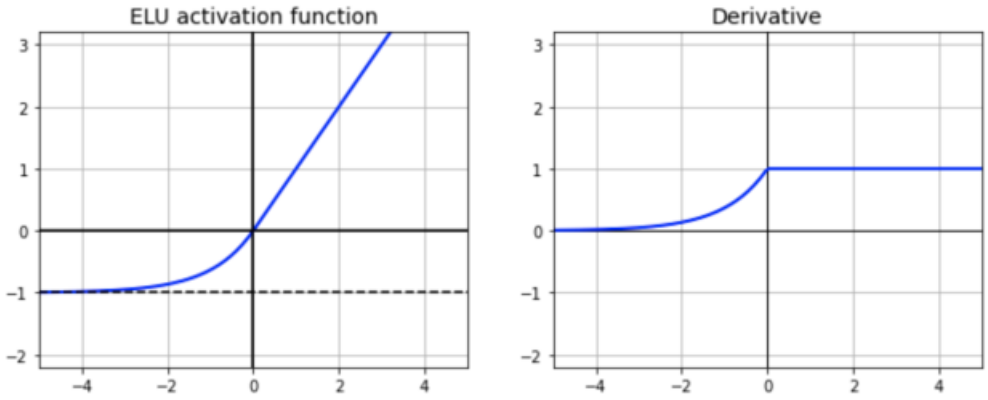


Fig. B.8: Elu activation function and its derivative

B.9 Selu activation function

Selu or scaled exponential unit. Given by equation

$$f(x) = \lambda \begin{cases} x & \text{if } x \geq 0 \\ \alpha(e^x - 1) & \text{if } x < 0 \end{cases} \tag{B.10a}$$

$$f'(x) = \lambda \begin{cases} 1 & \text{if } x < 0 \\ \alpha e^x & \text{if } x > 0 \end{cases} \tag{B.10b}$$

The parameters λ and α are arbitrary and are decided according to the need making it more flexible than elu and expecially relu, as it can omit the exploding output value problem for the certain input value. The figure B.9 give the corresponding graph.

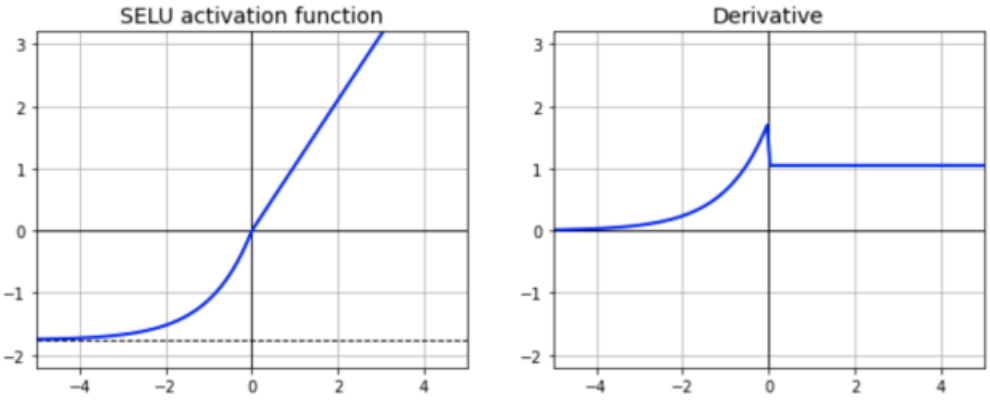


Fig. B.9: Selu activation function and its derivative

The final activation function is exponential which needs no explanation.