PREDICTION OF FLUID FLOW AROUND 2D SQUARE OBJECTS INSIDE A DOUBLE LID DRIVEN CAVITY USING CFD AND ARTIFICIAL NEURAL NETWORK

MD. ATIF YASIR

(*B.Sc. Engg., MIST*)

A THESIS SUMITTED FOR THE DEGREE OF

MASTER OF SCIENCE IN MECHANICAL ENGINEERING

DEPARTMENT OF MECHANICAL ENGINEERING

MILITARY INSTITUTE OF SCIENCE AND TECHNOLOGY

2021

APPROVAL OF BOARD OF EXAMINERS

The thesis titled "**PREDICTION OF FLUID FLOW AROUND 2D SQUARE OBJECTS INSIDE A DOUBLE LID DRIVEN CAVITY USING CFD AND ARTIFICIAL NEURAL NETWORK**" Submitted by Md. Atif Yasir, Roll No: 0418180012 (P), Session: 2017-18 has been accepted as satisfactory in partial fulfillment of the requirement for the degree of Master of Science in Mechanical Engineering.

BOARD OF EXAMINERS

1. _____ Supervisor
   Professor Dr. Dipak Kanti Das
   Department of Mechanical Engineering
   Military Institute of Science and Technology

2. _____ Co-supervisor
   Professor Dr. Md. Mahbubur Rahman
   Department of Computer Science and Engineering
   Military Institute of Science and Technology

3. _____ Ex-officio
   Brig Gen Md. Humayun Kabir Bhuiyan
   Department of Mechanical Engineering
   Military Institute of Science and Technology

4. _____ Member (Internal)
   Lt Col Tahmina Sultana, PhD
   Department of Science and Humanities (Math)
   Military Institute of Science and Technology

5. _____ Member (External)
   Professor Dr. A.B.M. Toufique Hasan
   Department of Mechanical Engineering
   Bangladesh University of Engineering and Technology

DECLARATION

I hereby declare that this thesis is my original work and it has been written by me in its entirety. I have duly acknowledged all the sources of information, which have been used in the thesis.

_____

Md. Atif Yasir
Department of Mechanical Engineering
Military Institute of Science and Technology
10 March 2021

SUMMARY

Using computational fluid dynamics (CFD) to solve fluid flow problems can use a lot of computer processing power and simulation time. Artificial neural networks (ANN) can be regarded as universal learners that are capable of learning nonlinear patterns or relationships among many variables. A very well-known benchmark problem for viscous incompressible fluid flow in the lid-driven cavity problem. People have developed different numerical procedures to solve it. It is widely regarded as the first problem people usually try to solve when they come up with a new approach. This research aims to apply fully connected neural networks to learn and predict fluid flow inside a lid-driven cavity. A double lid-driven cavity with top and bottom moving walls having some internal square objects was selected as a training data to train, test and compare several fully connected neural networks having different parameters to predict fluid flow inside it. The results show that by training a neural network to recognize fluid velocity patterns around simple square objects inside the cavity, it is possible to predict fluid velocities around objects having relatively complex geometries with significant accuracy in a fraction of the time required by a CFD solver. The results also show the comparison between effects of using different mesh sizes in CFD and different learning rates in the neural network model.

Keywords: *CFD, ANN, Double lid-driven cavity, Fluid flow prediction*

ACKNOWLEDGEMENTS

First and foremost, I would like to thank the Almighty for His continuous blessings that enabled me to come this far. What started as a prayer has now come to a successful ending.

I would like convey my sincerest gratefulness to my supervisor Professor Dr. Dipak Kanti Das and my co-supervisor Professor Dr. Md. Mahbubur Rahman for giving me the opportunity and guiding me in every step in performing a collaborative research involving computational fluid dynamics and artificial neural networks. Without their continuous support, this would have never been possible. It has truly been my honor and privilege to work with them.

I would like to thank the Head of the department of Mechanical Engineering, Brig Gen Md Humayun Kabir Bhuiyan for his powerful encouragements throughout the whole course. I would also like to thank Professor Dr. G. M. Jahid Hasan for supporting me academically and personally. Sincere thanks to Professor Dr A.B.M. Toufique Hasan and Lt Col Tahmina Sultana, PhD for helping me to evaluate this research.

Finally, I would like to thank my parents and my siblings for supporting me and taking care of me in every step of my life. I am truly indebted to them for their understanding, encouragement and patience that they have shown throughout all my life.

TABLE OF CONTENTS

LIST OF SYBMOLS

w	Neural network weights
b	Neural network biases
U, u	Velocity in x-direction
V, v	Velocity in y-direction
P, p	Pressure
ρ	Density
μ	Dynamic Viscosity
t	Time
Re	Reynolds number
∇	Divergence operator
δ	Artificial compressibility
L	Characteristic Linear Dimension

LIST OF FIGURES

LIST OF TABLES

CHAPTER ONE

INTRODUCTION

1.1. Background of the Study

Computational fluid dynamics (CFD) is a subset of fluid mechanics that uses numerical simulation and data structures to study and solve problems concerning fluid flows. Computers are used to perform the calculations used to model the free-stream flow of the fluid and the interaction of the fluid (liquids and gases) with surfaces defined by boundary conditions. Better solutions can be found with high-speed supercomputers, which are often used to solve the biggest and most difficult problems. Current research is yielding software that increases the accuracy and speed of complex simulation scenarios like transonic or turbulent flows. Usually, experimental apparatus such as wind tunnels are used to conduct initial validation of such applications. Furthermore, a recently completed analytical or scientific study of a particular problem may be compared. Full-scale simulation, such as flight simulations, is often used for final confirmation. Aerodynamics and aerospace analysis, weather modeling, natural science and environmental engineering, manufacturing device architecture and analysis, biological engineering, fluid flows and heat transfer, and engine and combustion analysis are only a few of the scientific and engineering issues that CFD is used to solve in a variety of fields and industries.

Software programs like ANSYS, SimScale, OpenFOAM can accurately create such simulations and using these simulations, engineers can modify or redesign according to their needs. These software programs use iterative algorithms to calculate fluid velocity, pressure and temperature from initial and boundary values. The iterative approaches are time-consuming and often take days to complete. A new possible approach can be to use

Artificial Intelligence (AI) to simulate the fluid flow. It is modeled after our brain in a way which enables it to learn from examples just like how we learn from the very day we are born. In case of fluid mechanics or fluid flow predictions, an AI can be trained using thousands of previously generated data with Artificial Neural Network (ANN) architectures, which can be used to predict the solution to an unknown set of problems in a fraction of time. This approach differs from the traditional CFD algorithms, which do not have the capabilities to "remember" how fluid behaves around several shapes of objects. This can, for example, be a lot helpful for airplane manufacturers who have to spend a great deal of time trying to make a more efficient wing design. Instead of using CFD software to simulate fluid flow around the wings, they can use AI to predict fluid flow around them in a reduced amount of time.

1.2. Objectives

This research has the following three objectives:

i. To use the Navier-Stokes equations to calculate fluid velocities inside a double lid-driven cavity with different grid sizes having a square object inside it in different positions.

ii. To train several fully connected neural networks incorporating these velocities generated using different grid sizes with globally fixed learning rates for all the layers and layer-specific different learning rates.

iii. To test all the neural networks with objects placed in different locations and compare their accuracies with the CFD results in terms of grid sizes, learning rates and computational times.

CHAPTER TWO

REVIEW OF LITERATURE

2.1. Previous Works

Kutz [1] demonstrated the benefits of combining deep learning and fluid dynamics. Deep neural networks (DNNs), he argued in his article, would almost certainly have a transformative effect on modeling high-dimensional complex systems like turbulent flows. This latest technology would force researchers to use this increasingly evolving data analysis method for enhancing predictive capability by integrating multiple diverse data sets. DNNs obviously reflect a paradigm shift for the group, who are attempting to come up with fresh and creative ways to replace current structures in order to better explain how the underlying mechanism operates and how trends can be observed in even the most complex and seemingly random systems. Many developments have been influenced by expert-in-the-loop insight and mechanically interpretable models, but DNNs have defied these common theories by creating prediction engines that clearly outperform competing approaches without supplying concrete examples of why.

Baymani et al. [2] have used a new neural network-based approach for obtaining the solution of the Navier–Stokes equations in an analytical function form in their paper. The solution protocol was based on the formation of a two-part trial solution. As a result, there were no customizable parameters in the first section, which fulfilled the boundary conditions directly. The second component was designed to satisfy the governing equation within the solution domain while leaving the boundary conditions alone. This part involved a feed-forward neural network with adjustable parameters (weights) that had to be calculated such that the estimated error function generated was as small as possible. The

capabilities of the method were shown by solving the Navier–Stokes problem with various boundary conditions, and the method's details were discussed. By comparing the method's efficiency and the consistency of the results to the existing numerical and analytical solutions, the method's performance and accuracy were assessed.

Mccracken [3] has developed a novel method for solving the Navier-Stokes Equations for turbulence by training a neural network to model ionospheric velocity fields based on 3-dimensional inputs using Bayesian Cluster and SOM neighbor weighting. The velocity, Reynold's number, Prandtl number, and temperature were all used in this issue. Data from Johns Hopkins University was used to train the neural network in MATLAB for this research. The velocity fields were mapped by the neural network with a 67 percent accuracy using the validation results.

Sabir and Ya [4] have used a novel ANN technique for fluid flow modeling. They attempted to obtain instantaneous numerical simulation for fluid flow using artificial neural networks in their study. The geometrical boundaries profile was considered a significant contribution for the ANN training process in the proposed system. Their research was motivated by the need for quick responses, especially in medical situations, surgeon diagnoses, engineering crises and when unusual circumstances arise. They were able to achieve satisfactory results for 1D-flow equations in terms of both energy and momentum equations. Their ANN method was effective in predicting fluid flow with known boundary velocity.

Guo et al. [5] used Convolutional Neural Networks (CNN) for steady flow approximation. They suggested a general and scalable approximation model based on

convolutional neural networks for real-time prediction of non-uniform steady laminar flow in a 2D or 3D domain (CNNs). They looked at different options for CNN geometry representation and network architecture. They demonstrated that convolutional neural networks could approximate velocity fields two orders of magnitude faster than a GPU-accelerated CFD solver and four orders of magnitude faster than a CPU-based CFD solver while maintaining a low error rate.

Tsunooka et al. [6] used high-speed CFD simulation to forecast crystal growth. They used a neural network to optimize the growth conditions and quickly predict the outcomes of computational fluid dynamics (CFD) simulations for SiC solution growth. A single CFD simulation was 10^7 times faster than the prediction speed. As a result of the combination of CFD simulation and machine learning, optimal parameters for high-quality and large-diameter crystals could be determined. As a result, they predicted such a simulation to become the technology used in the design and control of crystal growth processes.

2.2. This Work

Motivated by these researches, this research aims to apply fully connected neural networks to predict fluid flow in order to decrease the time required for calculating the velocities while preserving much of the accuracy of a full-fledged CFD solution. A double lid-driven cavity with top and bottom moving walls having some internal square objects was selected as a training data to train, test and compare several fully connected neural networks having different parameters to predict fluid flow inside it.

CHAPTER THREE

ARTIFICIAL NEURAL NETWORKS

3.1. Foundation

Neural networks and deep learning are hot topics in the realm of computer science and technology. Since the beginning of this century, the world has seen many developments in machine learning algorithms. Such algorithms are extremely complex and are often used for doing things that were thought to be impossible even as little as 40 years ago. Things like recognizing a face in a stadium full of thousands of people, driving a car through 9am traffic, playing chess with renowned grandmasters etc. are some of the accomplishments people have achieved with artificial intelligence and machine learning technology. This wonderful technology is modeled after probably the most complex object in the world that we often take for granted, and that is our brain. The fundamental computational or logical unit of the brain is a neuron as shown in Figure 3.1.

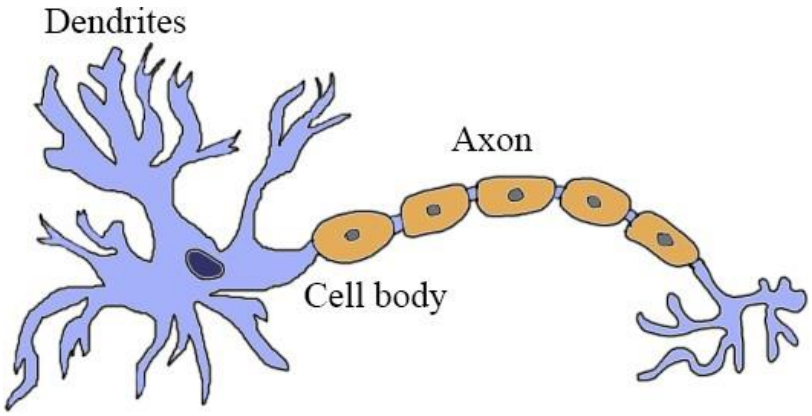


Figure 3.1: Biological neuron

The entire brain is made up of billions of neurons connected to each other in a never-ending maze of complexity. These connections are called Synapses and there are

approximately 10^{14} - 10^{15} of such in a single brain. They enable humans to learn, adapt and take control of nature most of the time without even having to think about it. The basic structure of a neuron involves a cell body, which has some small branches attached to it and they are called dendrites. There is also a long distinctive wire like object attached to the cell body and it is called the axon. The axon carries information from the cell body and passes it to the next neuron through the dendrites. This connection between the two neurons is called a synapse. After we are born, we start to learn both consciously and subconsciously. Our brain takes in signals from our nerve endings and processes them or remembers them by making connections between the neurons. When we walk, we usually do not have to concentrate too much on walking. We do not calculate every step and take necessary actions. Our brain has learned the method of walking through many failed attempts when we were taught to walk in our childhood days. The synapses in our brain "remembers" these failed attempts and adjusts our leg muscles in a way that enables them to keep us walking without falling. It is no wonder that Artificial Intelligence, being the driving force of modern technology is modeled after what is thought to be the most complex system ever known to mankind, our brain.

3.2. Basic Structure

The basic structure of a neural network also involves neurons just like the brain. It receives information from the previous neurons, processes it and sends it to the next neuron. Information from the previous neurons are received as numerical values (x_0, x_1, x_2 etc.) as show in Figure 3.2. These are multiplied with the "weights" (w_0, w_1, w_2 etc.), which are another set of numerical values assigned to the connections. The multiplied values of these weights (w_0, w_1, w_2 etc.) and input values (x_0, x_1, x_2 etc.), along with another numerical value assigned to each neuron, called the bias (b), are then added up and processed by the

cell body using an activation function. The processed value is then passed to the next neuron.

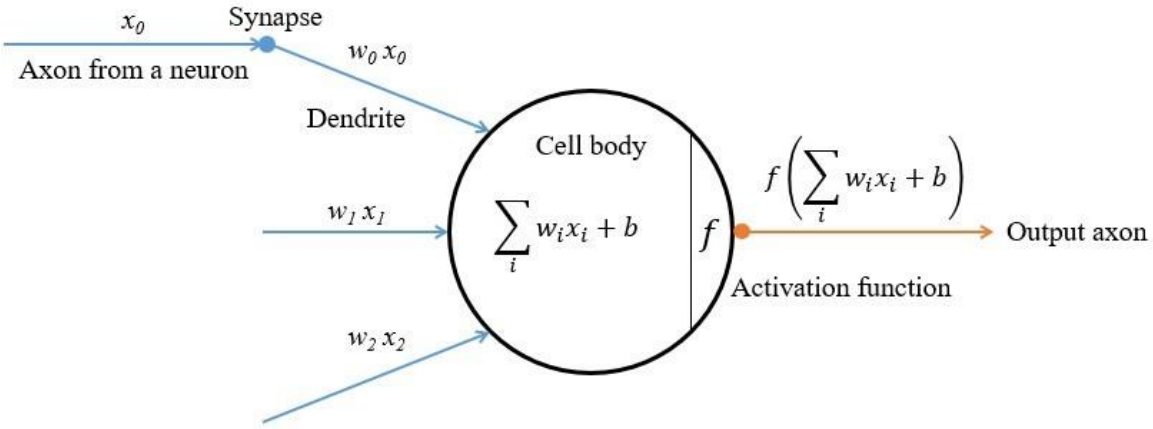


Figure 3.2: Neuron in neural network

Several activation functions are used to get output from each neuron. Sigmoid function, hyperbolic tan (tanh), rectified linear unit (ReLU) are some of the most widely used activation functions in neural networks. Sigmoid and ReLU functions have been used as activation functions in this research.

3.3. Types of Neural Networks

There are many types of neural networks and each of them have their own applications along with a set of advantages and disadvantages. The models that are mostly used include:

3.3.1. Feedforward fully connected neural network

A feedforward fully connected neural network is an artificial neural network where the connections between the neurons do not repeat or in other words, there are no cycles. In this network, the information moves through the neurons in a single direction. During

the forward stroke or forward movement of data, information flows from the input neurons in the input layers, through the hidden nodes in the hidden layer (if any) and to the output nodes in the output layer. Output from any neuron does not end up being an input to its preceding neuron. Similarly, during training of the network, the network parameters are updated sequentially in a reverse direction. This starts from the output layer, goes through all the hidden layers and ends with the input layer. The input layer takes input and the output layer generates output. Each neuron in each layer is connected to all the neurons in the previous layer; hence, the name is "Fully connected neural network". Feedforward neural networks are mainly used in classification, regression and data prediction problems. A fully connected feedforward neural network has been used in this research where the input nodes represent the structure of the environment in which fluid is flowing and the output nodes represent the velocities of the fluid in that specific environment. A four-layer neural network model has been used which has one input layer, one output layer and two hidden layers.

3.3.2. Convolutional neural network

Convolutional neural networks work in a different way. These networks are typically used with image classification problems. In contrast with fully connected neural networks, the neurons or nodes in convolution neural networks are not fully connected. When dealing with image classification problems, the input neurons represent the color value of the pixels that make up the image and not all pixels are connected to the next layer of the network. Instead, the image is divided into several regions and the inputs of these regions are connected separately to the nodes in the succeeding layer. Finally, fully connected layers are used as the outer most hidden layers and the output layers.

3.3.3. Recurrent neural network

Recurrent neural networks have direct connections between neurons or nodes in a certain layer and its preceding layer. In contrast with feedforward networks where information flows in a single direction only, in recurrent networks, cycles can be seen in different stages of the network that enables the network to feed information directly from a certain layer to its preceding layer. These networks are typically used in prediction of power consumptions in electric grid systems throughout the world.

3.4. Activation Functions

Using the idea that a neuron can "fire" inside our brain that represents flow of data from one neuron to another, the mathematical modelling of neural networks also have functions that mimics the brain and enables a neuron to "fire" or transfer data from one neuron to another. Since these neurons are connected to each other, these transfer of data from one neuron to another forms a chain of complex network that enables the entire network to get modelled in such a way that they can remember what input values can recreate which neurons to fire. The functions that make the neurons fire are called activation functions. Several types of activation functions are used in neural networks. Some of the mostly used activation functions include:

3.4.1. Sigmoid function

The sigmoid function is a well-known mathematical function that can take any numerical value and output a value ranging from 0 to 1. The sigmoid function can be written as:

$$sigmoid(x) = \frac{1}{1 + e^{-x}} \tag{1}$$

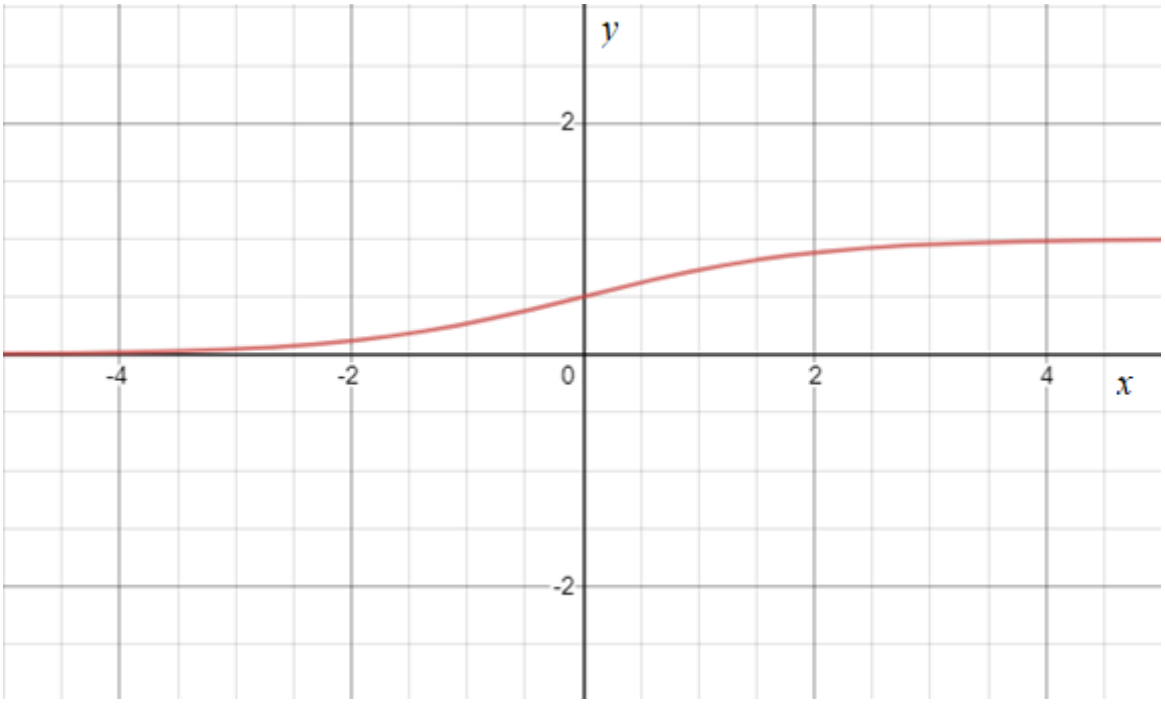


Figure 3.3: Sigmoid function

The main advantage of using a sigmoid function as activation function is that it always produces an output ranging from 0 to 1. This crunching of data enables all the neurons in all the layers of the network to have a controlled and efficient flow of information through the network. Sigmoid functions play a major role in classification problems where the output neurons simply have to generate a value close to 0 or 1 to represent the input data being classified between several classifications. One good example of this is a neural network, which is designed to classify between handwritten numbers. There are ten numbers in the typical 10-based numbering system and so this network can have 10 neurons in the output layer. If only the first neuron produces a value close to 1 and other neurons produce values close to 0, the input number can be classified as a "0". So the output from the output layer may look like "1000000000" for "0". Similarly, for "1", the output layer may look like "0100000000" and for "9", the output layer may look like "0000000001".

3.4.2. Hyperbolic tan function

The hyperbolic tan or tanh(x) is closely related to the sigmoid function in terms of the output it can generate based on the inputs it receives. Tts output ranges from -1 to 1 as opposed to the 0 to 1 range of the sigmoid function. The hyperbolic tan function can be written as:

$$\tanh(x) = \frac{e^{x} - e^{-x}}{e^{x} + e^{-x}} \tag{2}$$

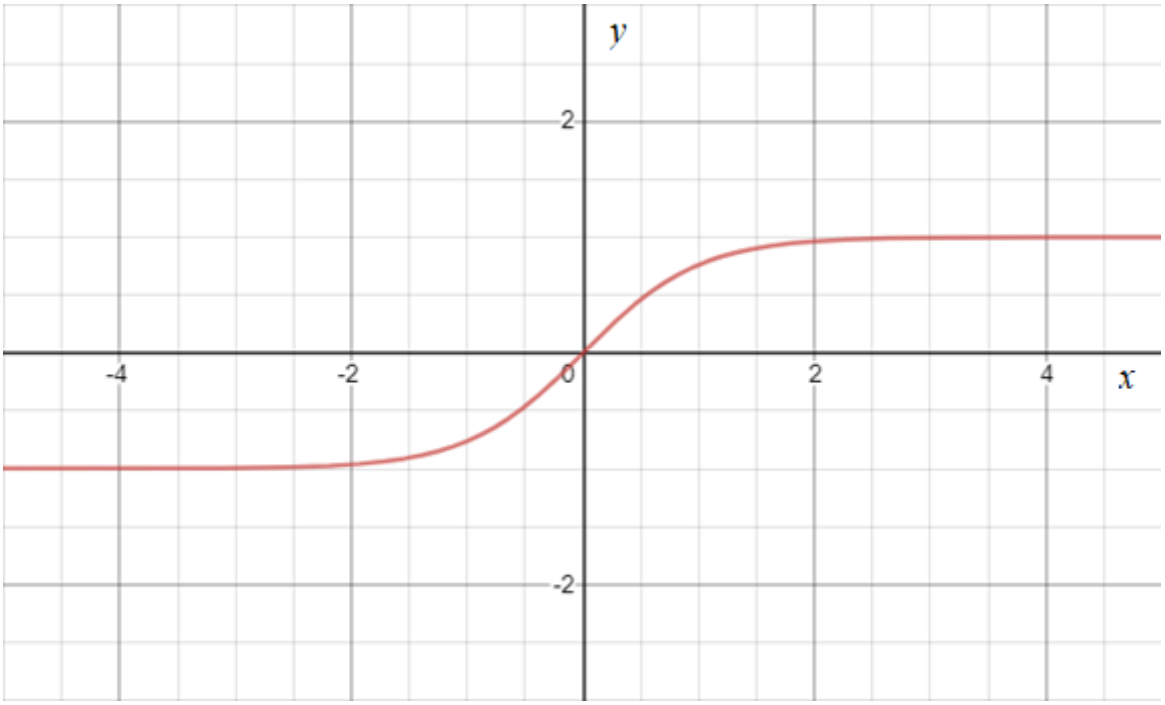


Figure 3.4: Hyperbolic tan function

3.4.2. Rectified linear unit (ReLU) function

The ReLU function is one of the mostly used activation functions of a neural network. It takes an input and outputs a 0 if the input is 0 or negative and outputs the input value unchanged if it is positive. It can be written as:

$$ReLU(x) = \begin{cases} 0, if\ x \leq 0 \\ x, if\ x > 0 \end{cases} \tag{3}$$

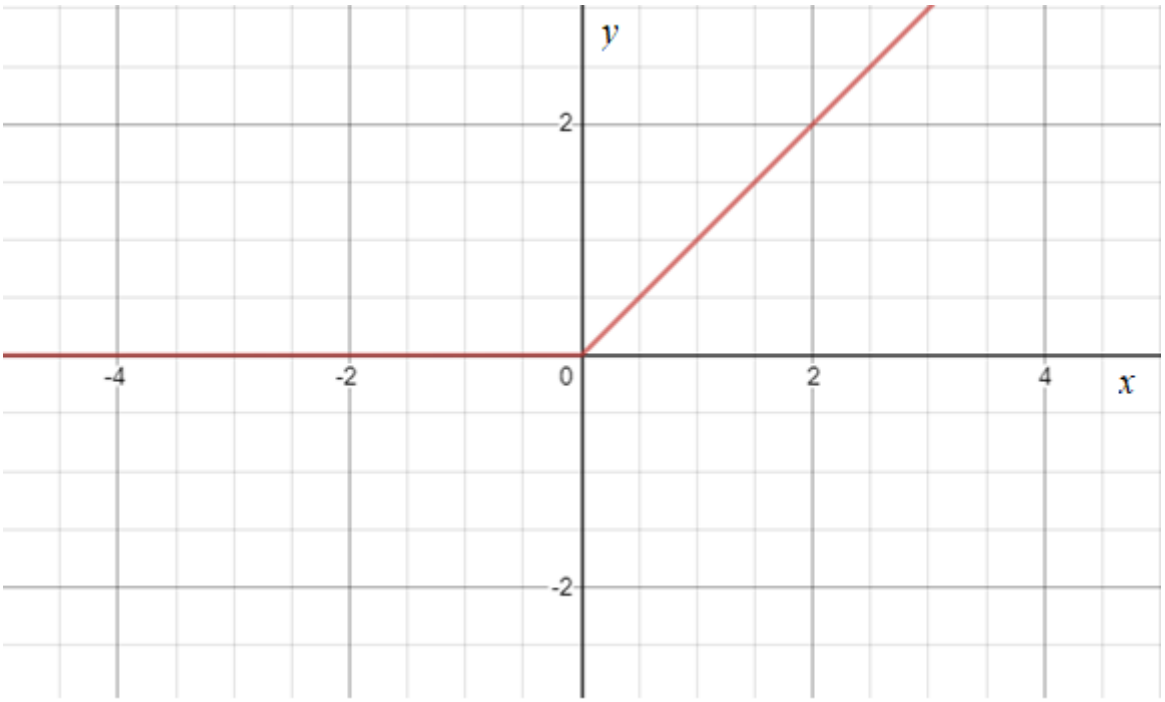


Figure 3.5: ReLU function

ReLU function is used quite often in neural network models as it takes less computational time and power to calculate. But unlike sigmoid or tanh function, ReLU function cannot crunch the data between two numerical values and can generate values ranging from 0 to infinity.

In this research, ReLU function has been used in the inner layers of the network and sigmoid function has been used in the outer layers of the network.

3.5. Neural Network Learning

To make a neural network learn involves setting known inputs and outputs to the neural network, assigning random values to the weights and biases and using an algorithm

called the back-propagation algorithm to tune the weights in such a way that the known inputs can produce the known outputs as accurately as possible.

3.5. Learning Rate

The rate at which the network learns is also another parameter called the "learning rate" and it is regarded as the most important hyper-parameter of a neural network architecture. A learning rate too small will result in fine-tuning of the weights but will take a very long time. A learning rate too high will result in faster training but the results will tend to have more errors. The objective is to find an optimum learning rate, which will enable the neural network model to learn accurately as fast as possible. In this research, two different learning methods were used to compare the outputs that they generated.

3.6. Neural Network Testing

Neural networks are usually tested with some input values unknown to them to see what output they can generate. This determines the accuracy of the neural networks. The methodology used in this research involves a similar procedure. Known inputs based on the shape and location of simple objects placed inside the lid-driven cavity and known output velocities generated using CFD algorithms are used to train the neural networks. They are then tested with unknown set of geometries the network never has trained with and the accuracies are measured which determines the effectiveness of the neural networks.

CHAPTER FOUR

LID-DRIVEN CAVITY

4.1 Foundation

The Navier-Stokes equations are a collection of partial differential equations that describe how viscous fluids move. They are named after Claude-Louis Navier, a French engineer and physicist, and George Gabriel Stokes, an Anglo-Irish physicist and mathematician. The Navier–Stokes equations mathematically express conservation of momentum and mass in Newtonian fluids. Typically, a state equation relating pressure, temperature, and density is used.

The lid-driven cavity problem is a well-known benchmark CFD problem involving incompressible viscous fluid flow. It has a relatively simple two-dimensional geometry consisting of unit lengths, widths, and unit velocities that can be altered in many different ways to produce different results or different flow patterns. This problem has been solved using both laminar flow and turbulent flow. Many different numerical techniques developed by many individuals have been used to compute these solutions. This is a nice problem for testing for several reasons. First, as mentioned above, there is a good amount of literature to compare with. Second, the laminar solution is steady. Third, the boundary conditions are simple and compatible with most numerical approaches. A double lid-driven cavity with some internal square obstacles has been used as the environment for this research. Starting with a single lid-driven cavity and no internal obstacles, the following section will give a brief illustration of the chosen environment and the mathematical model behind its solution using the Navier-Stokes equations.

4.2. Single Lid-driven Cavity

4.2.1. Basics

The single-lid driven cavity problem has been used for a long time to test or validate new codes incorporating new solution methods. The geometry is a simple two-dimensional square cavity having unit lengths. The top side is moving with a unit velocity (in +X direction) and all other sides are stationary. The boundary conditions are showed in Figure 4.1. Fluid exists inside the cavity and the fluid particles are given motion due to the one directional movement of the top lid. Keeping the velocities and the size of the cavity fixed and by using different Reynolds numbers, it is possible to use just the Navier-Stokes equation and a simple computer program to visualize fluid flow inside it.

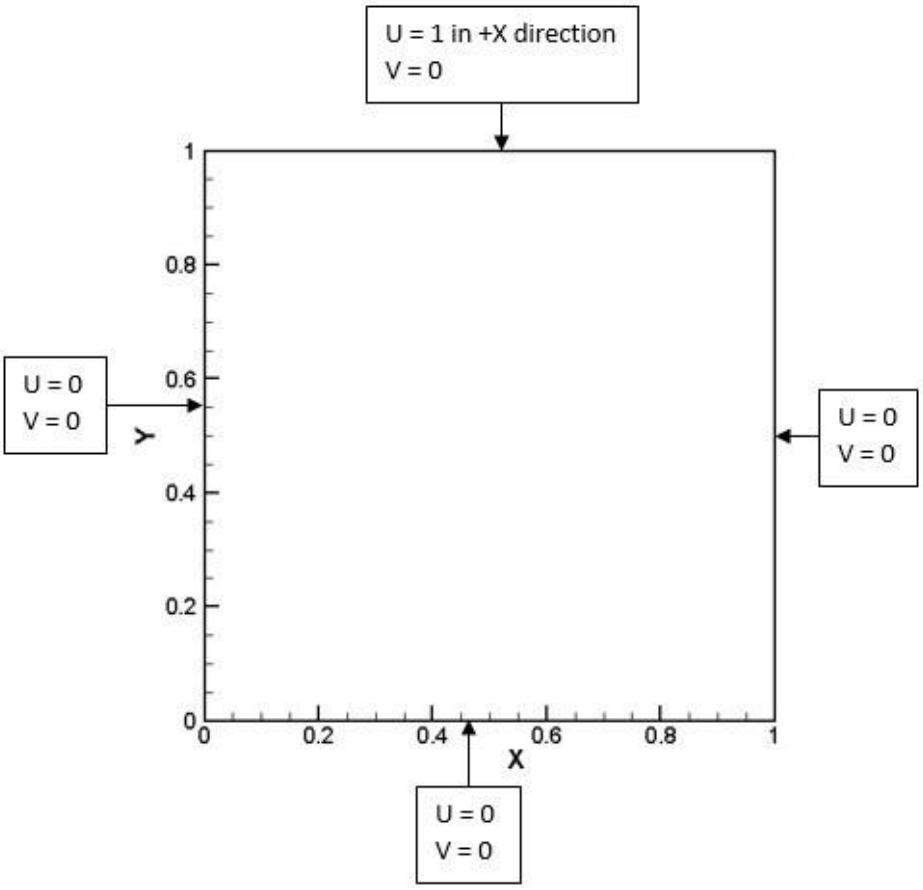
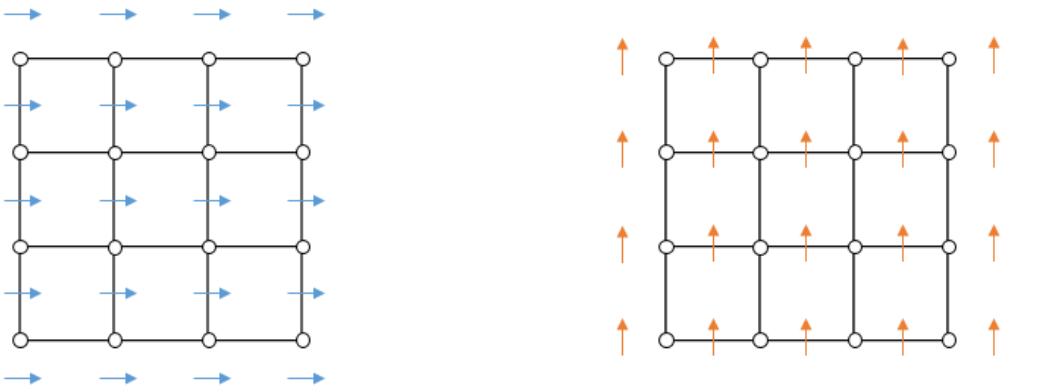


Figure 4.1: Single lid-driven cavity

Using Reynolds Number (Re) 100, 400 and 1000, Marchi et al. [7] provided numerical solutions with a 1024 × 1024 grid system. For this research, Reynolds number has been kept constant at 100 and using three staggered grid systems (24 × 24, 32 × 32 and 40 × 40) and incorporating the Navier-Stokes equation of x-momentum, y-momentum and continuity equation as shown by Chorin [8], x-directional velocity (U) and y-directional velocity (V) have been calculated. Since this is an iterative approach involving time derivative and the continuity equation does not contain a parameter for time, artificial compressibility method was employed, which incorporates a fictitious time derivative of pressure by adding it to the continuity equation and enables the set of equations modified from the incompressible Navier-Stokes equations to be solved implicitly by marching in pseudo time. However, the original equations are recovered when a steady-state solution is reached.

4.2.2. Staggered grids

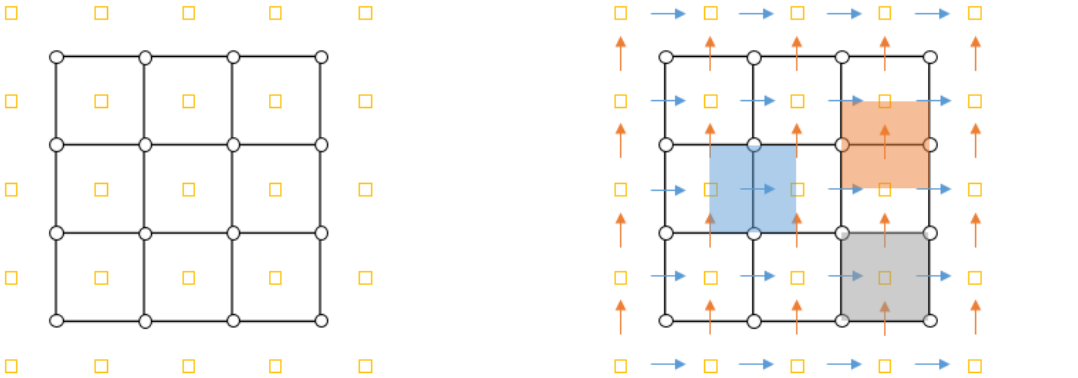
Staggering of grids involves calculating the velocities and pressures around actual nodal points of the grid. This is a common practice when dealing with calculating the nodal velocities using a grid system and the Navier-Stokes equations. To explain how this works, figure 4.2 and 4.3 shows a 4 × 4 grid system. In figure 4.2, the x-directional (U) velocities (blue arrows) and y-directional (V) velocities (orange arrows) are initialized and calculated around actual nodal points. In figure 4.3, pressure values (yellow boxes) are also initialized and calculated around actual nodal points. The combination of these three node variables are shown in figure 4.3. This is done so that the x and y-directional flows can be treated as purely pressure driven flows as each nodal point of U or V is surrounded by two pressure points.



4 × 4 grid with staggered U velocities

4 × 4 grid with staggered V velocities

Figure 4.2: Staggered grid system with U and V velocities



4 × 4 grid with staggered pressure points

4 × 4 grid with velocities and pressure points

Figure 4.3: Staggered grid system with pressure points (P) and combination of U, V and P

The equation for Reynolds number can be written as:

$$Re = \frac{\rho u L}{\mu} \tag{4}$$

Since this study involves the lid-driven cavity having unit lid velocity, in order to keep computations simple, 100 was used as the Reynolds number, which can be derived from

the aforementioned equation having set the values of ρ, u, L and μ to be equal to 1, 1, 1 and 0.01 respectively. This simplifies the model.

The continuity equation can be written as:

$$\frac{\partial \rho}{\partial t} + \nabla \cdot (\rho u) = 0 \tag{5}$$

Assuming the fluid having unit density, the continuity equation involving the artificial compressibility method can be written as:

$$\frac{1}{\delta}\frac{\partial p}{\partial t} + \frac{\partial u}{\partial x} + \frac{\partial v}{\partial y} = 0 \tag{6}$$

The terms can be expanded with a finite difference method as follows using a magnified image (figure 4.4) of the region highlighted in gray in Figure 4.3.

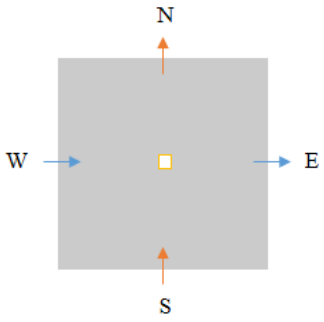


Figure 4.4: Continuity equation terms expansion

$$\frac{1}{\delta}\frac{\partial p}{\partial t} = \frac{1}{\delta}\frac{p_{new} - p}{dt} \tag{6a}$$

$$\frac{\partial u}{\partial x} = \frac{u_E - u_W}{dx} \tag{6b}$$

$$\frac{\partial v}{\partial y} = \frac{v_N - v_S}{dy} \tag{6c}$$

The Navier-Stokes equations control fluid motion and can be thought of as Newton's second law of fluid motion. They were derived by Navier, Poisson, Saint-Venant, and Stokes between 1827 and 1845. The equations can be written as:

$$\underbrace{\rho\left(\frac{\partial u}{\partial t}+u\cdot\nabla u\right)}_{\text{(i)}}=\underbrace{-\nabla p}_{\text{(ii)}}+\underbrace{\nabla\cdot\left(\mu(\nabla u+(\nabla u)^{T})-\frac{2}{3}\mu(\nabla\cdot u)I\right)}_{\text{(iii)}}+\underbrace{F}_{\text{(iv)}} \tag{7}$$

Here, U is the fluid velocity, p is the fluid pressure, ρ is the fluid density, and μ is the fluid dynamic viscosity. The different terms correspond to the inertial forces (i), pressure forces (ii), viscous forces (iii), and the external forces applied to the fluid (iv). For an incompressible flow with the fluid having unit density, the Navier-Stokes x-momentum equation can be written as:

$$\frac{\partial u}{\partial t}+\frac{\partial uu}{\partial x}+\frac{\partial uv}{\partial y}=-\frac{\partial p}{\partial x}+\frac{1}{Re}\nabla^{2}u \tag{8}$$

The terms can be expanded with a finite difference method as follows using a magnified image (figure 4.5) of the region highlighted in blue in figure 4.3.

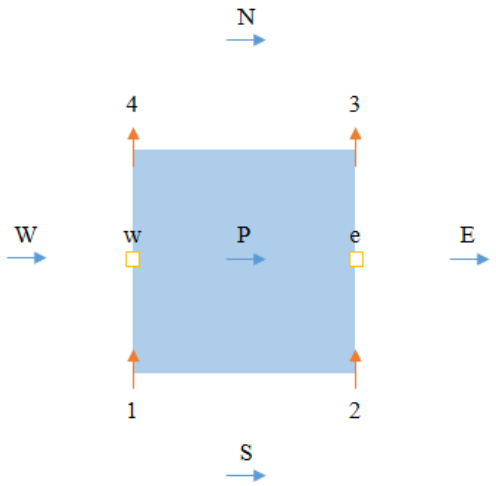


Figure 4.5: Navier-Stokes x-momentum equation terms expansion

$$\frac{\partial u}{\partial t} = \frac{u_{new} - u}{dt} \tag{8a}$$

$$\frac{\partial uu}{\partial x} = \frac{{u_E}^2 - {u_W}^2}{2dx} \tag{8b}$$

$$\frac{\partial uv}{\partial y} = \frac{\left(\frac{u_P + u_N}{2}\right)\left(\frac{v_3 + v_4}{2}\right) - \left(\frac{u_P + u_S}{2}\right)\left(\frac{v_1 + v_2}{2}\right)}{dy} \tag{8c}$$

$$\frac{\partial p}{\partial x} = \frac{p_e - p_w}{dx} \tag{8d}$$

$$\frac{1}{Re}\nabla^2 u = \frac{1}{Re}\frac{u_E - 2u_P + u_W}{dx^2} + \frac{1}{Re}\frac{u_N - 2u_P + u_S}{dy^2} \tag{8e}$$

Similarly, for an incompressible flow with the fluid having unit density, the Navier-Stokes y-momentum equation can be written as:

$$\frac{\partial v}{\partial t} + \frac{\partial uv}{\partial x} + \frac{\partial vv}{\partial y} = -\frac{\partial p}{\partial y} + \frac{1}{Re}\nabla^2 v \tag{9}$$

The terms can be expanded with a finite difference method as follows using a magnified image (figure 4.6) of the region highlighted in orange in figure 4.3.

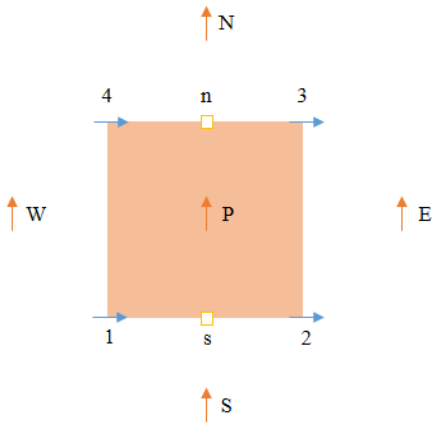


Figure 4.6: Navier-Stokes y-momentum equation terms expansion

$$\frac{\partial v}{\partial t} = \frac{v_{new} - v}{dt} \tag{9a}$$

$$\frac{\partial vv}{\partial y} = \frac{{v_N}^2 - {v_S}^2}{2dy} \tag{9b}$$

$$\frac{\partial uv}{\partial x} = \frac{\left(\frac{v_P + u_W}{2}\right)\left(\frac{u_1 + u_4}{2}\right) - \left(\frac{v_P + u_E}{2}\right)\left(\frac{u_2 + u_3}{2}\right)}{dy} \tag{9c}$$

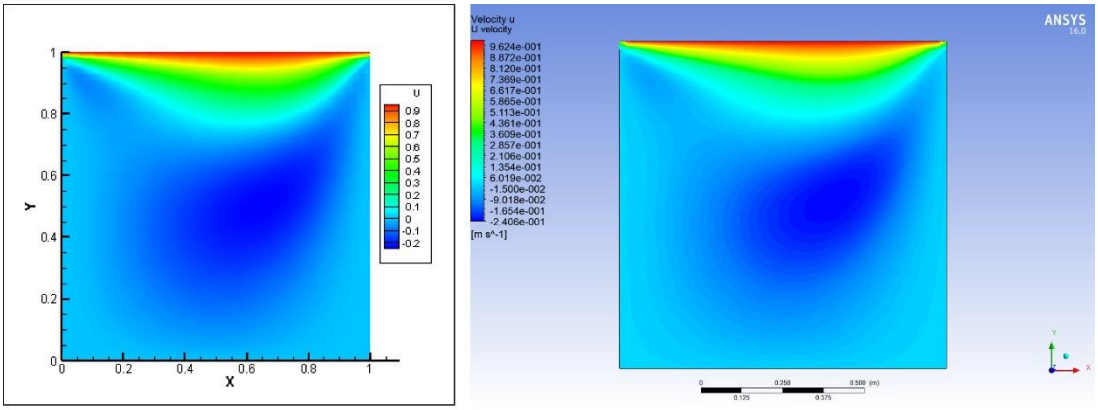
$$\frac{\partial p}{\partial y} = \frac{p_n - p_s}{dy} \tag{9d}$$

$$\frac{1}{Re}\nabla^2 v = \frac{1}{Re}\frac{v_E - 2v_P + v_W}{dx^2} + \frac{1}{Re}\frac{v_N - 2v_P + v_S}{dy^2} \tag{9e}$$

4.2.3. Solution using a C# code

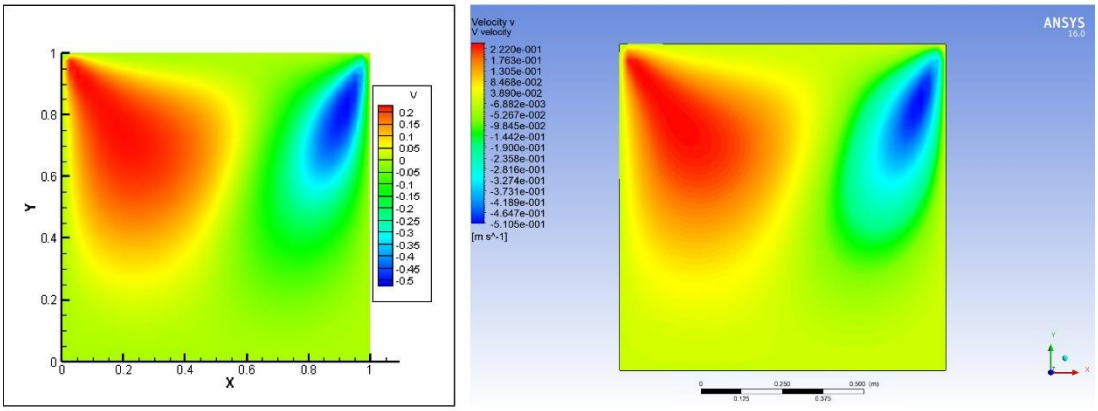
By entering appropriate initial and boundary conditions and using these equations, values of u_{new}, v_{new} and p_{new} were calculated for this research with a C# code by an iterative approach employing the values of dt and δ as 0.001 and 4.5 respectively. Computations were terminated when summation of absolute relative errors between the adjacent x and y-directional velocities dropped below 0.00001. Average of two adjacent U velocities were taken and actual nodal U velocities were calculated. The same was done with V velocities. For calculating pressure values, average of four values were taken and actual nodal pressure values were calculated. Tecplot software was used to view the results obtained from the CFD code. Commercially used ANSYS software was used to validate the results obtained from the CFD code. Figure 4.7 compares U and V velocities obtained from the C# code and ANSYS software. It is clear that the output from the C# code is almost the same as the output obtained from ANSYS software. Some minor variations can be seen here and there but these are mainly because of small differences in the color profiles set by Tecplot and

ANSYS. C# code was used instead of ANSYS software to generate fluid data because the neural network algorithms were integrated into the same C# code and a single C# program did the entire dataset generation, training and testing for this research.



U (x-directional) velocity obtained from CFD code using 40 × 40 grid

U (x-directional) velocity obtained from ANSYS using 40 × 40 grid



V (y-directional) velocity obtained from CFD code using 40 × 40 grid

V (y-directional) velocity obtained from ANSYS using 40 × 40 grid

Figure 4.7: Single lid-driven cavity flow with C# CFD code and ANSYS

4.3. Double Lid-driven Cavity

In a similar way, using the same set of equations as mentioned earlier, the C# code was used to simulate fluid flow inside a double lid driven cavity where the top and bottom sides are moving with a unit velocity (in +X direction) and all other sides are stationary. In this setup, the fluid inside the cavity is moved by two moving lids instead of only one as seen in the previous single lid-driven cavity. The two moving lids adds a bit more complexity to the way how fluid flows inside it. The boundary conditions are showed in Figure 4.8. Mawarsih et al. [9] have used top and bottom moving lids to simulate fluid flow inside it using staggered grids. Saha [10] used a double lid-driven cavity with walls moving in different directions to predict fluid flow inside it.

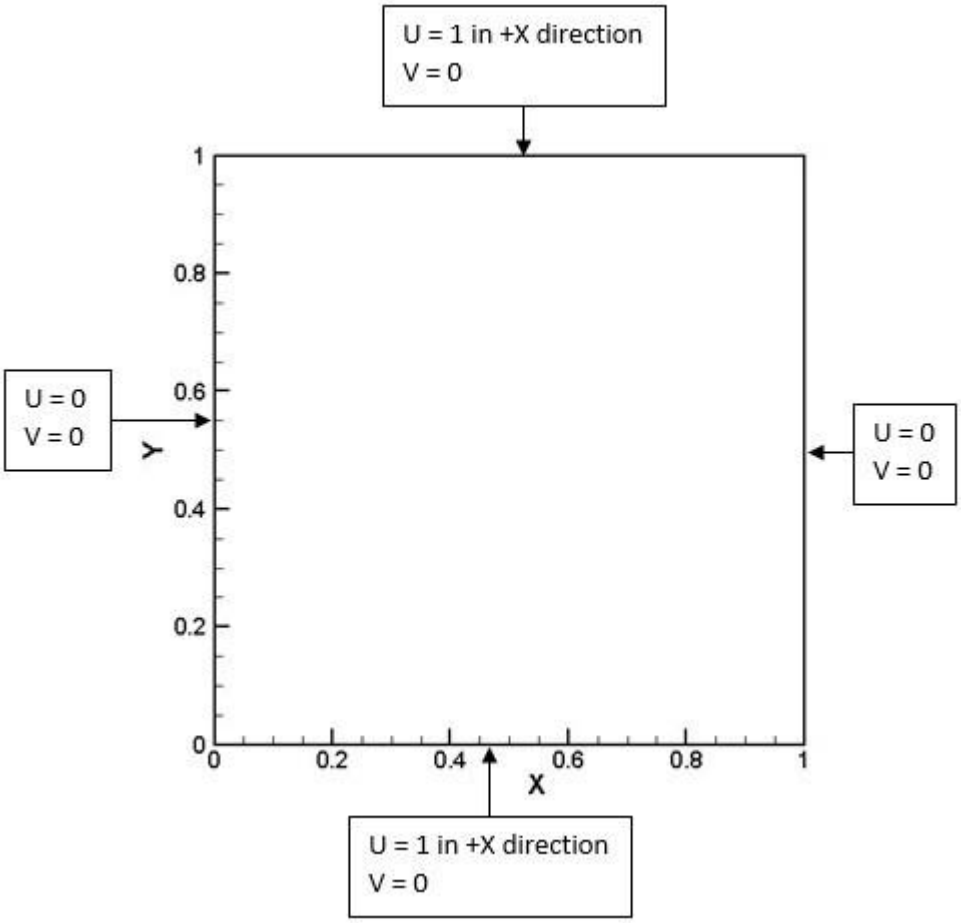
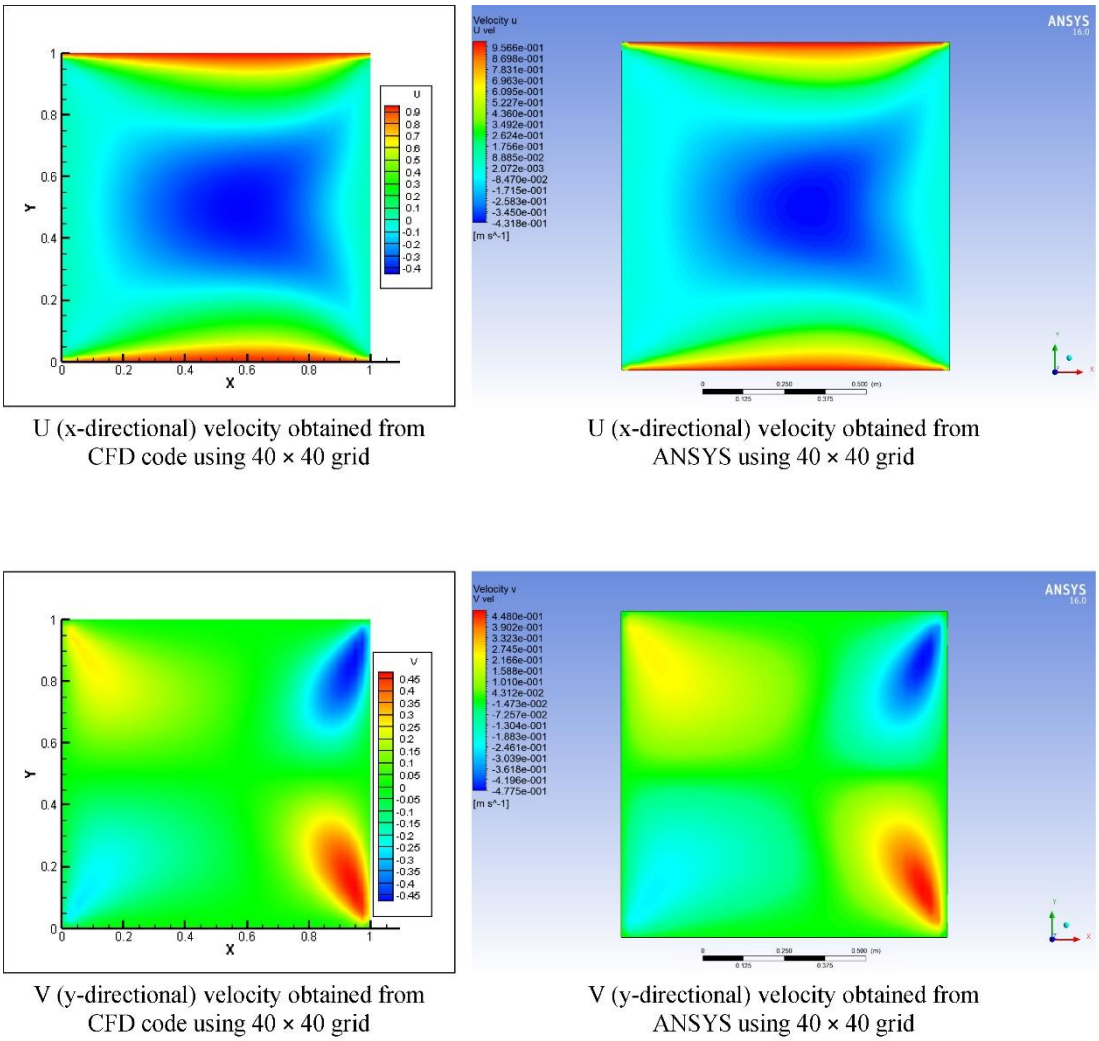


Figure 4.8: Double lid-driven cavity

Figure 4.9 compares U and V velocities obtained from the C# code and ANSYS software. It is again clear that the output from the C# code is almost the same as the output obtained from ANSYS software. Some minor variations can also be seen here and there but these are mainly because of small differences in the color profiles set by Tecplot and ANSYS.



U (x-directional) velocity obtained from CFD code using 40 × 40 grid

U (x-directional) velocity obtained from ANSYS using 40 × 40 grid

V (y-directional) velocity obtained from CFD code using 40 × 40 grid

V (y-directional) velocity obtained from ANSYS using 40 × 40 grid

Figure 4.9: Double lid-driven cavity flow with C# CFD code and ANSYS

4.4. Double Lid-driven Cavity with an Internal Square Obstacle

This research involves making a neural network learn fluid flow patterns. In order to achieve such a thing, different data generated using different setups are needed, which have some similarities between them and patterns can be found among them. A single lid driven cavity is just a plain tool for assessing newly discovered numerical methods to simulate fluid flow inside it. A double lid-driven cavity is an updated version of the single lid-driven cavity as it makes the environment a little bit more complex. To keep the environment simple but still get different datasets for the neural network to learn, a double lid-driven cavity with some internal square objects can be used as a training data because it adds a lot more complexity to the simple single lid-driven cavity problem as well as the double lid-driven cavity problem and it also enables different datasets to be generated by placing the square objects in different positions inside the cavity allowing the neural network to actually learn what effects the placement of the square object will have on the fluid flow patterns generated by the two moving top and bottom lids.

Again in a similar way, the same C# code was used to simulate fluid flow inside a double lid driven cavity with an obstacle occupying 1/64th of the total cavity area placed inside the cavity where the top and bottom sides of the cavity were moving with a unit velocity (in +X direction) and all other sides were stationary. The obstacle placed inside the cavity acted as a resistance to the flow generated by the moving lids as no slip conditions exists on the sides of the square obstacle. The whole purpose of this was to see if it is possible to achieve prediction of fluid flow around shapes created by stacking together multiple square objects, therefore creating a relatively complex object. Huang and Lim [11] performed simulation of lid-driven cavity flow with internal circular obstacles. Circular

objects were not selected, as they can be stacked together without leaving behind empty spaces between them. Figure 4.10 show how the cavity can be divided into 64 segments and how the square object can be placed inside it in different positions as illustrated by the horizontal and vertical lines that make up the segmentations. The figure also shows the boundary conditions for this setup. The reason the environment was chosen to be divided into 64 segments is because of the different grid sizes used in this research. The three different grid sizes that had been used had a common factor of 8. And so, (8 × 8) or 64 is the number of segments that is needed in order to be able to place the object uniformly in the different grid setups allowing them to have the exact same placement of the object that coincided perfectly with the grid points of the separate grid systems.

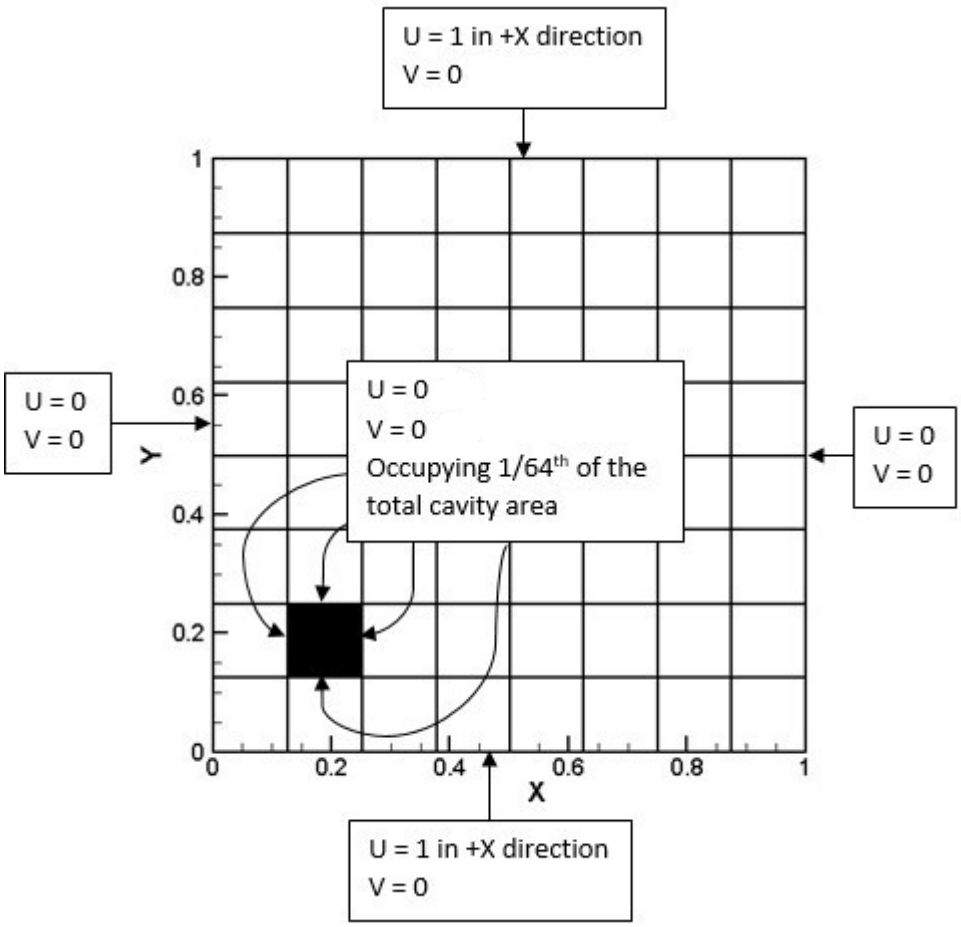
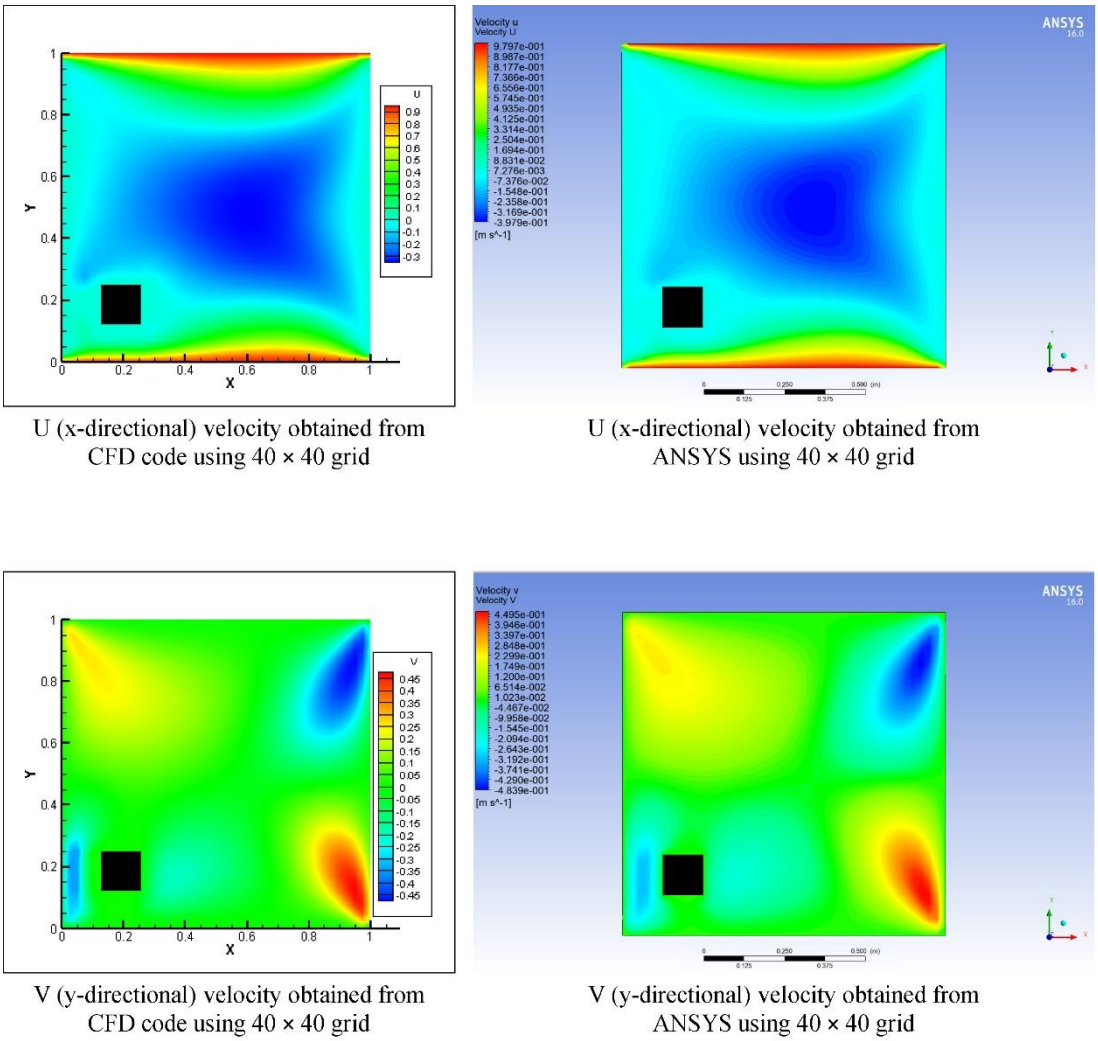


Figure 4.10: Double lid-driven cavity with an obstacle

Figure 4.11 compares U and V velocities obtained from the C# code and ANSYS software. In addition, it is again clear that the output from the C# code is almost the same as the output obtained from ANSYS software. Again, some minor variations can be seen here and there but these are mainly because of small differences in the color profiles set by Tecplot and ANSYS.



U (x-directional) velocity obtained from CFD code using 40 × 40 grid

U (x-directional) velocity obtained from ANSYS using 40 × 40 grid

V (y-directional) velocity obtained from CFD code using 40 × 40 grid

V (y-directional) velocity obtained from ANSYS using 40 × 40 grid

Figure 4.11: Double lid-driven cavity flow having a square obstacle with C# CFD code and ANSYS

CHAPTER FIVE

CFD DATASET GENERATION

5.1. Grid Setup

The chosen double lid-driven cavity environment was divided into three different grid systems to see if the neural networks remains consistent in predicting accurate values and if a denser grid system can help the neural network in learning more accurately as denser grid systems have more nodal points in the same area thus improving the accuracy of the output from the CFD iterations which are directly used to train the neural networks. To keep the computational efforts relatively low, the following grid systems were used in this research:

1. 24 × 24 grid resulting in 25 × 25 grid points
2. 32 × 32 grid resulting in 33 × 33 grid points
3. 40 × 40 grid resulting in 41 × 41 grid points

As mentioned earlier, the square object was placed inside the cavity such that it occupied 1/64th of the total cavity area. By dividing the cavity area into 64 segments, it was possible to place the square object in 36 different places, which covered most of the cavity region. The plan was to place the square object in these places and train the neural network with only these 36 input-output datasets so that complex shapes made up with these square objects could be used to test the neural network. Using three different grid sizes, 3 CFD datasets each having 36 input-output pairs were generated for the neural networks to train.

5.2. Dataset Generation using 24 × 24 Grid System

24 × 24 grids can be divided into 64 segments (8 × 8), each made up of 3 grids as shown in figure 5.1. The square object occupying 3 × 3 grids can be uniformly placed at 36 different positions. This resulted in the input data for the neural network, which has 36 datasets each consisting of 81 0's and 1's reflecting the position of the square object inside the cavity. Figure 5.1 shows how placing the object in the middle of the cavity can be interpreted by these 0's and 1's. Figure 5.1 also shows the placement of the object in 36 different locations. Using the C# code, as shown earlier, fluid velocities can be calculated for each of the 36 setups. The corresponding output data for the neural network consisted of 625 (25^2) pairs of nodal U and V velocities obtained from the C# CFD code. This enabled the neural network to establish a relation between the placement of the object inside the cavity and the flow pattern associated with it. So the input datasets for each of the two directional velocities of this grid system for the neural network had 36 lines, where each line had 91 0's and 1's. The target datasets each also had 36 lines having 91 velocities in each line.

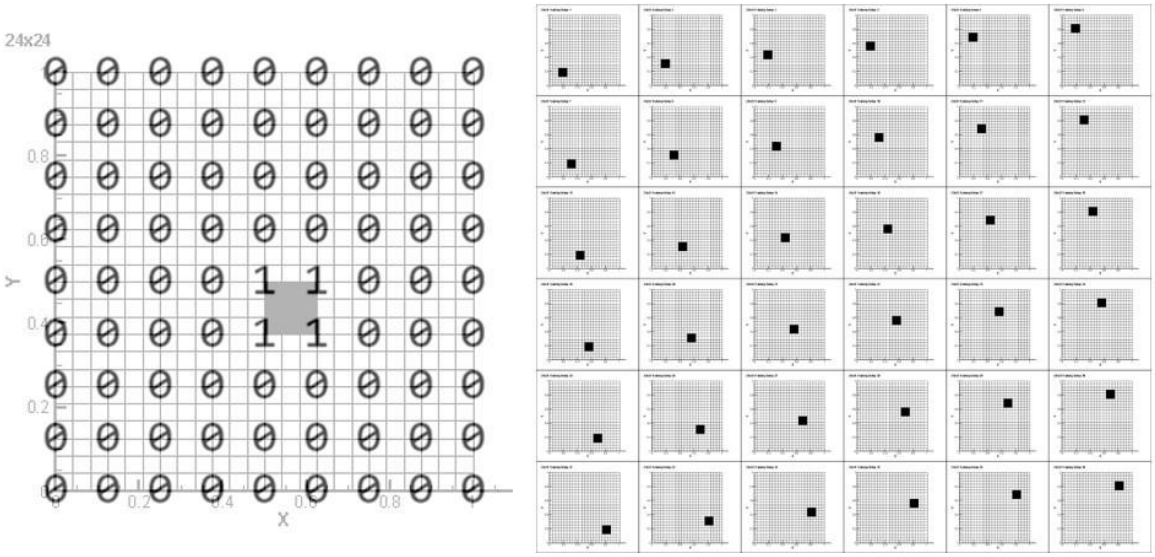


Figure 5.1: Environment segmentation for 24 × 24 grid system

5.3. Dataset Generation using 32 × 32 Grid System

In a similar way, 32 × 32 grids can be divided into 64 segments (8 × 8), each made up of 4 grids as shown in figure 5.2. The square object occupying 4 × 4 grids can be uniformly placed at 36 different positions. Similar to the previous setup, this resulted in the input data for the neural network, which has 36 datasets each consisting of 81 0's and 1's reflecting the position of the square object inside the cavity. The corresponding output data for the neural network consisted of 1089 (33^2) pairs of nodal U and V velocities obtained from the C# CFD code.

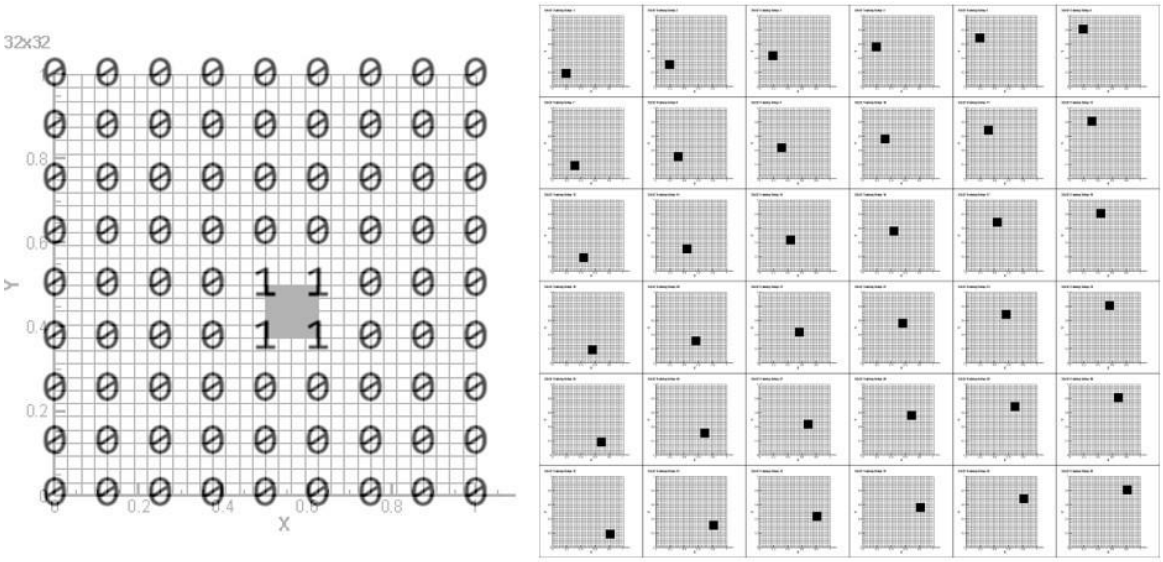


Figure 5.2: Environment segmentation for 32 × 32 grid system

5.4. Dataset Generation using 40 × 40 Grid System

Again, in a similar way, 40 × 40 grids can be divided into 64 segments (8 × 8), each made up of 5 grids as shown in figure 5.3. The square object occupying 5 × 5 grids can be uniformly placed at 36 different positions. Again, identical to the previous two setups, this resulted in the input data for the neural network, which has 36 datasets each consisting of

81 0's and 1's reflecting the position of the square object inside the cavity. The corresponding output data for the neural network consisted of 1681 (41^2) pairs of nodal U and V velocities obtained from the C# CFD code.

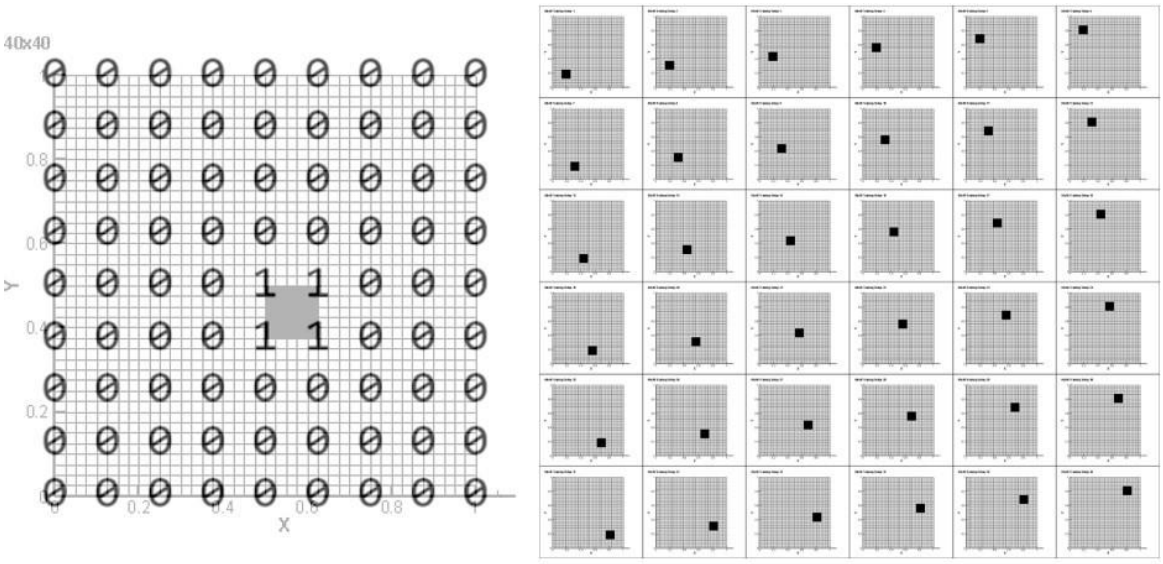


Figure 5.3: Environment segmentation for 40 × 40 grid system

CHAPTER SIX

ANN MODEL AND TRAINING

6.1. Network Structure

Figure 6.1 shows the schematic of the neural network model that has been used in this research. ReLU activation function has been used in the first and second hidden layer and Sigmoid activation function has been used in the output layer. ReLU is used because it is fast and easier to calculate. Sigmoid is used in the output layer because it generates an output that lies in between 0 and 1 which perfectly matches the lid-driven cavity's unit lid velocity as nothing inside the cavity will have a velocity more than 1.

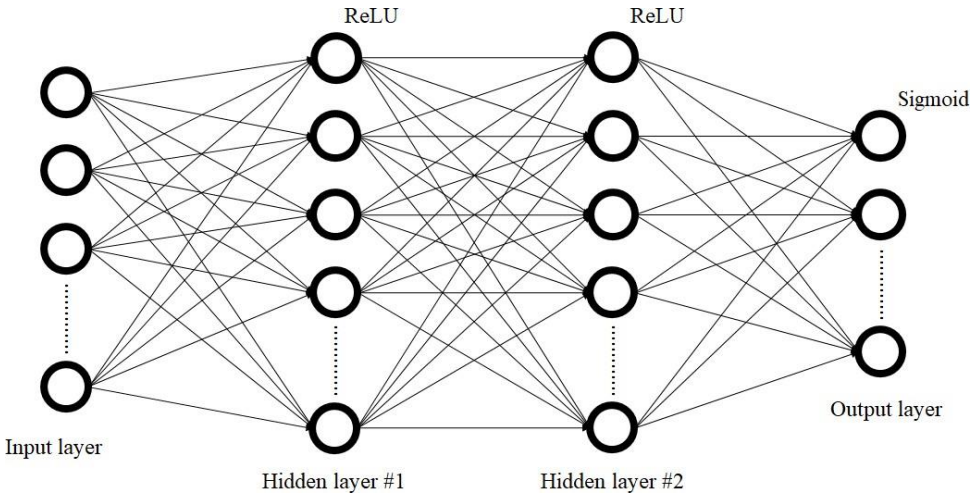


Figure 6.1: Neural network structure

Two different learning methods are also used in this research independently. The first one involves setting a globally fixed (GF) learning rate throughout the entire network. The second one involves using different learning rates in different layers or in other words, layer specific (LS) learning rates.

6.2. Neural Network Training

Separate neural networks have been used for each grid systems. Each neural network has two segments. One for the U velocity and one for the V velocity. There are no connections between the two segments of the neural network. It may be possible to use a single neural network to for all the two directional velocities, but the number of network parameters, (weights and biases) will be increased by an order of a couple magnitudes and thus will lead to unnecessary complications in the overall target of this research. To keep calculations simple and yet achieve a two-dimensional flow prediction, two different networks were used which were trained and tested separately. In each segment of the network, there are 81 input nodes in the input layer, 500 hidden nodes in the next hidden layer (#1) and another 1000 hidden nodes in the next hidden layer (#2).

For 24 × 24 grid system, there are 625 (25^2) output nodes in each of the output layer corresponding to 625 pairs of nodal U and V velocities as shown in figure 6.2.

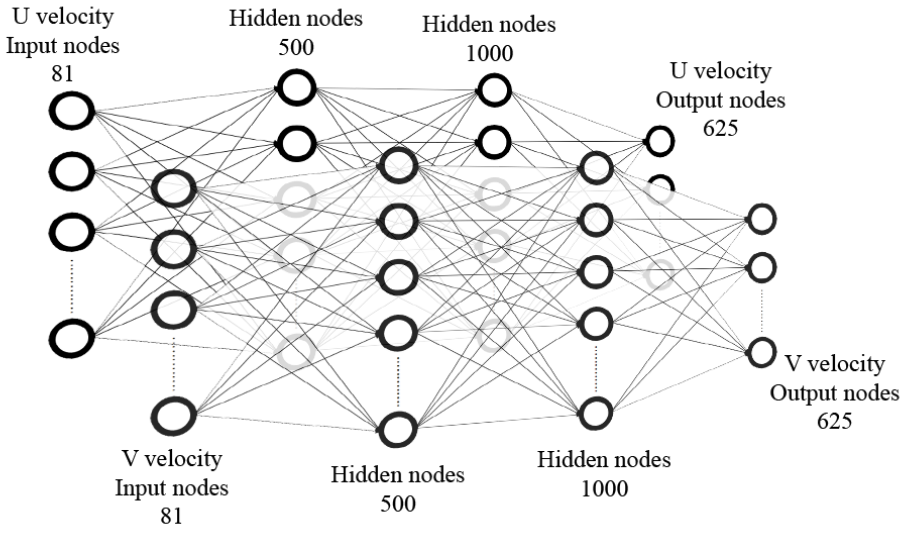


Figure 6.2: Neural network for 24 × 24 grid system

For 32 × 32 grid system, there are 1089 (33^2) output nodes in each of the output layer corresponding to 1089 pairs of nodal U and V velocities as shown in figure 6.3.

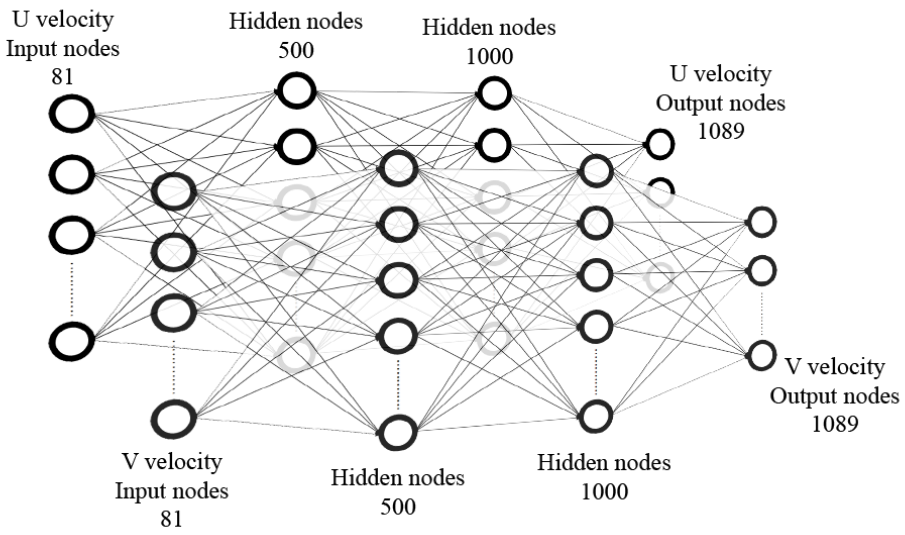


Figure 6.3: Neural network for 32 × 32 grid system

For 40 × 40 grid system, there are 1681 (41^2) output nodes in each of the output layer corresponding to 1681 pairs of nodal U and V velocities as shown in figure 6.4.

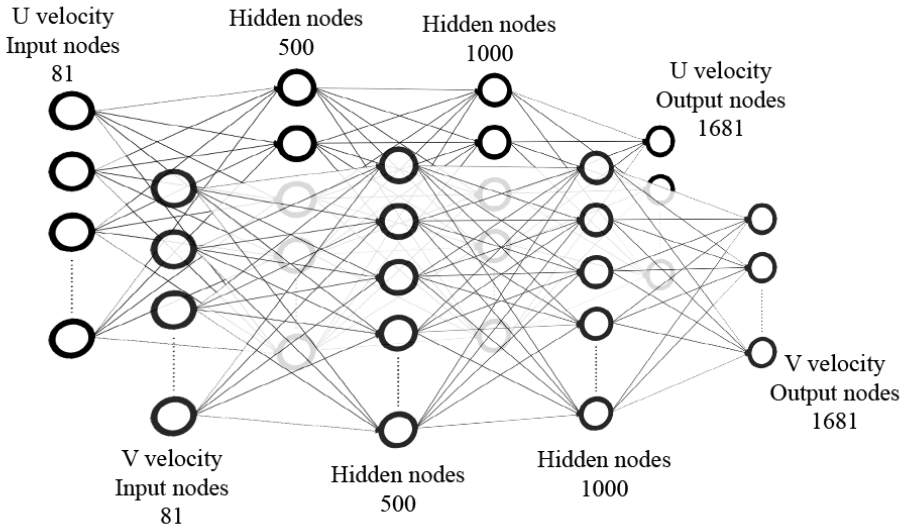


Figure 6.4: Neural network for 40 × 40 grid system

For the 36 different setups in each of the three different grid systems, the neural networks were trained with 10000 epochs and a globally fixed learning rate (GF) of 0.02 for all the layers. The networks were again trained with layer specific learning rates (LS) of 0.002 for the second layer, 0.02 for the third layer and 0.2 for the output layer. Smaller learning rates were used in the input layers because they have to learn more accurately as opposed to the outer layers where they have to cope with the expected output from the input-output training dataset and so they have to be changed a lot quite frequently. In other words, the output layers need to learn faster than the input layers, and so the learning rates have been kept large there. This is perfectly reflected in figure 6.5 where mean squared errors (MSE) obtained from these two learning methods are shown for the 10000 epochs. Table 6.1 shows the terminologies used in figure 6.5.

Table 6.1: Terminologies used in figure 6.5

Term	**Meaning**
MSE 24 × 24 GF	Mean squared error in 24 × 24 grid system incorporating globally fixed learning rates
MSE 24 × 24 LS	Mean squared error in 24 × 24 grid system incorporating layer specific learning rates
MSE 32 × 32 GF	Mean squared error in 32 × 32 grid system incorporating globally fixed learning rates
MSE 32 × 32 LS	Mean squared error in 32 × 32 grid system incorporating layer specific learning rates
MSE 40 × 40 GF	Mean squared error in 40 × 40 grid system incorporating globally fixed learning rates
MSE 40 × 40 LS	Mean squared error in 40 × 40 grid system incorporating layer specific learning rates

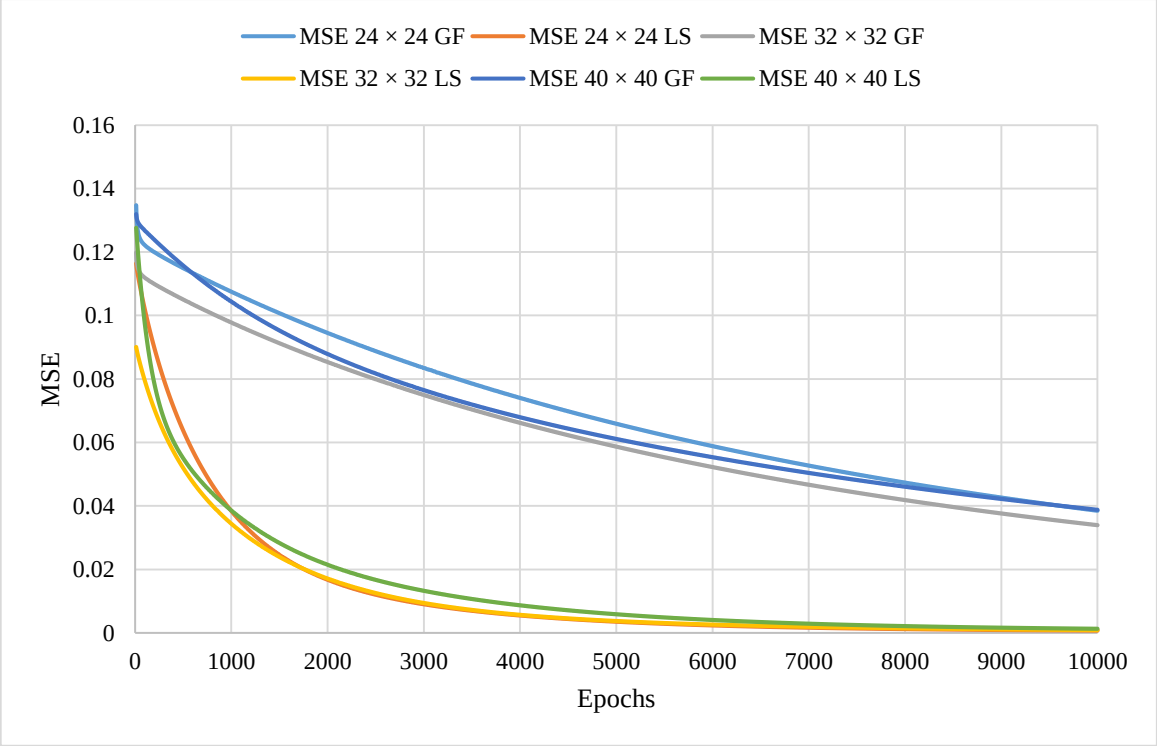


Figure 6.5: MSE comparison between GF and LS learning rates in all three grid systems

To sum it all up, a total number of six neural networks have been trained by incorporating the three different grid sizes and the two different learning approaches used. As each neural network had two segments for the U and V velocities, they generated different mean squared errors. However, they propagated with the number of epochs in a similar fashion. This is why each of the six sets of MSEs shown in figure 6.5 is actually a set of errors calculated by taking the average of the errors generated at each epoch by the two segments of the neural network.

It is clearly observed that the neural network models using LS learning rates were better optimized than GF learning rate, which resulted in the same accuracy achieved by the LS models just after 1000 epochs compared to the GF models' 10000 epochs.

CHAPTER SEVEN

RESULTS: CFD VS ANN

7.1. Preface

During training, only one obstacle was placed at a time inside the double lid-driven cavity. During testing, all trained networks for the three different grid systems each using two different learning methods were tested with all the training data as well as some new setups. New setups were made by putting together several square objects and placing multiple square objects in different places at the same time. The networks were never trained with such complex setups and were tasked to predict fluid flow around them. A good number of tests have been performed and it has been seen that the more complex the shape appeared to get, the more errors the neural network had in predicting the fluid flow. Results of two such test for each of the three different grid setups are shown in detail. Terminologies as mentioned in table 7.1 are used:

Table 7.1: Terminologies used in results

Term	Meaning
CFD	Results from CFD simulation using NS equations
ANN LS	Results from ANN using Layer Specific learning rates
ANN LS Differences	Absolute differences between CFD results and ANN LS results
ANN GF	Results from ANN using Globally Fixed learning rates
ANN GF Differences	Absolute differences between CFD results and ANN GF results
Time (ms)	Approximate time (in milliseconds) required by the computer to generate results (using a 7th generation core i5 with 8GBs of onboard memory)

7.2. First Test Setup

7.2.1. Environment

Two square objects were placed diagonally inside the double lid-driven cavity at the same time. The network was tested with CFD data for a single object placed at a time but how the fluid is going to behave when two objects were placed at the same time was a new challenge for the neural network. The boundary conditions in the cavity was the same as the training phase. The top and bottom walls were kept moving at unit velocities and the cavity had unit length and width. Figure 7.1 shows the arrangement of the square objects inside the cavity. The following pages compares the results of the U and V velocities generated using the three different grid systems incorporating the two different learning rates that have been used.

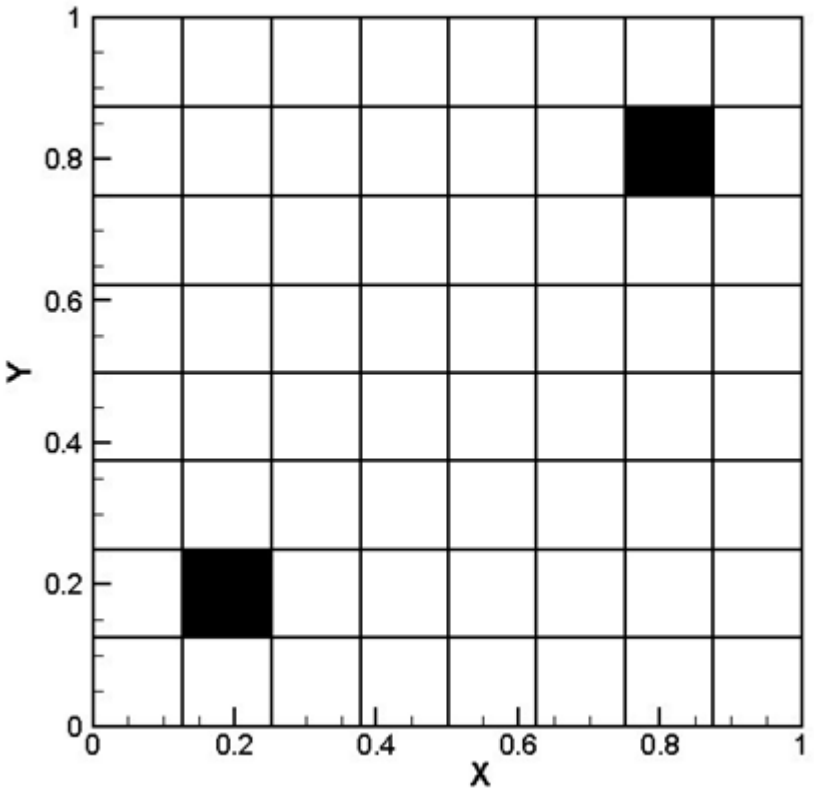


Figure 7.1: Double lid-driven cavity with two obstacles

7.2.2. X-directional (U) velocity prediction

The placement of the two square objects resulted in the following output of U velocity as shown in figure 7.2 from the neural network incorporating GF learning method in 24 × 24 grid system. The first flow field is from the CFD code, the second flow field is from the neural network, the third flow field shows the absolute differences between the two flow fields and the fourth graph shows a quantitative representation of the absolute differences. Some relative errors can be seen along the edges of the square objects. The CFD solution took 5380 milliseconds where the ANN prediction took only 15 milliseconds.

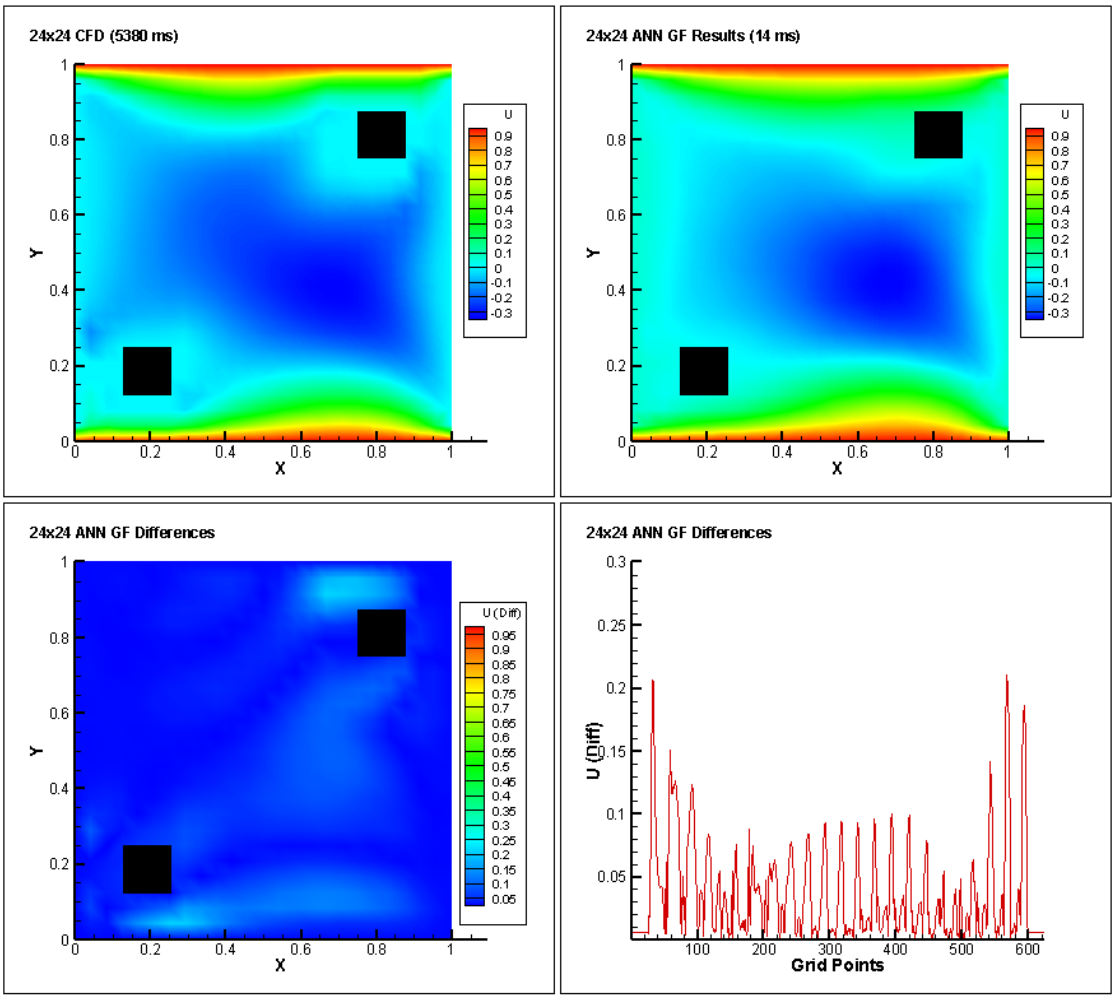


Figure 7.2: Test setup 1 - 24 × 24 grid system ANN GF comparison (U velocity)

The following output of U velocity as shown in figure 7.3 is from the neural network incorporating LS learning method in 24 × 24 grid system. The first flow field is from the CFD code, the second flow field is from the neural network, the third flow field shows the absolute differences between the two flow fields and the fourth graph shows a quantitative representation of the absolute differences. Relative errors in this case are less than that of the GF learning method. Some errors are seen in the central part of the cavity. The CFD solution took 5380 milliseconds where the ANN prediction took only 16 milliseconds.

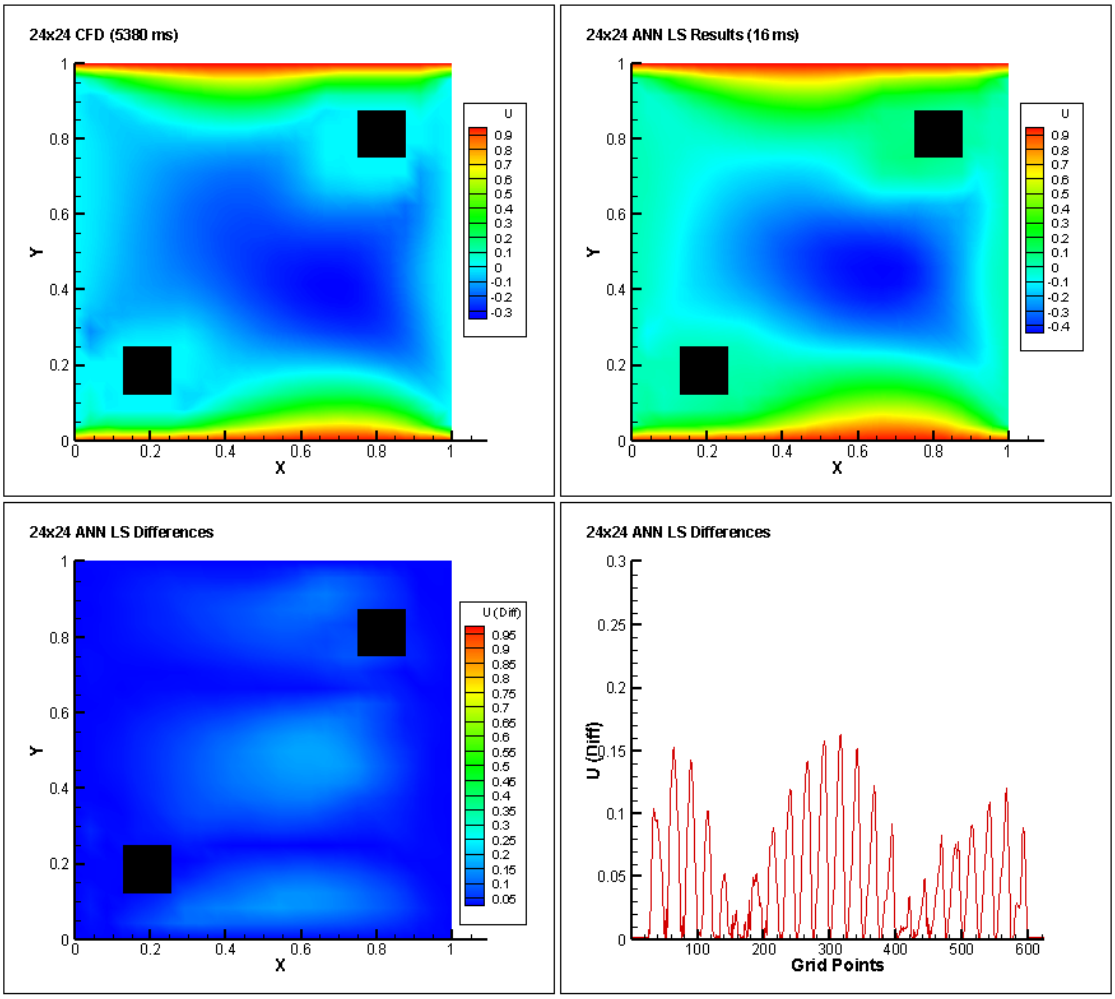


Figure 7.3: Test setup 1 - 24 × 24 grid system ANN LS comparison (U velocity)

The following output of U velocity as shown in figure 7.4 is from the neural network incorporating GF learning method in 32 × 32 grid system. The first flow field is from the CFD code, the second flow field is from the neural network, the third flow field shows the absolute differences between the two flow fields and the fourth graph shows a quantitative representation of the absolute differences. Some relative errors can be seen along the edges of the square objects. The CFD solution took 10774 milliseconds where the ANN prediction took only 15 milliseconds.

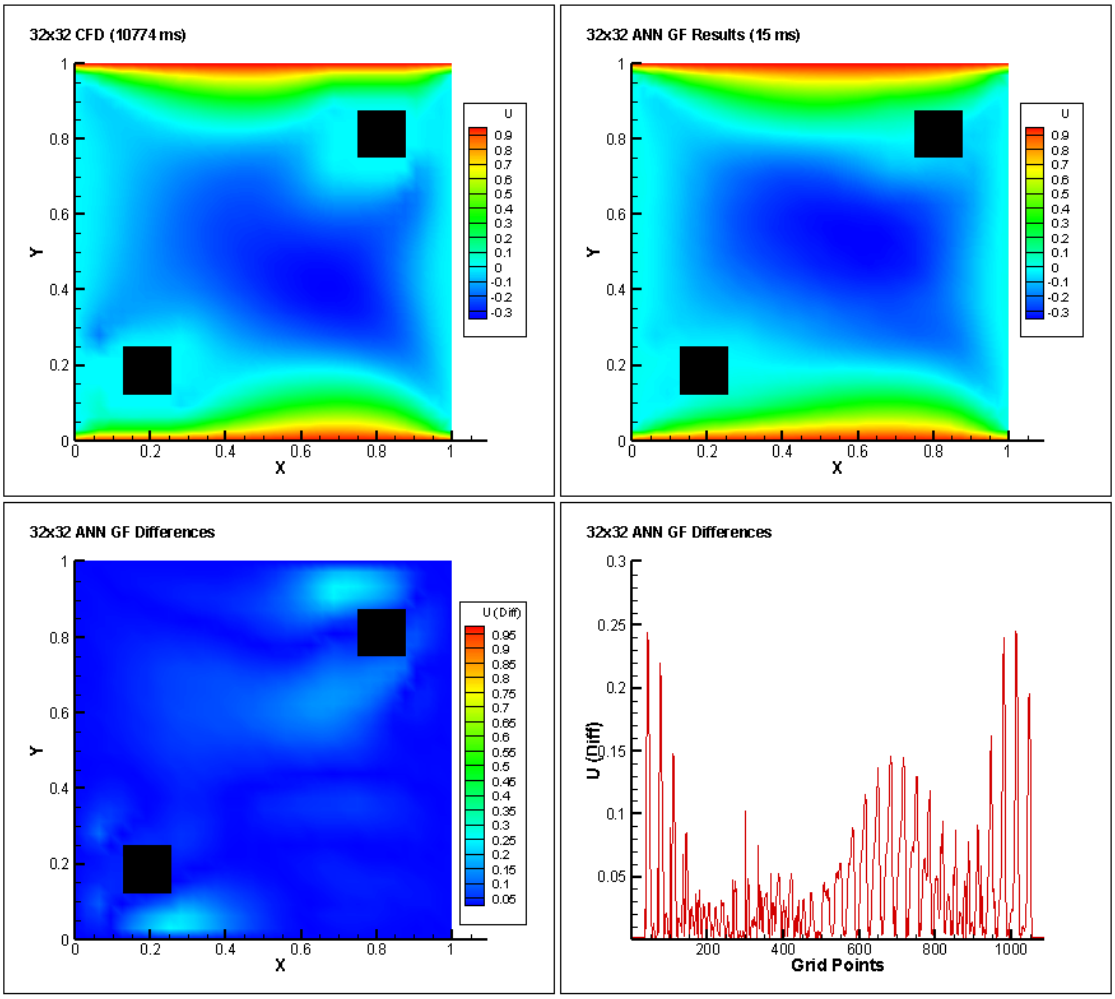


Figure 7.4: Test setup 1 - 32 × 32 grid system ANN GF comparison (U velocity)

The following output of U velocity as shown in figure 7.5 is from the neural network incorporating LS learning method in 32 × 32 grid system. The first flow field is from the CFD code, the second flow field is from the neural network, the third flow field shows the absolute differences between the two flow fields and the fourth graph shows a quantitative representation of the absolute differences. Relative errors in this case are less than that of the GF learning method. Some errors are seen in the central part of the cavity. The CFD solution took 10774 milliseconds where the ANN prediction took only 31 milliseconds.

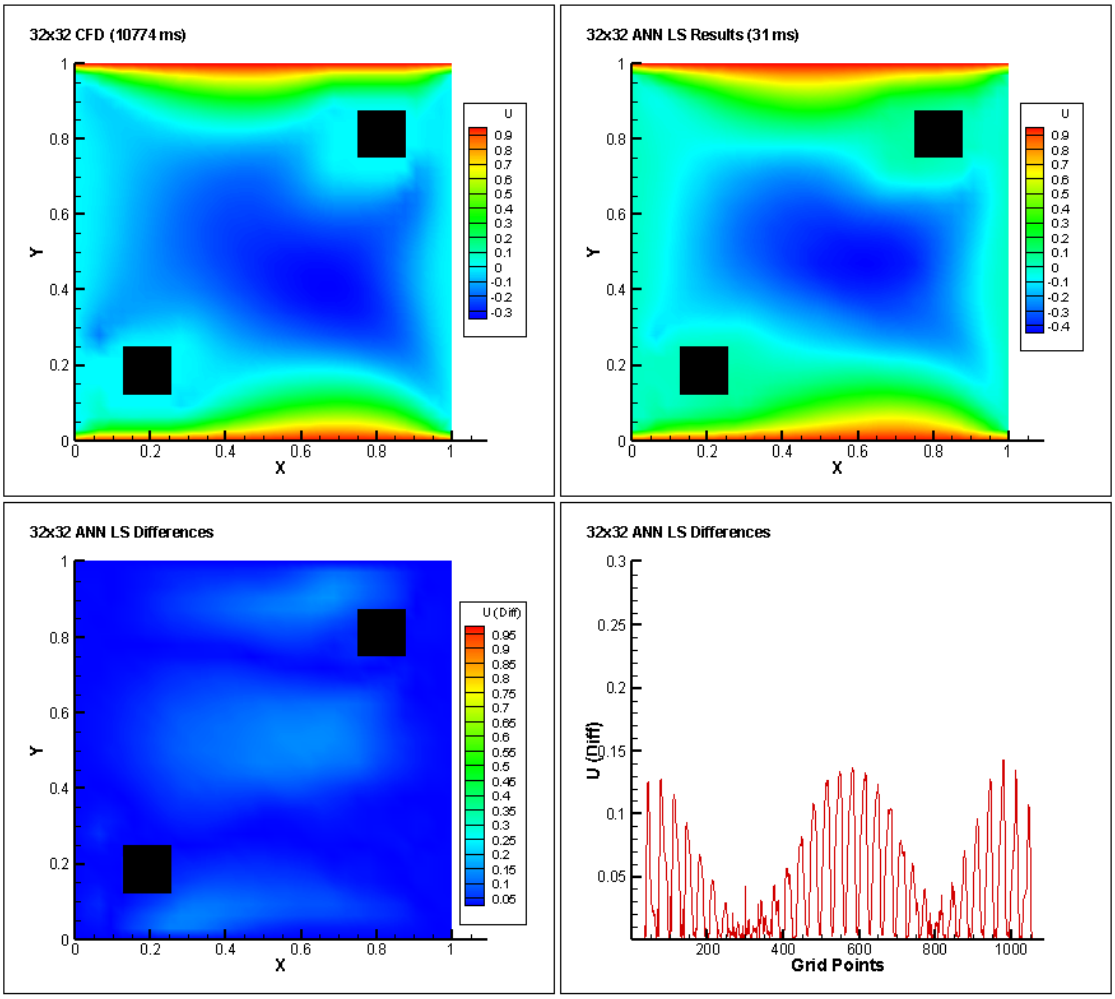


Figure 7.5: Test setup 1 - 32 × 32 grid system ANN LS comparison (U velocity)

The following output of U velocity as shown in figure 7.6 is from the neural network incorporating GF learning method in 40 × 40 grid system. The first flow field is from the CFD code, the second flow field is from the neural network, the third flow field shows the absolute differences between the two flow fields and the fourth graph shows a quantitative representation of the absolute differences. The CFD solution took 16261 milliseconds where the ANN prediction took only 31 milliseconds.

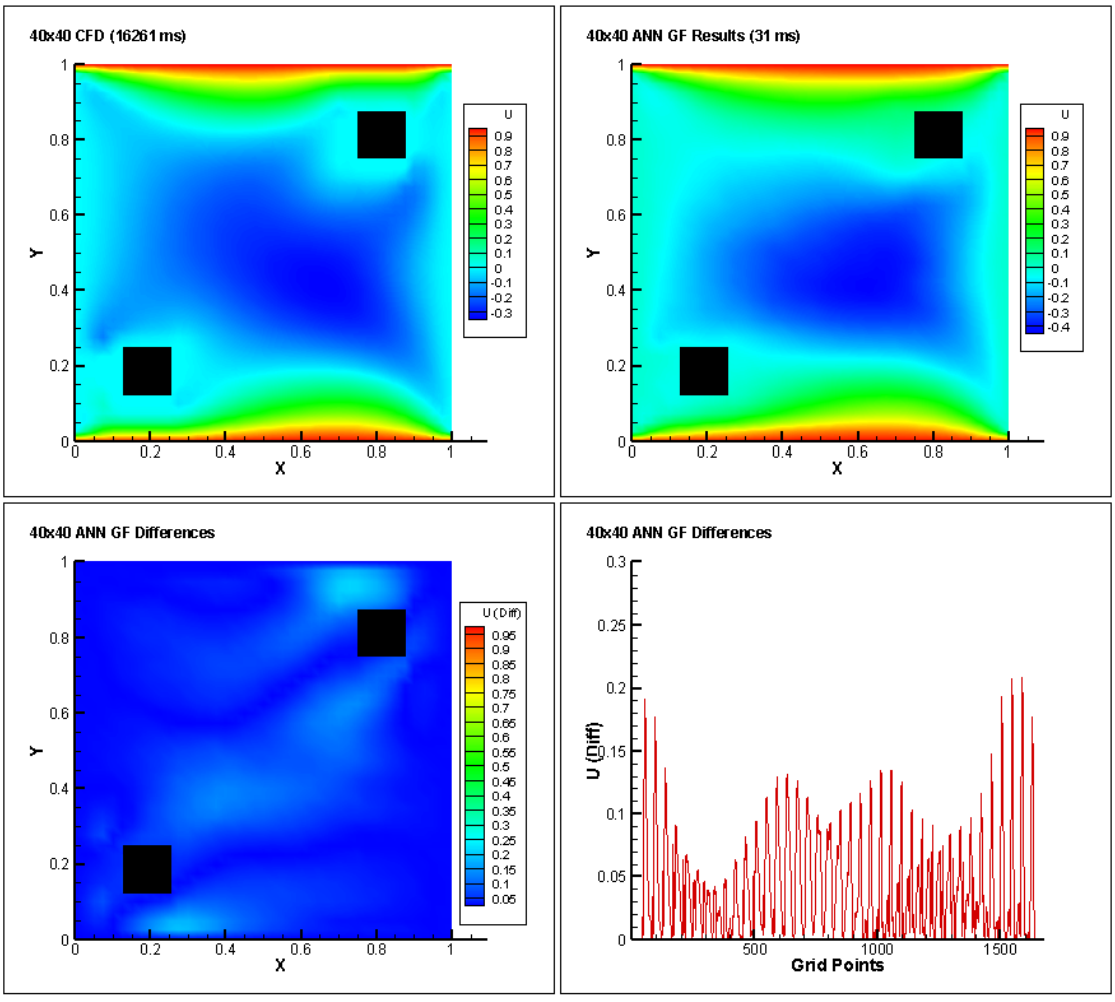


Figure 7.6: Test setup 1 - 40 × 40 grid system ANN GF comparison (U velocity)

The following output of U velocity as shown in figure 7.7 is from the neural network incorporating LS learning method in 40 × 40 grid system. The first flow field is from the CFD code, the second flow field is from the neural network, the third flow field shows the absolute differences between the two flow fields and the fourth graph shows a quantitative representation of the absolute differences. The CFD solution took 16261 milliseconds where the ANN prediction took only 31 milliseconds.

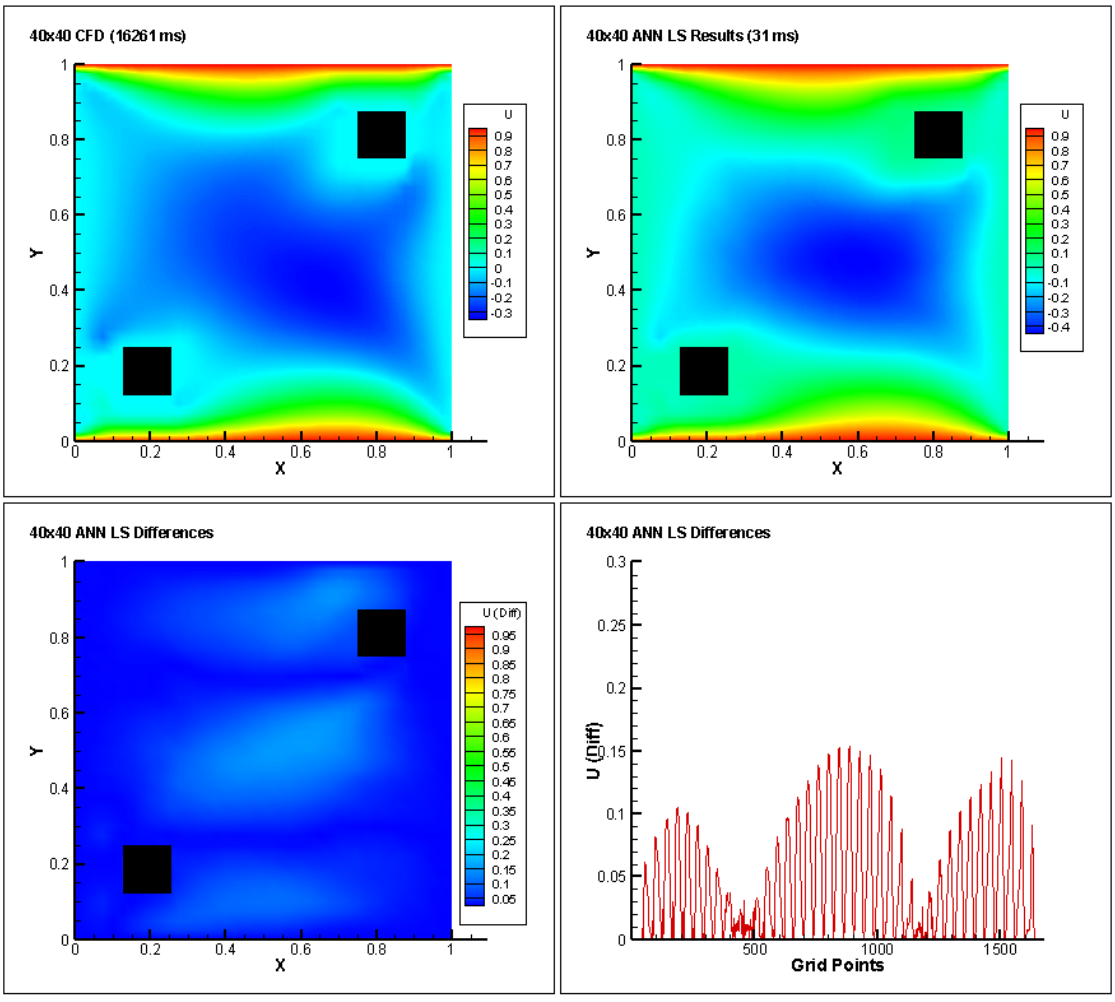


Figure 7.7: Test setup 1 - 40 × 40 grid system ANN LS comparison (U velocity)

7.2.3. Y-directional (V) velocity prediction

The following output of V velocity as shown in figure 7.8 is from the neural network incorporating GF learning method in 24 × 24 grid system. The first flow field is from the CFD code, the second flow field is from the neural network, the third flow field shows the absolute differences between the two flow fields and the fourth graph shows a quantitative representation of the absolute differences. Some relative errors can be seen along the edges of the square objects. The CFD solution took 5380 milliseconds where the ANN prediction took only 14 milliseconds.

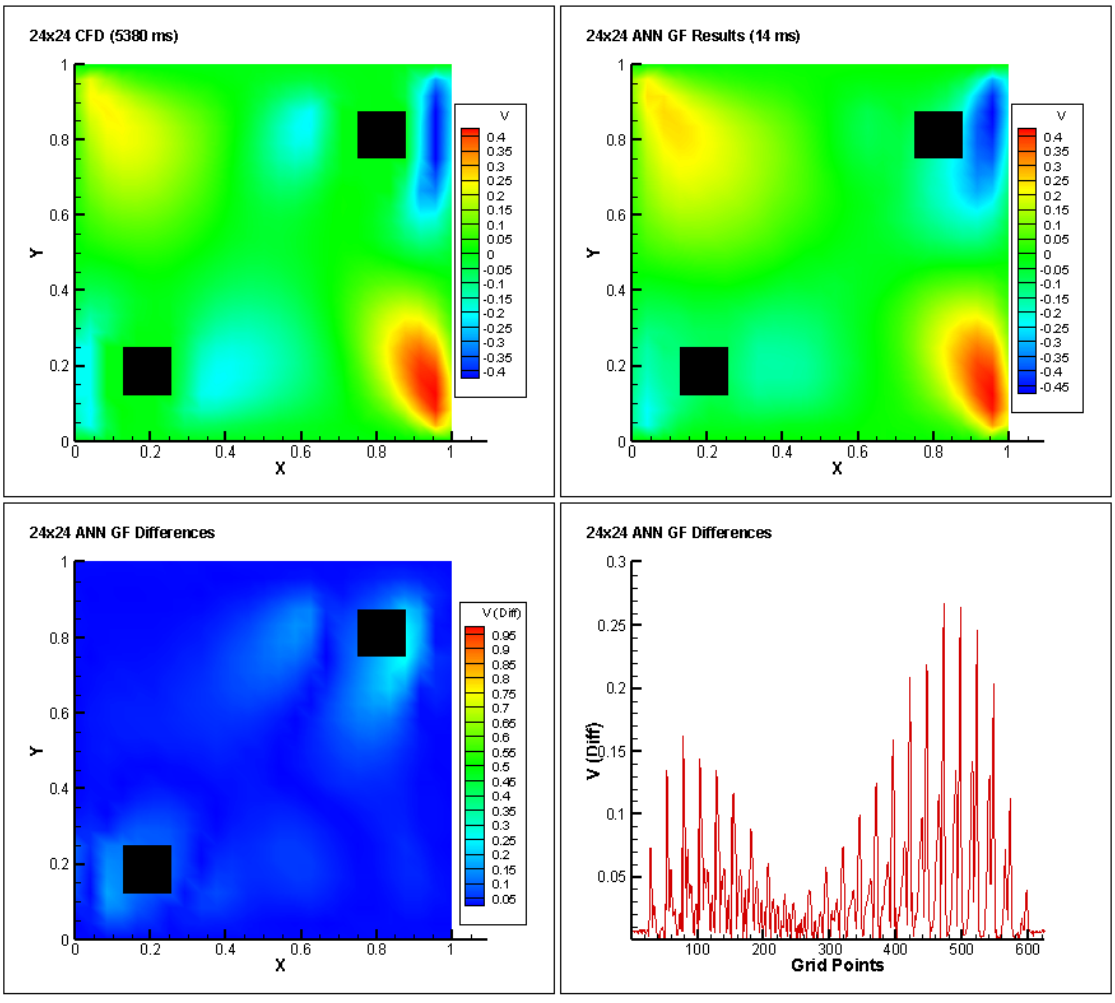


Figure 7.8: Test setup 1 - 24 × 24 grid system ANN GF comparison (V velocity)

The following output of V velocity as shown in figure 7.9 is from the neural network incorporating LS learning method in 24 × 24 grid system. The first flow field is from the CFD code, the second flow field is from the neural network, the third flow field shows the absolute differences between the two flow fields and the fourth graph shows a quantitative representation of the absolute differences. Relative errors in this case are significantly less than that of the GF learning method. The CFD solution took 5380 milliseconds where the ANN prediction took only 16 milliseconds.

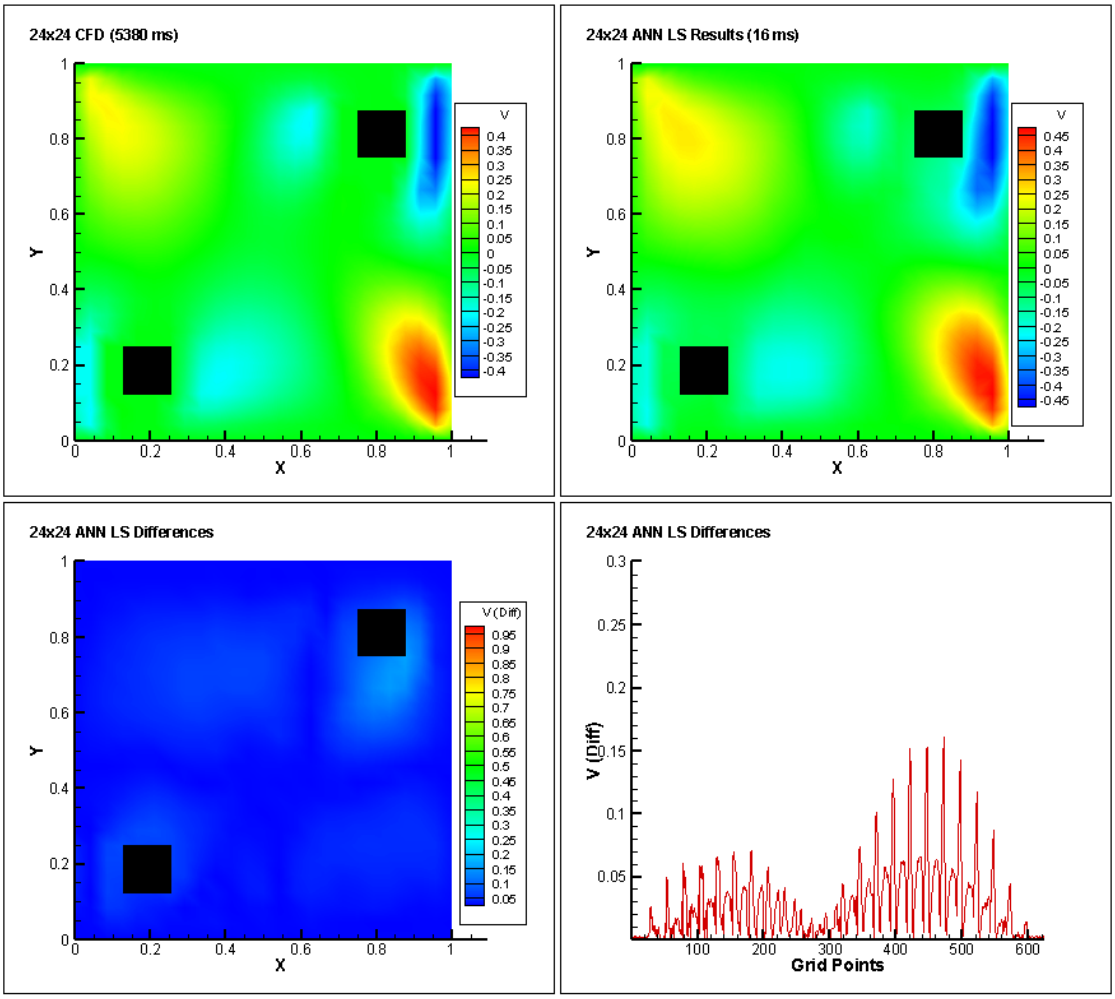


Figure 7.9: Test setup 1 - 24 × 24 grid system ANN LS comparison (V velocity)

The following output of V velocity as shown in figure 7.10 is from the neural network incorporating GF learning method in 32 × 32 grid system. The first flow field is from the CFD code, the second flow field is from the neural network, the third flow field shows the absolute differences between the two flow fields and the fourth graph shows a quantitative representation of the absolute differences. Some relative errors can be seen along the edges of the square objects. The CFD solution took 10774 milliseconds where the ANN prediction took only 15 milliseconds.

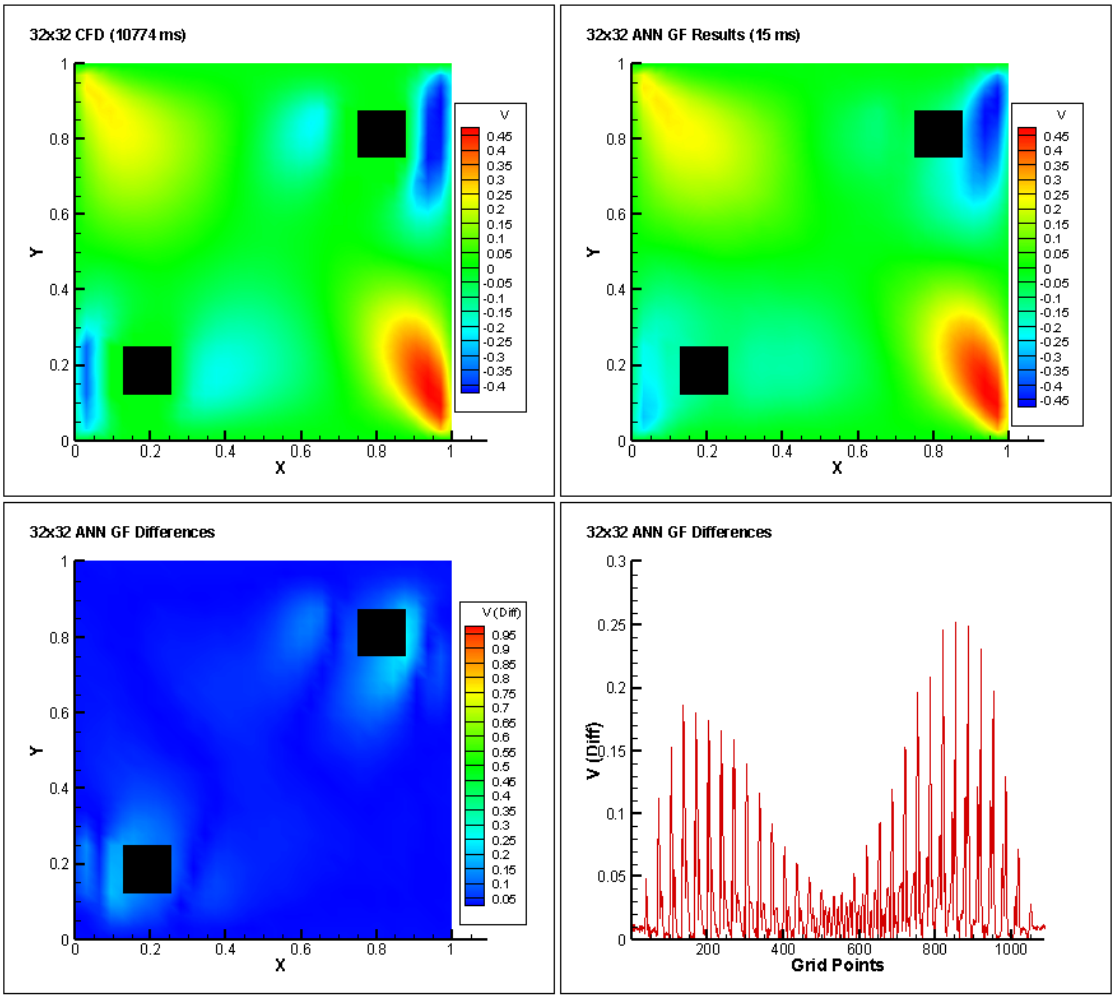


Figure 7.10: Test setup 1 - 32 × 32 grid system ANN GF comparison (V velocity)

The following output of V velocity as shown in figure 7.11 is from the neural network incorporating LS learning method in 32 × 32 grid system. The first flow field is from the CFD code, the second flow field is from the neural network, the third flow field shows the absolute differences between the two flow fields and the fourth graph shows a quantitative representation of the absolute differences. Relative errors in this case are significantly less than that of the GF learning method. The CFD solution took 10774 milliseconds where the ANN prediction took only 31 milliseconds.

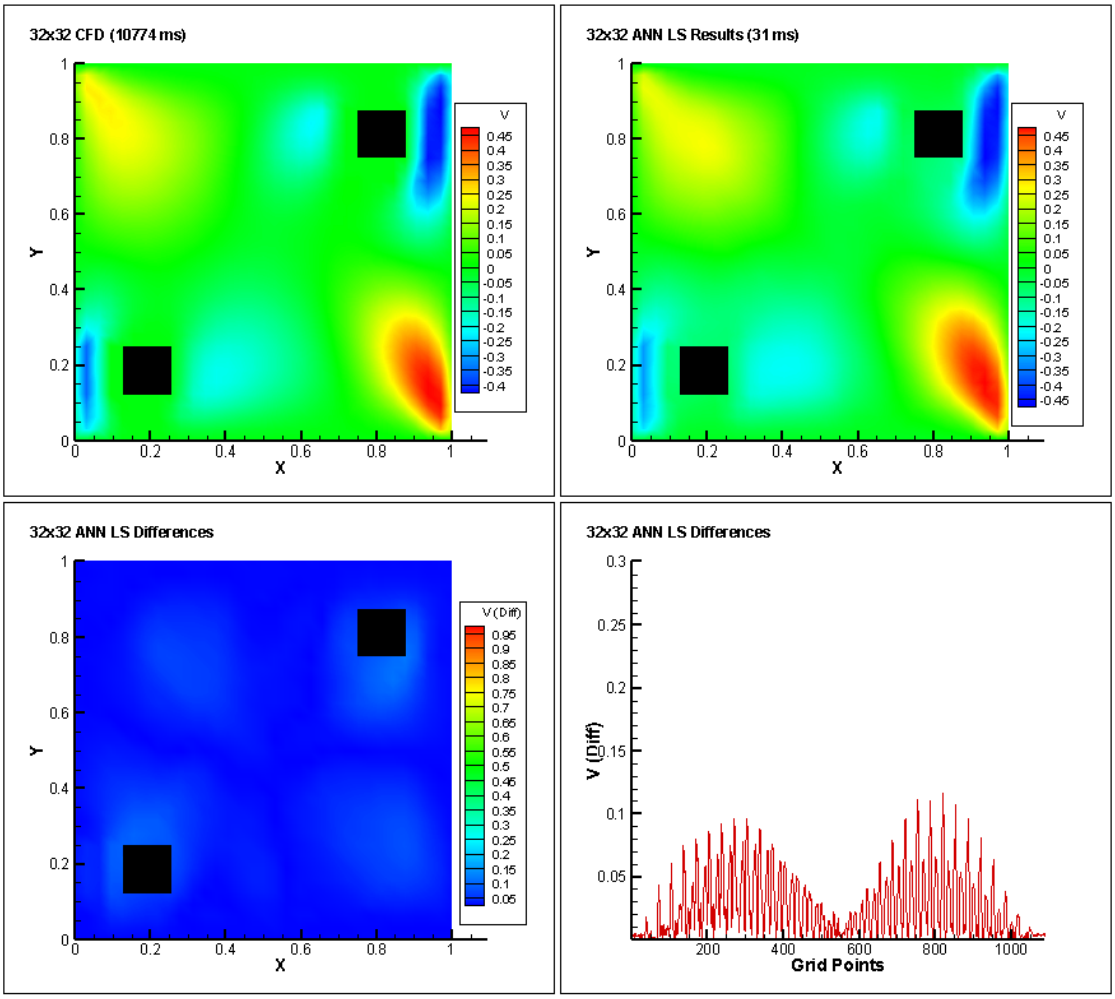


Figure 7.11: Test setup 1 - 32 × 32 grid system ANN LS comparison (V velocity)

The following output of V velocity as shown in figure 7.12 is from the neural network incorporating GF learning method in 40 × 40 grid system. The first flow field is from the CFD code, the second flow field is from the neural network, the third flow field shows the absolute differences between the two flow fields and the fourth graph shows a quantitative representation of the absolute differences. The CFD solution took 16261 milliseconds where the ANN prediction took only 31 milliseconds.

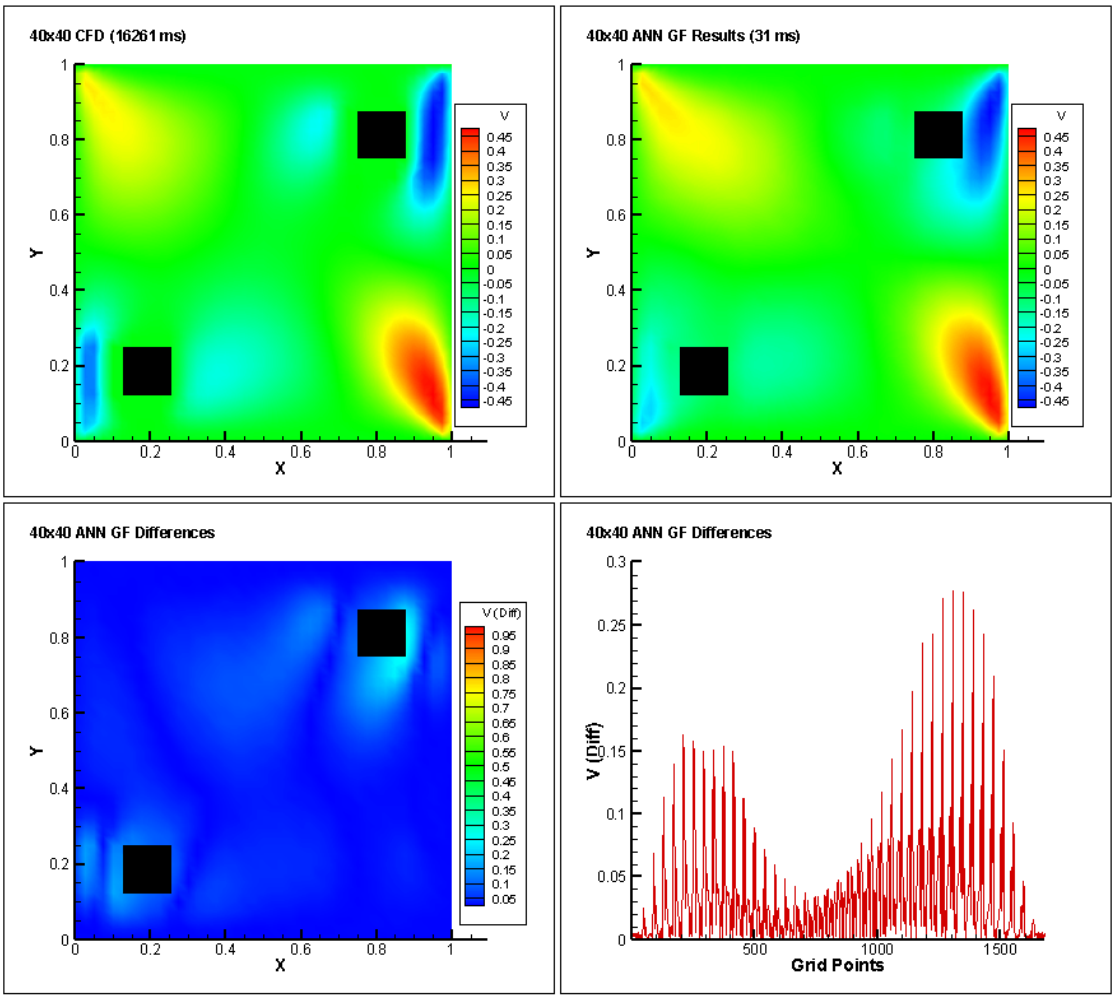


Figure 7.12: Test setup 1 - 40 × 40 grid system ANN GF comparison (V velocity)

The following output of V velocity as shown in figure 7.13 is from the neural network incorporating LS learning method in 40 × 40 grid system. The first flow field is from the CFD code, the second flow field is from the neural network, the third flow field shows the absolute differences between the two flow fields and the fourth graph shows a quantitative representation of the absolute differences. The CFD solution took 16261 milliseconds where the ANN prediction took only 31 milliseconds.

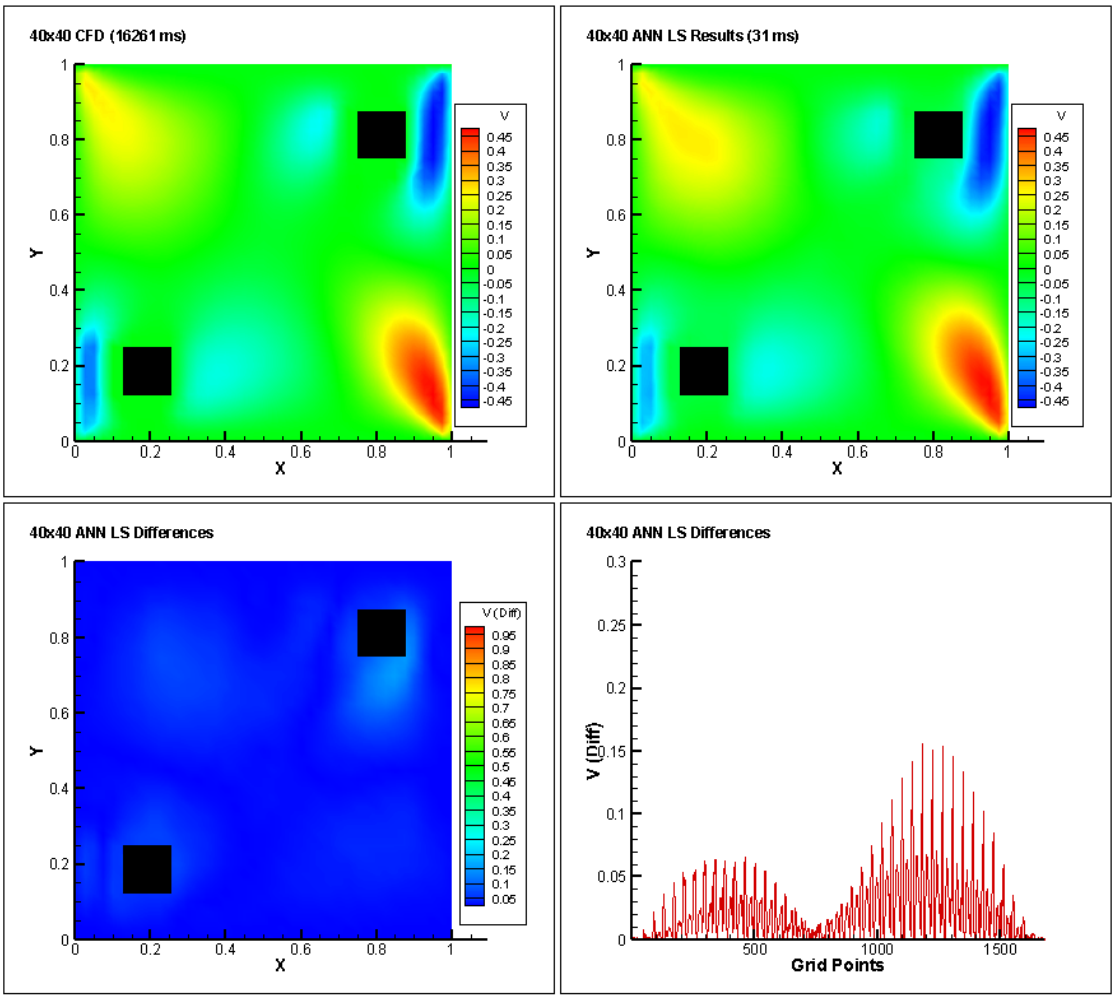


Figure 7.13: Test setup 1 - 40 × 40 grid system ANN LS comparison (V velocity)

7.3. Second Test Setup

7.3.1. Environment

Three square objects were placed together as shown in figure 7.14, inside the double lid-driven cavity in a way that they make up a relatively complex shape. Again, the boundary conditions in the cavity was the same as the training phase. Figure 7.14 shows the arrangement of the square objects inside the cavity. The following pages compares the results of the U and V velocities generated using the three different grid systems incorporating the two different learning rates that have been used.

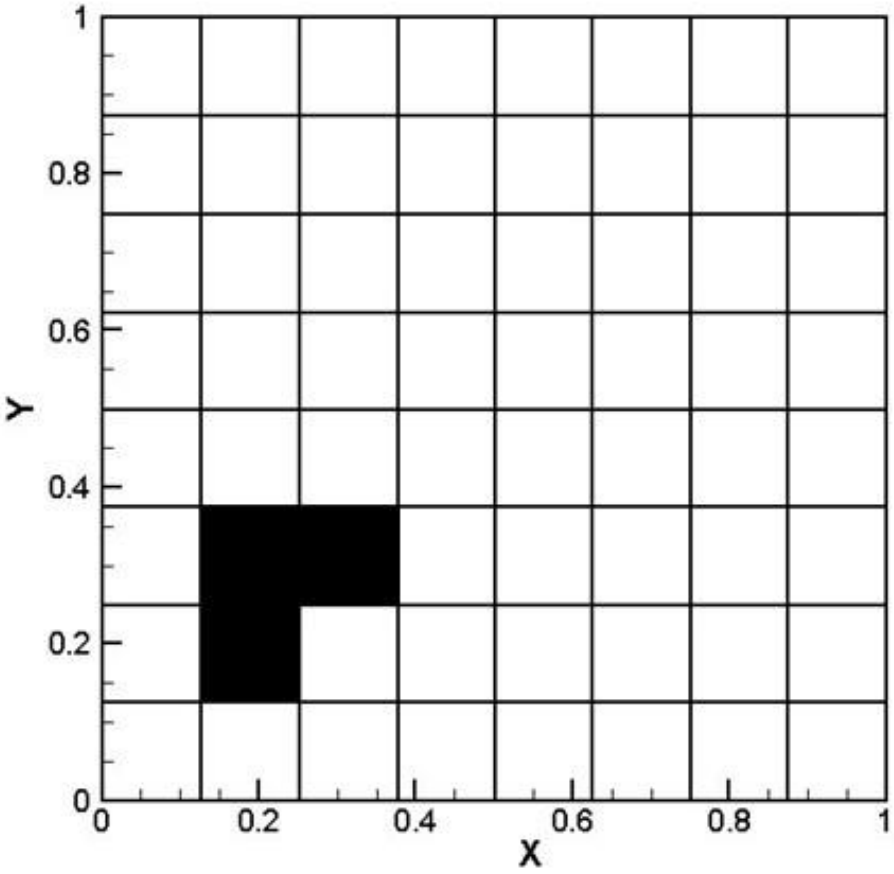


Figure 7.14: Double lid-driven cavity with three obstacles making a relatively complex shape

7.3.2. X-directional (U) velocity prediction

The placement of the three square objects resulted in the output of U velocity as shown in figure 7.15 from the neural network incorporating GF learning method in 24 × 24 grid system. The first flow field is from the CFD code, the second flow field is from the neural network, the third flow field shows the absolute differences between the two flow fields and the fourth graph shows a quantitative representation of the absolute differences. Some relative errors can be seen along the edges of the square objects. The CFD solution took 5023 milliseconds where the ANN prediction took only 20 milliseconds.

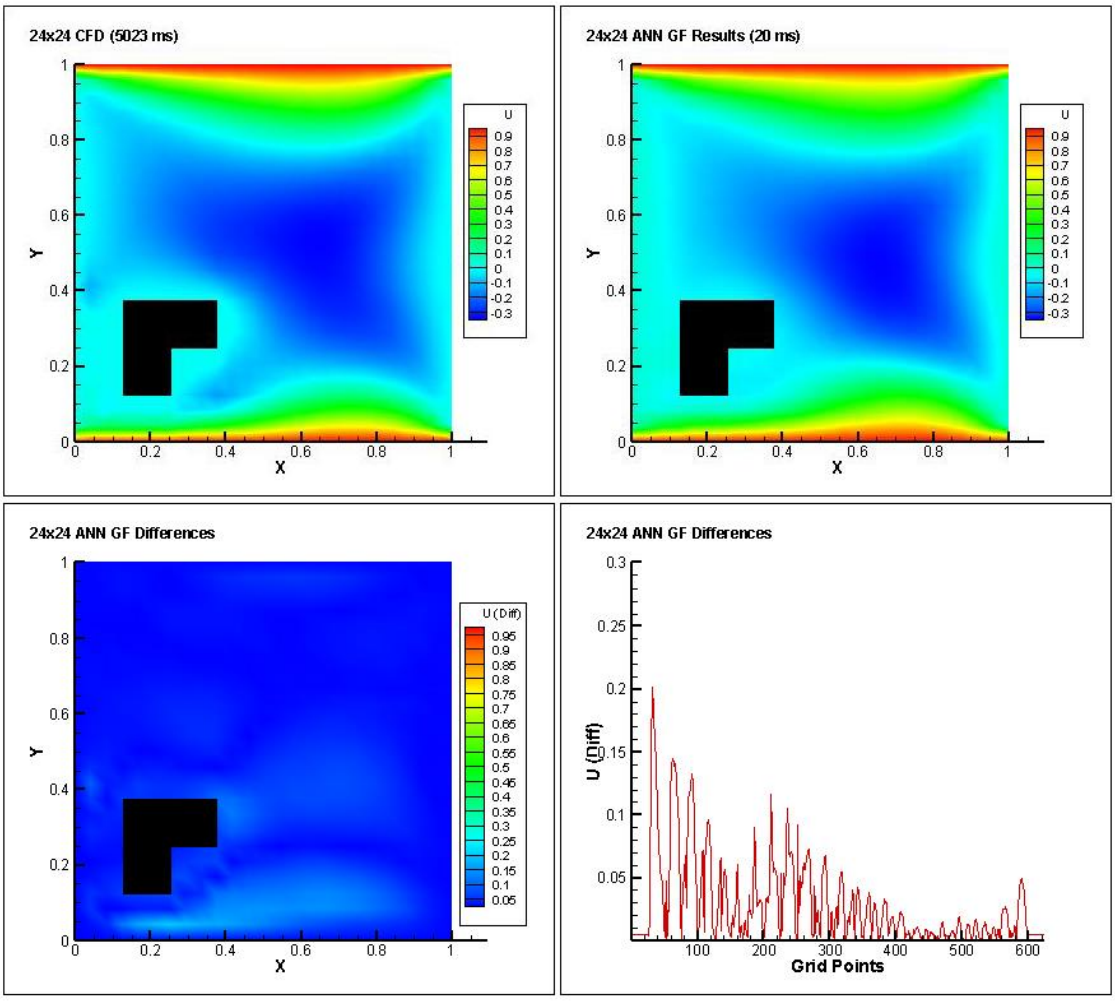


Figure 7.15: Test setup 2 - 24 × 24 grid system ANN GF comparison (U velocity)

The output of U velocity as shown in figure 7.16 is from the neural network incorporating LS learning method in 24 × 24 grid system. The first flow field is from the CFD code, the second flow field is from the neural network, the third flow field shows the absolute differences between the two flow fields and the fourth graph shows a quantitative representation of the absolute differences. Relative errors in this case are less than that of the GF learning method. Some errors are seen in the central part of the cavity. The CFD solution took 5023 milliseconds where the ANN prediction took only 15 milliseconds.

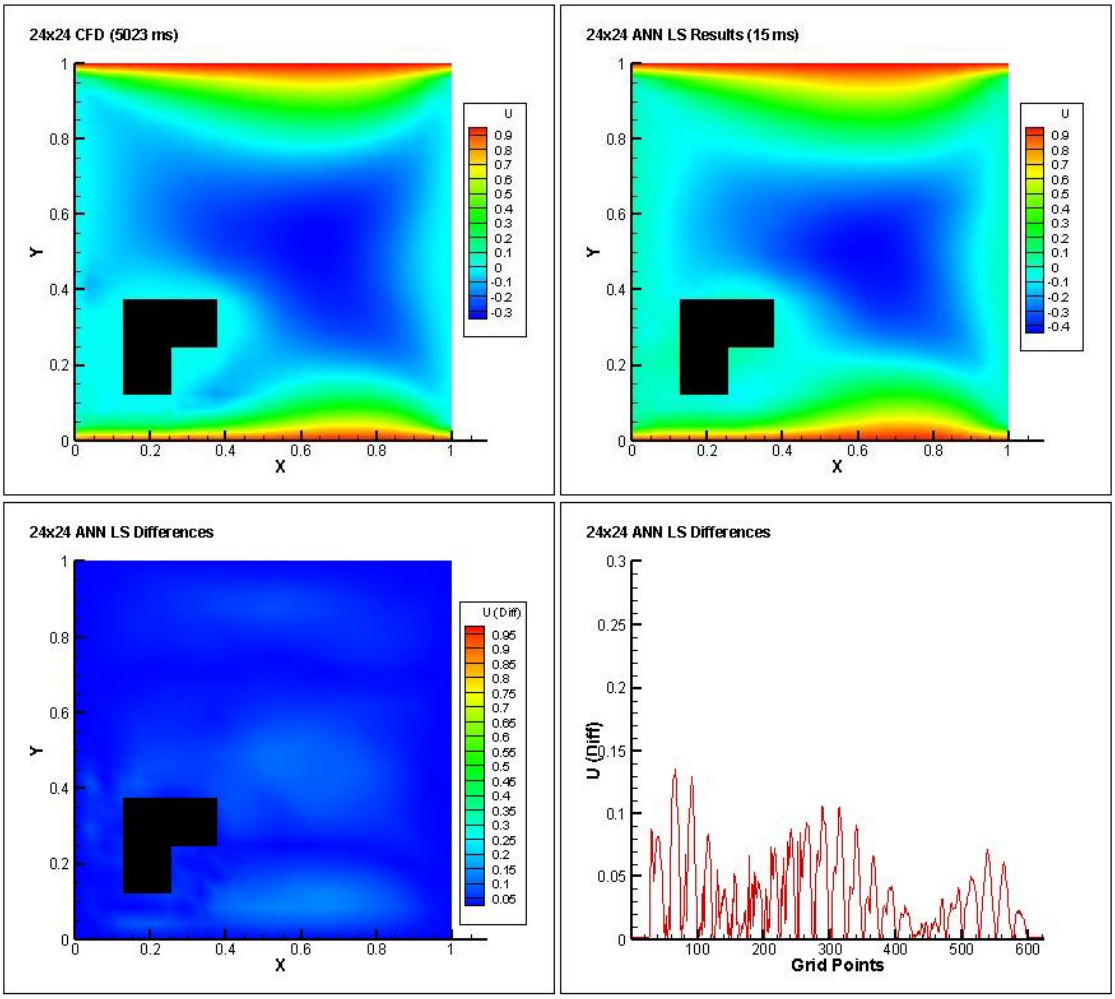


Figure 7.16: Test setup 2 - 24 × 24 grid system ANN LS comparison (U velocity)

The output of U velocity as shown in figure 7.17 is from the neural network incorporating GF learning method in 32 × 32 grid system. The first flow field is from the CFD code, the second flow field is from the neural network, the third flow field shows the absolute differences between the two flow fields and the fourth graph shows a quantitative representation of the absolute differences. Some relative errors can be seen along the edges of the square objects. The CFD solution took 8776 milliseconds where the ANN prediction took only 21 milliseconds.

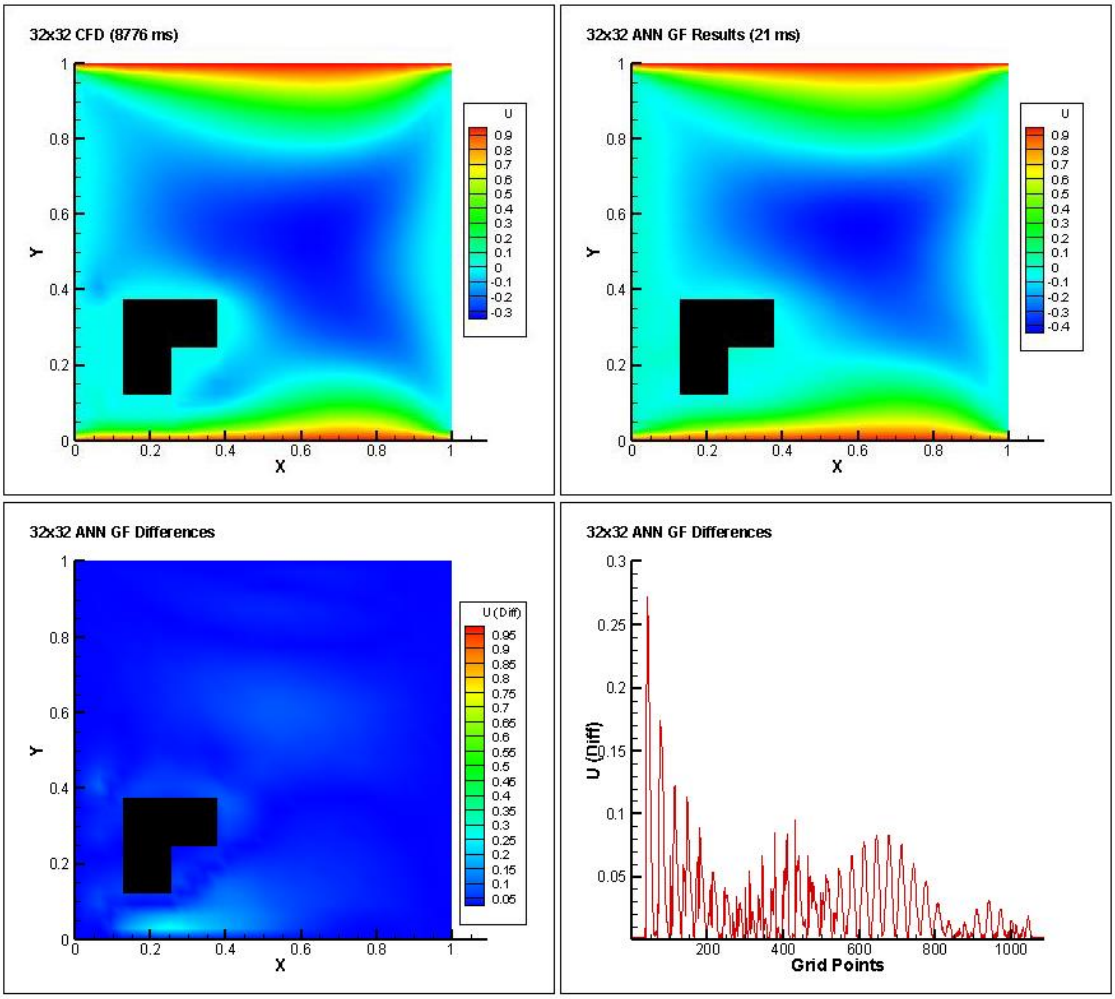


Figure 7.17: Test setup 2 - 32 × 32 grid system ANN GF comparison (U velocity)

The output of U velocity as shown in figure 7.18 is from the neural network incorporating LS learning method in 32 × 32 grid system. The first flow field is from the CFD code, the second flow field is from the neural network, the third flow field shows the absolute differences between the two flow fields and the fourth graph shows a quantitative representation of the absolute differences. Relative errors in this case are less than that of the GF learning method. Some errors are seen in the central part of the cavity. The CFD solution took 8776 milliseconds where the ANN prediction took only 23 milliseconds.

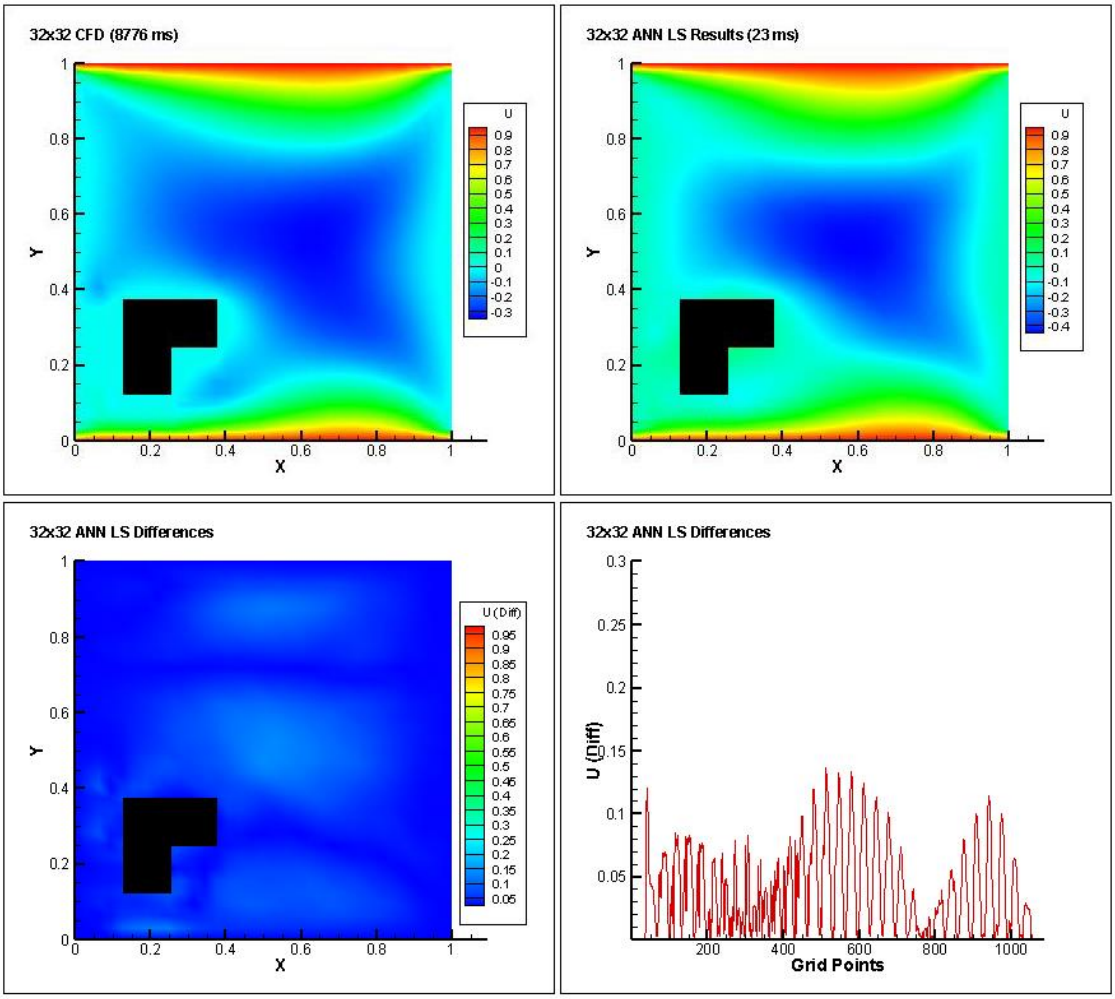


Figure 7.18: Test setup 2 - 32 × 32 grid system ANN LS comparison (U velocity)

The output of U velocity as shown in figure 7.19 is from the neural network incorporating GF learning method in 40 × 40 grid system. The first flow field is from the CFD code, the second flow field is from the neural network, the third flow field shows the absolute differences between the two flow fields and the fourth graph shows a quantitative representation of the absolute differences. Some relative errors can be seen along the edges of the square objects. The CFD solution took 26605 milliseconds where the ANN prediction took only 57 milliseconds.

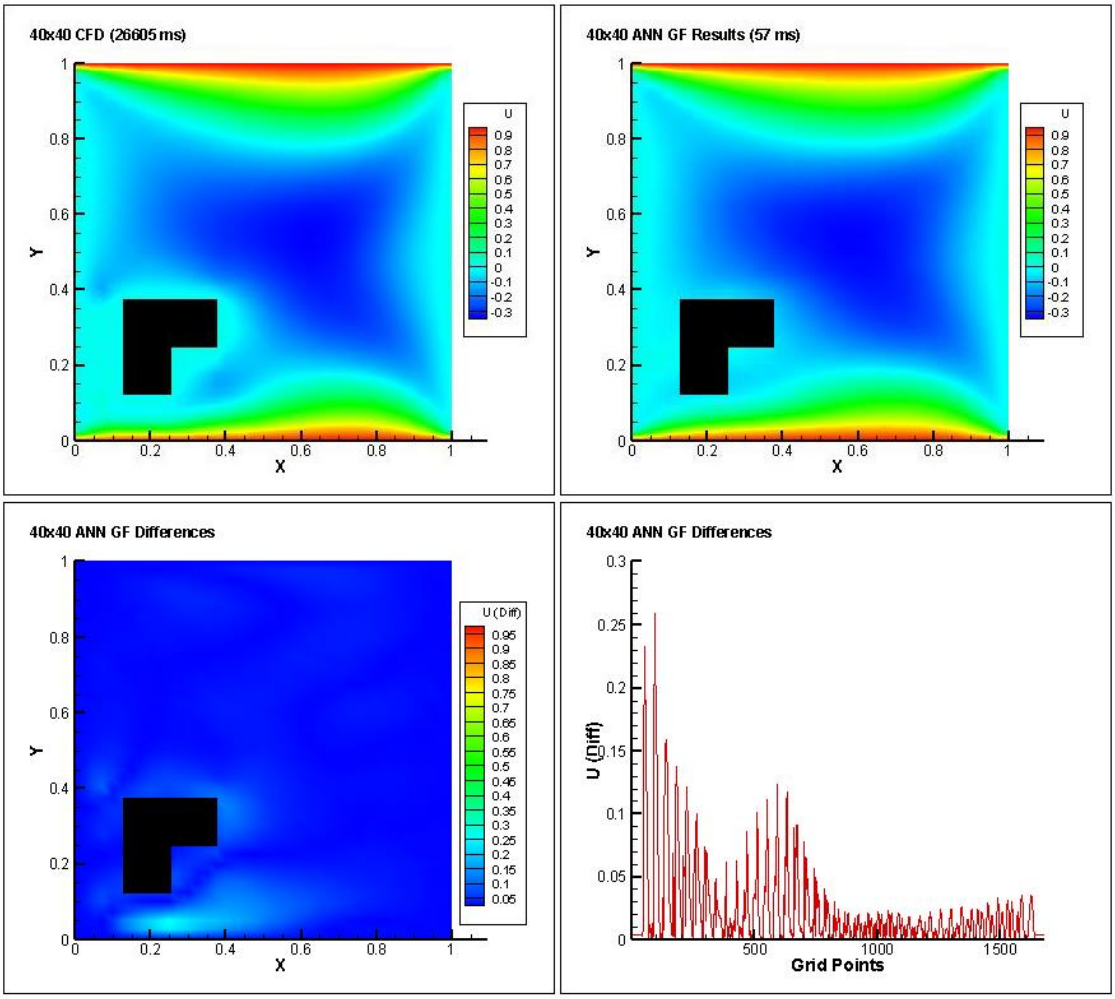


Figure 7.19: Test setup 2 - 40 × 40 grid system ANN GF comparison (U velocity)

The output of U velocity as shown in figure 7.20 is from the neural network incorporating LS learning method in 40 × 40 grid system. The first flow field is from the CFD code, the second flow field is from the neural network, the third flow field shows the absolute differences between the two flow fields and the fourth graph shows a quantitative representation of the absolute differences. Relative errors in this case are less than that of the GF learning method. The CFD solution took 26605 milliseconds where the ANN prediction took only 53 milliseconds.

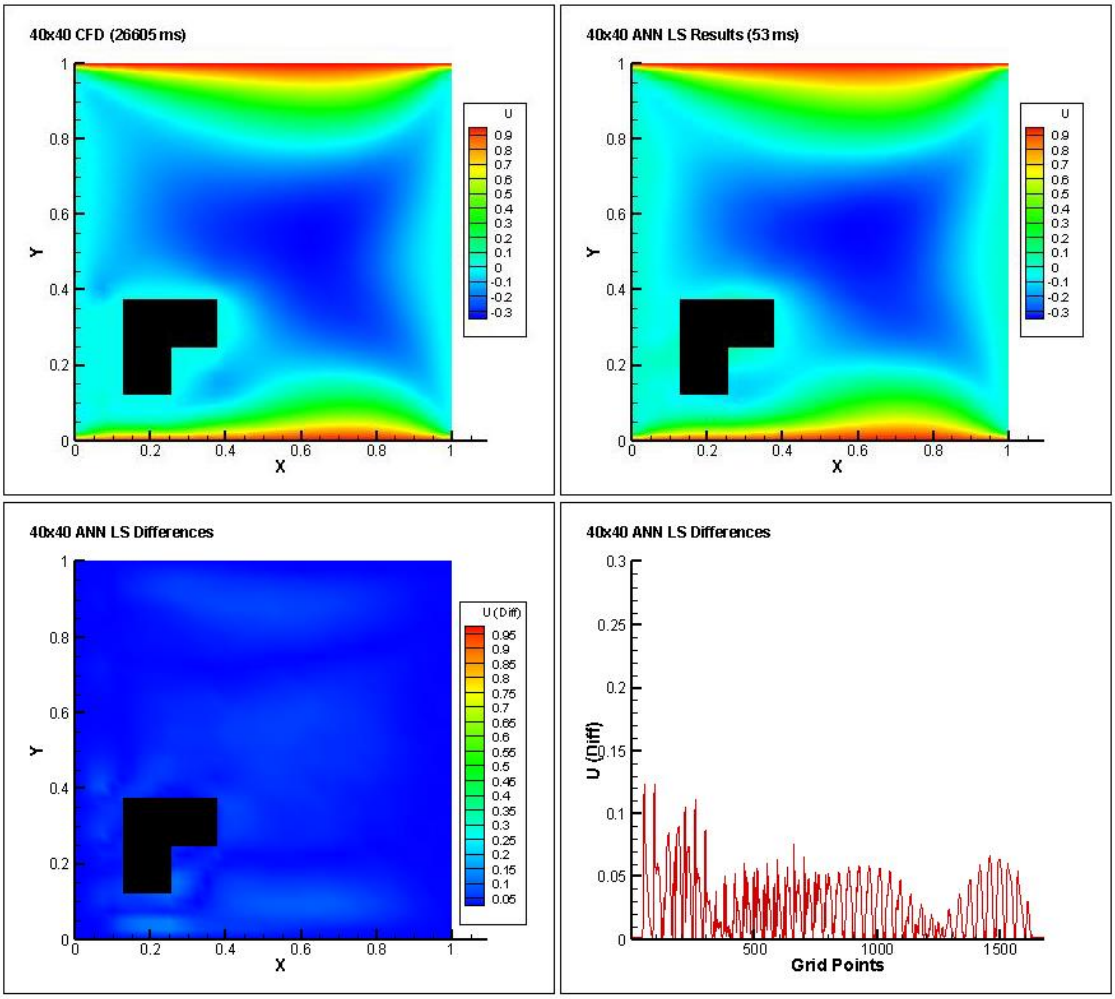


Figure 7.20: Test setup 2 - 40 × 40 grid system ANN LS comparison (U velocity)

7.3.3. Y-directional (V) velocity prediction

The output of V velocity as shown in figure 7.21 is from the neural network incorporating GF learning method in 24 × 24 grid system. The first flow field is from the CFD code, the second flow field is from the neural network, the third flow field shows the absolute differences between the two flow fields and the fourth graph shows a quantitative representation of the absolute differences. Some relative errors can be seen along the edges of the square objects. The CFD solution took 5023 milliseconds where the ANN prediction took only 20 milliseconds.

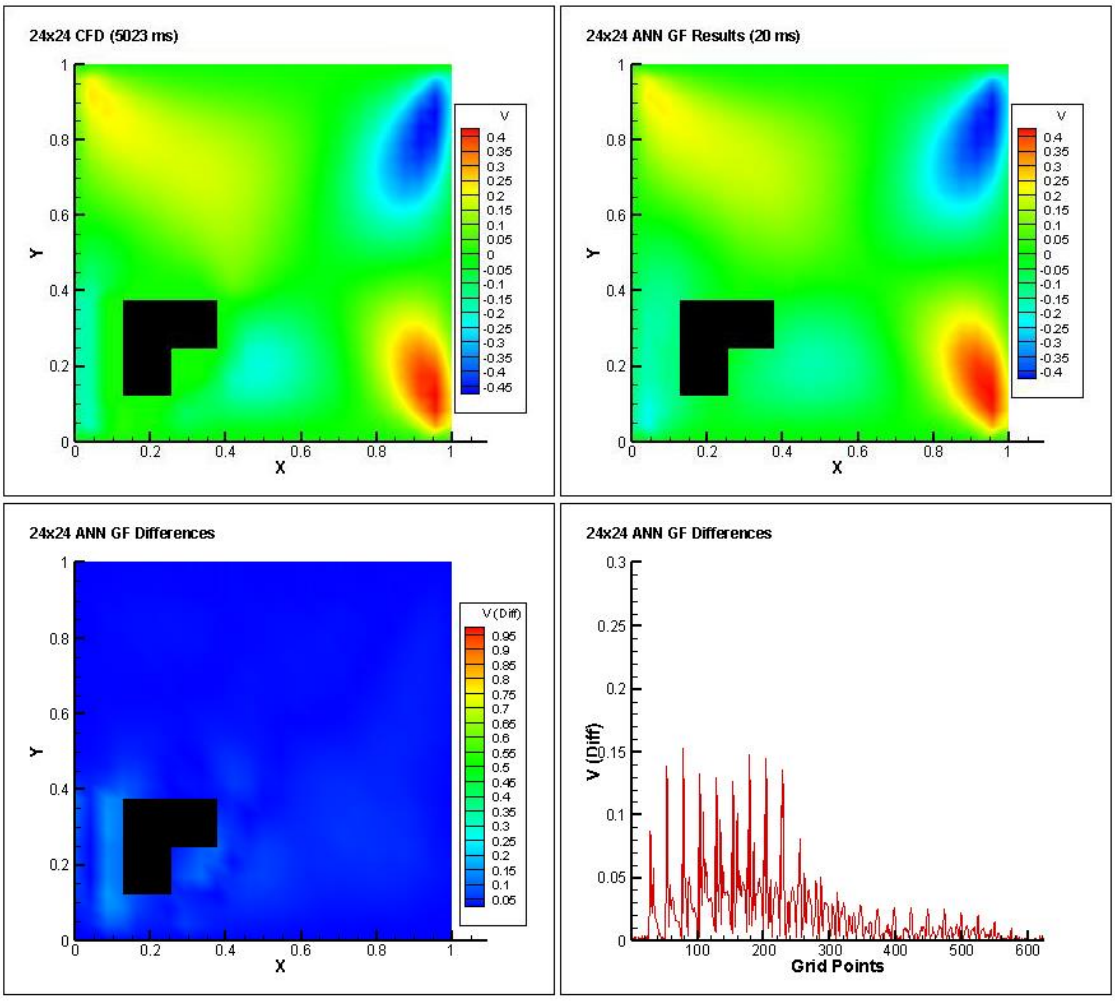


Figure 7.21: Test setup 2 - 24 × 24 grid system ANN GF comparison (V velocity)

The output of V velocity as shown in figure 7.22 is from the neural network incorporating LS learning method in 24 × 24 grid system. The first flow field is from the CFD code, the second flow field is from the neural network, the third flow field shows the absolute differences between the two flow fields and the fourth graph shows a quantitative representation of the absolute differences. Relative errors in this case are approximately the same as of the GF learning method. The CFD solution took 5023 milliseconds where the ANN prediction took only 15 milliseconds.

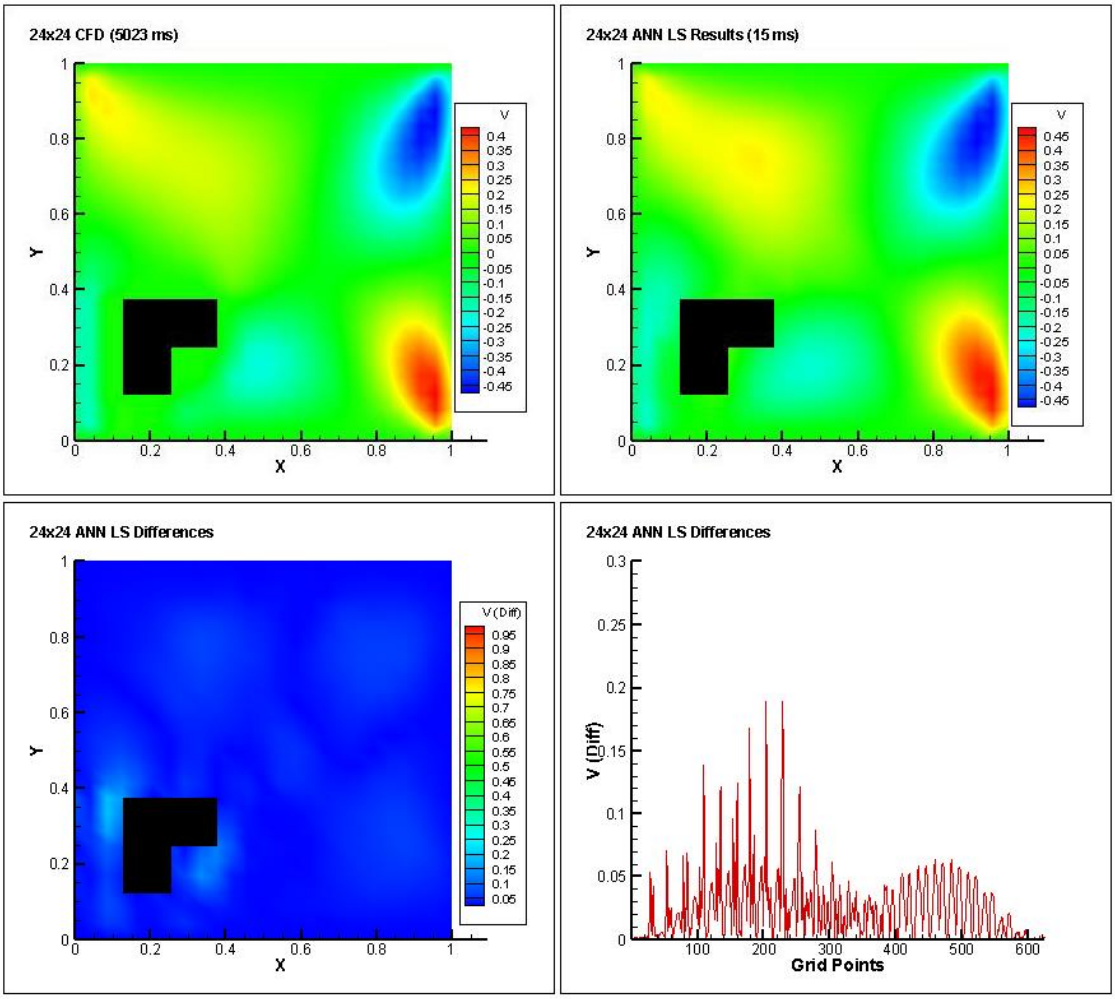


Figure 7.22: Test setup 2 - 24 × 24 grid system ANN LS comparison (V velocity)

The output of V velocity as shown in figure 7.23 is from the neural network incorporating GF learning method in 32 × 32 grid system. The first flow field is from the CFD code, the second flow field is from the neural network, the third flow field shows the absolute differences between the two flow fields and the fourth graph shows a quantitative representation of the absolute differences. Some relative errors can be seen along the edges of the square objects. The CFD solution took 8776 milliseconds where the ANN prediction took only 21 milliseconds.

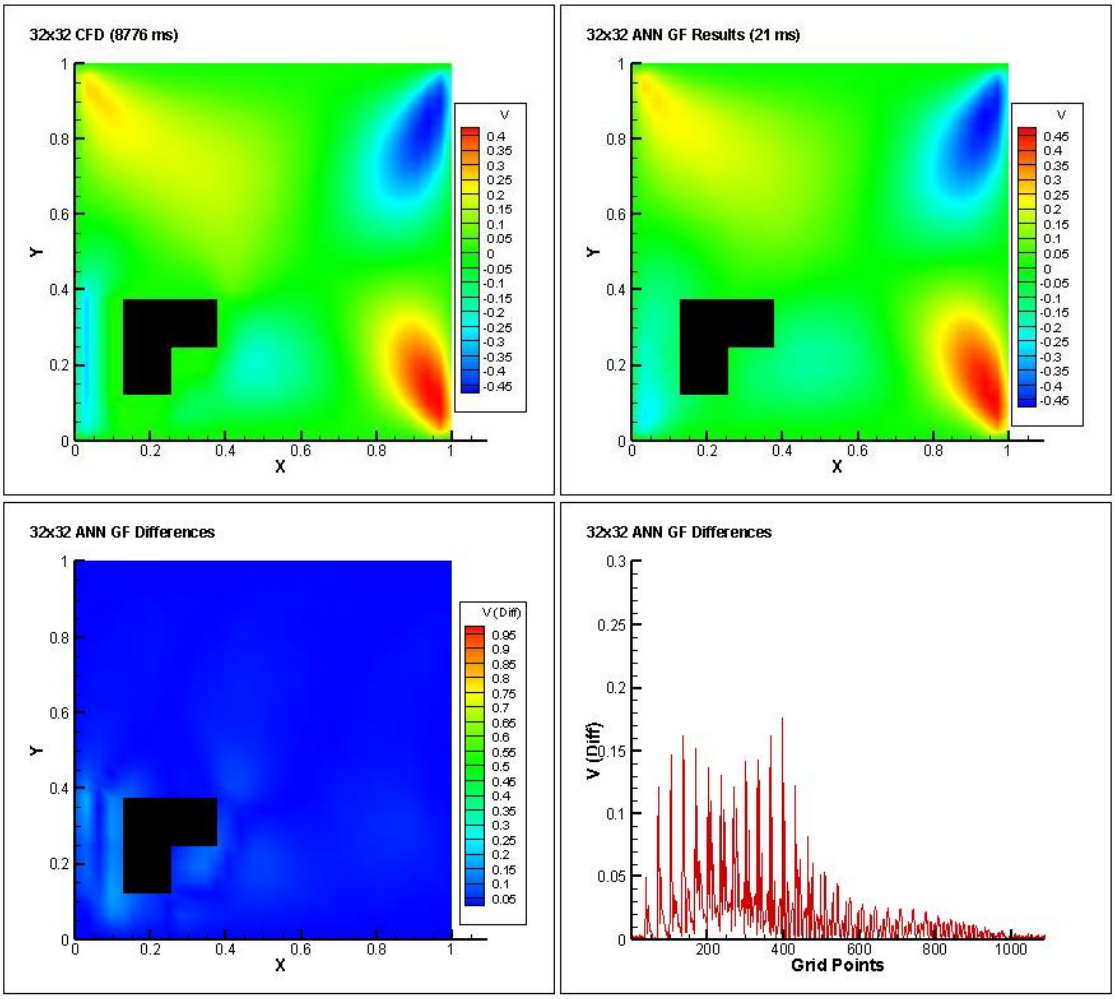


Figure 7.23: Test setup 2 - 32 × 32 grid system ANN GF comparison (V velocity)

The output of V velocity as shown in figure 7.24 is from the neural network incorporating LS learning method in 32 × 32 grid system. The first flow field is from the CFD code, the second flow field is from the neural network, the third flow field shows the absolute differences between the two flow fields and the fourth graph shows a quantitative representation of the absolute differences. Relative errors in this case are a little less than that of the GF learning method. The CFD solution took 8776 milliseconds where the ANN prediction took only 23 milliseconds.

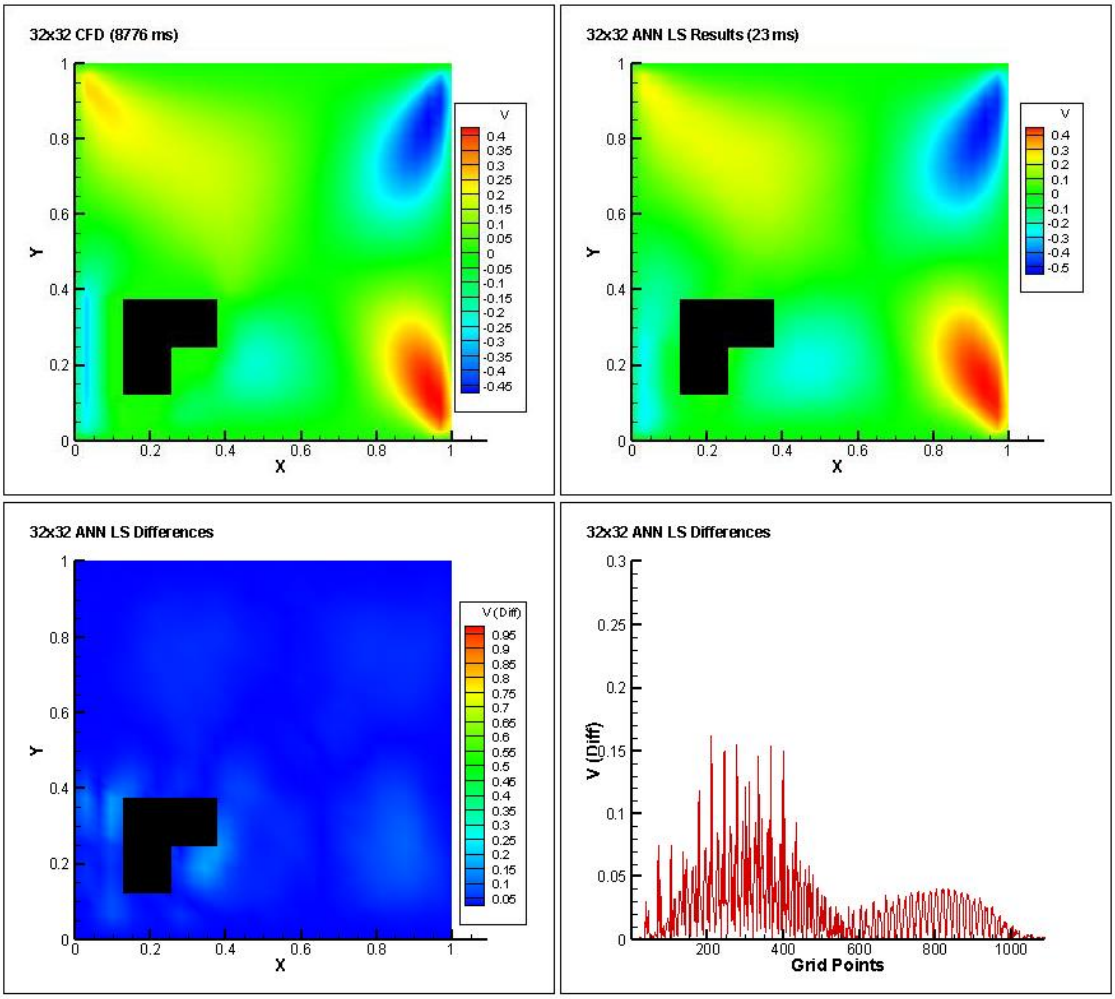


Figure 7.24: Test setup 2 - 32 × 32 grid system ANN LS comparison (V velocity)

The output of V velocity as shown in figure 7.25 is from the neural network incorporating GF learning method in 40 × 40 grid system. The first flow field is from the CFD code, the second flow field is from the neural network, the third flow field shows the absolute differences between the two flow fields and the fourth graph shows a quantitative representation of the absolute differences. Some relative errors can be seen along the edges of the square objects. The CFD solution took 26605 milliseconds where the ANN prediction took only 57 milliseconds.

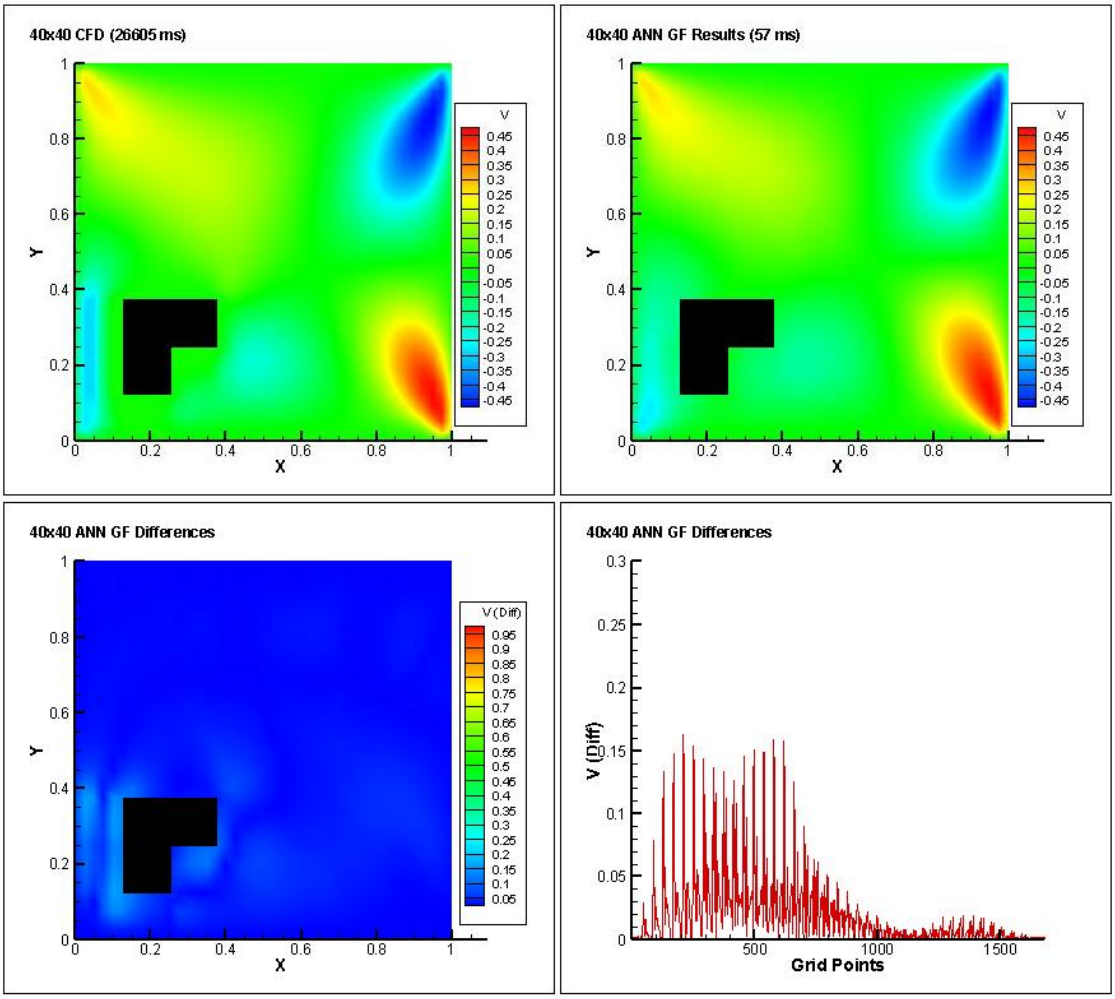


Figure 7.25: Test setup 2 - 40 × 40 grid system ANN GF comparison (V velocity)

The output of V velocity as shown in figure 7.26 is from the neural network incorporating LS learning method in 40 × 40 grid system. The first flow field is from the CFD code, the second flow field is from the neural network, the third flow field shows the absolute differences between the two flow fields and the fourth graph shows a quantitative representation of the absolute differences. Relative errors in this case are a little less than that of the GF learning method. The CFD solution took 26605 milliseconds where the ANN prediction took only 53 milliseconds.

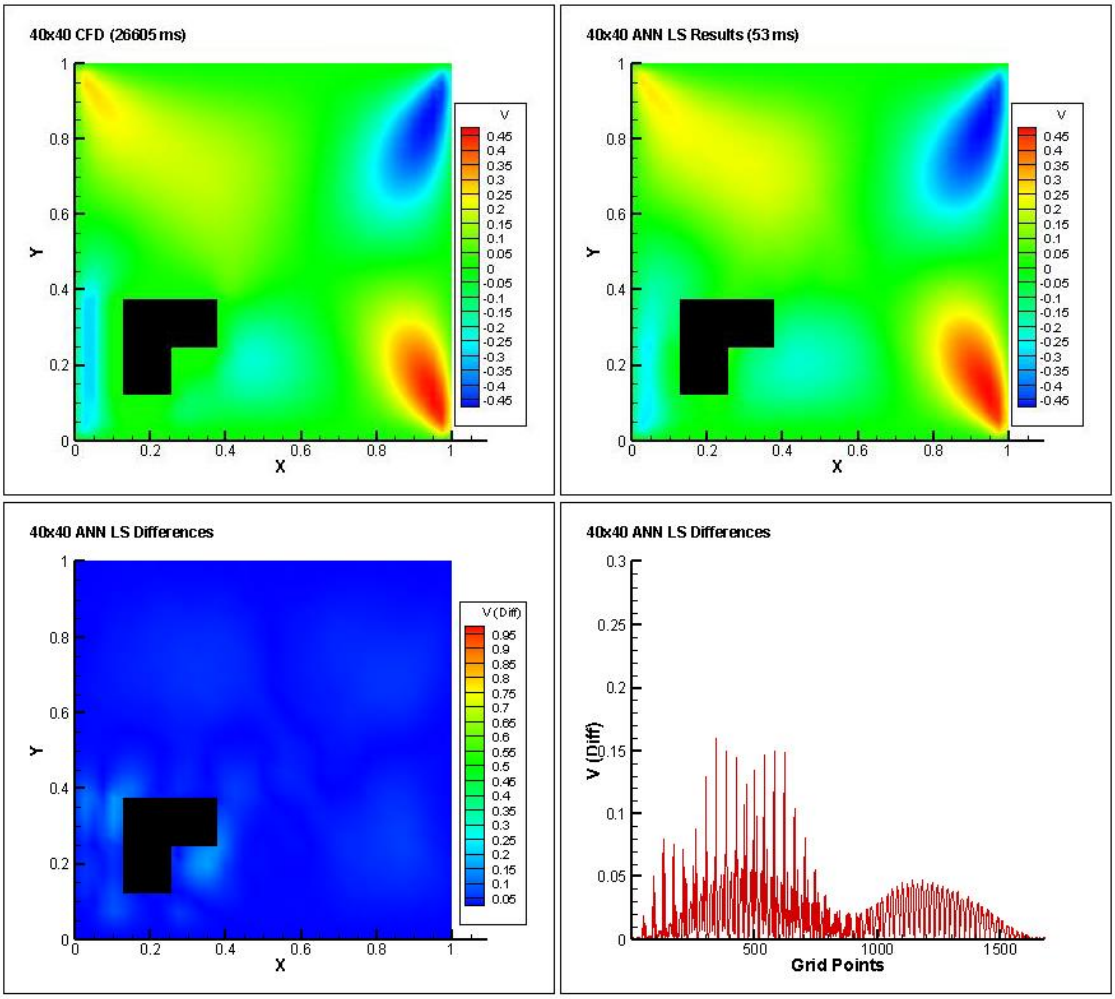


Figure 7.26: Test setup 2 - 40 × 40 grid system ANN LS comparison (V velocity)

7.4. Time Comparison

Table 7.2 compares the time required for the CFD solver vs the two different ANN solvers using 5 different setups.

Table 7.2: Time comparison between CFD solver and ANN solver

Setup	**Time (milliseconds)**
24 × 24 grid system	
Setup 1 CFD	6227
Setup 1 ANN GF	31
Setup 1 ANN LS	15
Setup 2 CFD	4156
Setup 2 ANN GF	15
Setup 2 ANN LS	15
Setup 3 CFD	4670
Setup 3 ANN GF	14
Setup 3 ANN LS	18
Setup 4 CFD	4578
Setup 4 ANN GF	18
Setup 4 ANN LS	23
Setup 5 CFD	5116
Setup 5 ANN GF	14
Setup 5 ANN LS	15
32 × 32 grid system	
Setup 1 CFD	10250
Setup 1 ANN GF	31
Setup 1 ANN LS	31
Setup 2 CFD	6547

Setup	**Time (milliseconds)**
Setup 2 ANN GF	15
Setup 2 ANN LS	31
Setup 3 CFD	7214
Setup 3 ANN GF	19
Setup 3 ANN LS	28
Setup 4 CFD	7874
Setup 4 ANN GF	20
Setup 4 ANN LS	31
Setup 5 CFD	8107
Setup 5 ANN GF	20
Setup 5 ANN LS	26
40 × 40 grid system	
Setup 1 CFD	16094
Setup 1 ANN GF	31
Setup 1 ANN LS	15
Setup 2 CFD	9570
Setup 2 ANN GF	27
Setup 2 ANN LS	28
Setup 3 CFD	10874
Setup 3 ANN GF	29
Setup 3 ANN LS	32
Setup 4 CFD	13605
Setup 4 ANN GF	28
Setup 4 ANN LS	31
Setup 5 CFD	11619
Setup 5 ANN GF	29
Setup 5 ANN LS	29

7.5. Two More Results

Two more test results are shown using 40 × 40 grid system and layer specific learning method. The left images are generated by the CFD C# code and right images are generated by the neural networks. The first one, as shown in figure 7.27, involves a geometry involving five obstacles placed together forming an "L" shaped object. It is clearly evident that the more complex the geometry becomes, the more erroneous the neural network's output becomes. However, by using deeper neural networks or by reduced order modelling of the system, it may be possible by future researches to refine the output of the network.

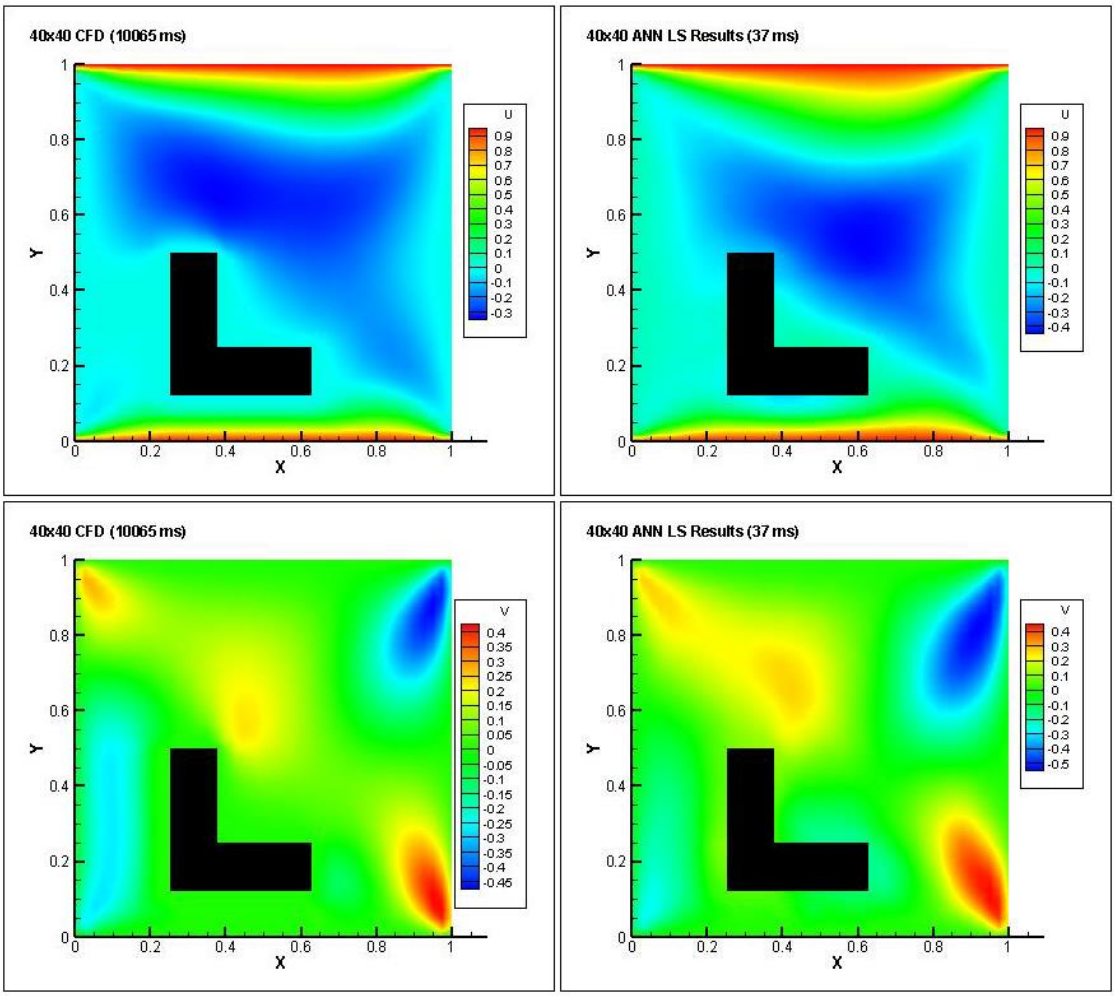


Figure 7.27: Test setup 3 - 40 × 40 grid system and LS learning method

The second test result, as shown in figure 7.28, involves six obstacles placed together in such a way that they make up a rectangle.

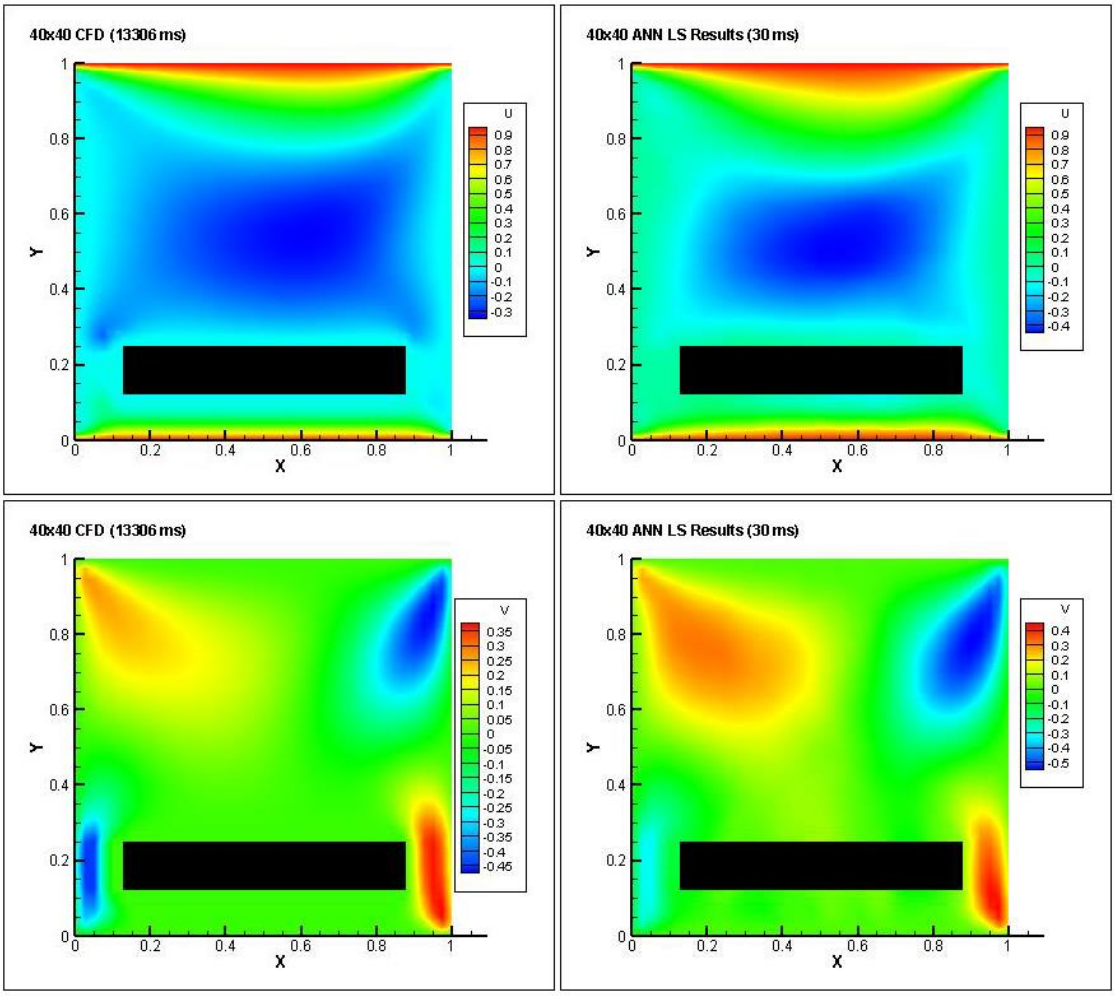


Figure 7.28: Test setup 4 - 40 × 40 grid system and LS learning method

It is observed that the neural networks are able to predict fluid velocities with a fair amount of accuracy in all of the cases in a fraction of the time taken by the CFD solver. Both GF and LS methods did a good job in predicting the fluid flow. However, LS method was better in predicting fluid flow near the obstacles as it had less errors near the obstacles when compared to the GF method. Furthermore, the velocities obtained from the neural networks can be used as initial values in CFD solvers to make them even more accurate.

CHAPTER EIGHT

CONCLUSION

8.1. Concluding Remarks

The results obtained from this research clearly shows that it is indeed possible to predict fluid flow patterns using neural networks. Researches often have to come up with quick solutions or a snapshot of what the actual flow field will look like in order to get an idea of what they are working with before generating a full-fledged CFD solution. In this case, trained neural networks may provide a faster alternative to traditional CFD approaches. Although this requires training the neural networks with vast amount of data, which takes a lot of time, training has to be done only once. If a community driven platform for practicing and implementing data driven fluid dynamics is established, where people will share their trained neural networks with other people, it is quite possible to develop a general artificial intelligence which can tackle even the most complex fluid mechanics problems. Although different flow fields may appear to be unique and random, nature likes to keep things simple. This is why patterns and similarities exists in how fluid behaves in different situations under different physical constraints. It is the very nature of how neural networks work that enables them to recognize these patterns and perform predictions. Neural networks, albeit not as dynamic and capable as industrial CFD software at this very moment, so was not facial recognition back in the early 2000s. Therefore, at the current pace at which researchers are looking into developments in machine learning, it is quite likely that artificial intelligence will play a major role in every aspect of peoples' lives in the days to come. This research may contribute to bridge the gap between fluid mechanics and machine learning. The comparisons may help future researchers to come up with a

better way of integrating machine learning into fluid mechanics. The ultimate goal is to establish a data driven approach to solve fluid mechanics problems in real life.

8.2. Future Work

In this work, several parameters were kept constant. Future works can be done to improve the output by treating these parameters as variables.

8.2.1. Changing the neural network model

The model can be changed to add more hidden layers and neurons in each hidden layer. Alternatively, the activation functions can be changed to see what effects they have on the model.

8.2.2. Changing the training data and environment

Training can be done with smaller square obstacles so that any shape can be generated using a number of these smaller square obstacles. The lid velocities can be changed and different Reynolds numbers can be used to make the model learn more.

REFERENCES

[1] Kutz, J.N. Deep learning in fluid dynamics. Journal of Fluid Mechanics 2017; 814:1–4. URL https://dx.doi.org/10.1017/jfm.2016.803.

[2] Baymani, M., Effati, S., Niazmand, H., Kerayechian, A. Artificial neural network method for solving the Navier–Stokes equations. Neural Computing and Applications 2015; 26(4):765–773. URL https://dx.doi.org/10.1007/s00521-014-1762-2.

[3] Mccracken, M.F. Artificial Neural Networks in Fluid Dynamics: A Novel Approach to the Navier-Stokes Equations. Proceedings of the Practice and Experience on Advanced Research Computing 2018. URL https://doi.org/10.1145/3219104.3229262

[4] Sabir, O., Ya, T.M.Y.S.T. A New Artificial Neural Network Approach for Fluid Flow Simulations. Proceedings of the International Conference on Neural Computation Theory and Applications 2014. URL https://doi.org/10.5220/0005157503340338

[5] Guo, X., Li, W., Iorio, F. Convolutional Neural Networks for Steady Flow Approximation. Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining 2016. URL https://doi.org/10.1145/2939672.2939738

[6] Tsunooka, Y., Kokubo, N., Hatasa, G., Harada, S., Tagawa, M., Ujihara, T. High-speed prediction of computational fluid dynamics simulation in crystal growth. CrystEngComm 2018; 20(41):6546–6550. URL 10.1039/c8ce00977e; https://dx.doi.org/10.1039/c8ce00977e.

[7] Marchi, C.H., Suero, R., Araki, L.K. The lid-driven square cavity flow: numerical solution with a 1024 × 1024 grid. Journal of the Brazilian Society of Mechanical

Sciences and Engineering 2009; 31(3):186–198. URL https://dx.doi.org/10.1590/s1678-58782009000300004.

[8] Chorin, A.J. A numerical method for solving incompressible viscous flow problems. Journal of Computational Physics 1967; 2(1):12–26. URL https://dx.doi.org/10.1016/0021-9991(67)90037-x.

[9] Mawarsih, E., Budiana, E.P., Yuana, K.A., Indarto, S., Kamal, D. Simulation of lid-driven cavity with top and bottom moving boundary conditions using implicit finite difference method and staggered grid. AIP Conference Proceedings 2018. URL https://doi.org/10.1063/1.5062719

[10] Saha, K.C. Double Lid Driven Cavity with Different Moving Wall Directions for Low Reynolds Number Flow. International Journal of Applied Mathematics and Theoretical Physics 2018; 4(3):67–67. URL https://doi.org/10.11648/j.ijamtp.20180403.11

[11] Huang, T., Lim, H.C. Simulation of Lid-Driven Cavity Flow with Internal Circular Obstacles. Applied Sciences 2020; 10(13):4583–4583. URL https://doi.org/10.3390/app10134583

APPENDIX

C# Code for CFD Simulation

```
using System;
using System.Collections;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.IO;

namespace CSCFD
{
    class CFD
    {
        public List<double> train_data = new List<double>();
        public List<double> train_label_x = new List<double>();
        public List<double> train_label_y = new List<double>();

        public double[,] u = new double[Variables.gridi + 1, Variables.gridj];
        public double[,] un = new double[Variables.gridi + 1, Variables.gridj];
        public double[,] uc = new double[Variables.gridi, Variables.gridj];
        public double[,] uo = new double[Variables.gridi, Variables.gridj];

        public double[,] v = new double[Variables.gridi, Variables.gridj + 1];
        public double[,] vn = new double[Variables.gridi, Variables.gridj + 1];
        public double[,] vc = new double[Variables.gridi, Variables.gridj];
        public double[,] vo = new double[Variables.gridi, Variables.gridj];

        public double[,] p = new double[Variables.gridi + 1, Variables.gridj + 1];
        public double[,] pn = new double[Variables.gridi + 1, Variables.gridj + 1];
```

```
public double[,] pc = new double[Variables.gridi, Variables.gridj];
public double[,] po = new double[Variables.gridi, Variables.gridj];

public double[,] m = new double[Variables.gridi + 1, Variables.gridj + 1];

public double[,] bn = new double[Variables.gridi, Variables.gridj];

public int i, j, step;
public double dx, dy, dt, delta, error, Re, ipr;


public int object_x;
public int object_y;
public int object_length;
public int object_width;
public string file_name;
public bool save_file;
public double lvel;

public void setParams(int x, int y, int l, int w, string f, double lv, bool s = false)
{
    object_x = x;
    object_y = y;
    object_length = l;
    object_width = w;
    file_name = f;
    save_file = s;
    lvel = lv;
}

public void RunSim()
{
    step = 1;
    dx = 1.0/(Variables.gridj-1);
    dy = 1.0/(Variables.gridi-1);
    dt = 0.001;
    delta = 4.5;
```

```
error = 1.0;
Re = 100.0;

ipr = 1.0;


 // Initializing u
for (i = 0; i <= (Variables.gridi); i++)
{
    for (j = 0; j <= (Variables.gridj - 1); j++)
    {
        u[i, j] = 0.0;
        u[Variables.gridi, j] = lvel;
        u[Variables.gridi - 1, j] = lvel;
        u[0, j] = lvel;
        u[1, j] = lvel;
    }
}

for (i = object_y - 1; i <= (object_width + object_y + 1); i++)
{
    for (j = object_x; j <= (object_length + object_x); j++)
    {
        u[i, j] = 0.0;
    }
}

// Initializing v
for (i = 0; i <= (Variables.gridi - 1); i++)
{
    for (j = 0; j <= (Variables.gridj); j++)
    {
        v[i, j] = 0.0;
    }
}

for (i = object_y; i <= (object_width + object_y); i++)
```

```
{
    for (j = object_x - 1; j <= (object_length + object_x + 1); j++)
    {
        v[i, j] = 0.0;
    }
}
// Initializing p
for (i = 0; i <= (Variables.gridi); i++)
{
    for (j = 0; j <= (Variables.gridj); j++)
    {
        p[i, j] = ipr;
    }
}

//storing simsetup

for (i = 0; i <= 8; i++)
{
    for (j = 0; j <= 8; j++)
    {
        bn[i, j] = 0;
    }
}

for (i = object_y / 3; i <= (object_width + object_y) / 3; i++)
{
    for (j = object_x / 3; j <= (object_length + object_x) / 3; j++)
    {
        bn[i, j] = 1;
    }
}

//storing original velocity values

for (i = 0; i <= (Variables.gridi - 1); i++)
{
```

```
    for (j = 0; j <= (Variables.gridj - 1); j++)
    {
        uo[i, j] = 0.0;
        uo[Variables.gridi - 1, j] = lvel;
        uo[0, j] = lvel;
    }
}

for (i = 0; i <= (Variables.gridi - 1); i++)
{
    for (j = 0; j <= (Variables.gridj - 1); j++)
    {
        vo[i, j] = 0.0;
    }
}

for (i = 0; i <= (Variables.gridi - 1); i++)
{
    for (j = 0; j <= (Variables.gridj - 1); j++)
    {
        po[i, j] = ipr;
    }
}


while (error > 0.00001)
{
    // Solves u-momentum equation
    for (i = 1; i <= (Variables.gridi - 1); i++)
    {
        for (j = 1; j <= (Variables.gridj - 2); j++)
        {
            un[i, j] = u[i, j] - dt * ((((u[i, j + 1] * u[i, j + 1]) - (u[i, j - 1] * u[i, j - 1])) / (2.0 * dx)) +
            ((1.0 / (4.0 * dy)) * (((u[i, j] + u[i + 1, j]) * (v[i, j] + v[i, j + 1])) - ((u[i, j] + u[i - 1, j]) *
            (v[i - 1, j] + v[i - 1, j + 1])))) + ((p[i, j + 1] - p[i, j]) / dx) - ((1.0 / Re) * (((u[i, j + 1] -
            (2.0 * u[i, j]) + u[i, j - 1]) / ((dx * dx))) + ((u[i + 1, j] - (2.0 * u[i, j]) + u[i - 1, j]) / ((dy *
            dy)))))
```

```
        );
    }
}

// Boundary conditions

for (j = 0; j <= (Variables.gridj - 1); j++)
{
    un[0, j] = 2 * lvel - un[1, j];
    un[Variables.gridi, j] = 2 * lvel - un[Variables.gridi - 1, j];
}

for (i = 1; i <= (Variables.gridi - 1); i++)
{
    un[i, 0] = 0.0;
    un[i, Variables.gridj - 1] = 0.0;
}

for (i = object_y - 1; i <= (object_width + object_y + 1); i++)
{
    for (j = object_x; j <= (object_length + object_x); j++)
    {
        un[i, j] = 0.0;
    }
}


// Solves v-momentum
for (i = 1; i <= (Variables.gridi - 2); i++)
{
    for (j = 1; j <= (Variables.gridj - 1); j++)
    {
        vn[i, j] = v[i, j] - dt * ((((v[i + 1, j] * v[i + 1, j]) - (v[i - 1, j] * v[i - 1, j])) / (2.0 * dy)) +
        ((1.0 / (4.0 * dx)) * (((v[i, j] + v[i, j + 1]) * (u[i + 1, j] + u[i, j])) - ((v[i, j] + v[i, j - 1]) *
        (u[i + 1, j - 1] + u[i, j - 1])))) + ((p[i + 1, j] - p[i, j]) / dy) - ((1.0 / Re) * (((v[i, j + 1] -
        (2.0 * v[i, j]) + v[i, j - 1]) / ((dx * dx))) + ((v[i + 1, j] - (2.0 * v[i, j]) + v[i - 1, j]) / ((dy *
        dy)))))
```

```
        );
    }
}

// Boundary conditions

for (i = 1; i <= (Variables.gridi - 2); i++)
{
    vn[i, 0] = -vn[i, 1];
    vn[i, Variables.gridj] = -vn[i, Variables.gridj - 1];
}

for (i = 0; i <= (Variables.gridi - 1); i++)
{
    vn[i, 0] = 0.0;
    vn[i, Variables.gridj] = 0.0;
}

for (i = object_y; i <= (object_width + object_y); i++)
{
    for (j = object_x - 1; j <= (object_length + object_x + 1); j++)
    {
        vn[i, j] = 0.0;
    }
}


// Solves continuity equation
for (i = 1; i <= (Variables.gridi - 1); i++)
{
    for (j = 1; j <= (Variables.gridj - 1); j++)
    {
        pn[i, j] = p[i, j] - dt * delta * ((un[i, j] - un[i, j - 1]) / dx + (vn[i, j] - vn[i - 1, j]) / dy);
    }
}
```

```
// Displaying error
error = 0.0;

for (i = 1; i <= (Variables.gridi - 1); i++)
{
    for (j = 1; j <= (Variables.gridj - 1); j++)
    {
        m[i, j] = ((un[i, j] - un[i, j - 1]) / dx + (vn[i, j] - vn[i - 1, j]) / dy);
        error = error + Math.Abs(m[i, j]);
    }
}

if (step % 1000 == 1)
{
    Console.WriteLine("Error is " + error + " for the step " + step);
}


// Iterating u
for (i = 0; i <= (Variables.gridi); i++)
{
    for (j = 0; j <= (Variables.gridj - 1); j++)
    {
        u[i, j] = un[i, j];
    }
}

// Iterating v
for (i = 0; i <= (Variables.gridi - 1); i++)
{
    for (j = 0; j <= (Variables.gridj); j++)
    {
        v[i, j] = vn[i, j];
    }
}

// Iterating p
```

```
    for (i = 0; i <= (Variables.gridi); i++)
    {
        for (j = 0; j <= (Variables.gridj); j++)
        {
            p[i, j] = pn[i, j];
        }
    }



    step = step + 1;

}

for (i = 0; i <= (Variables.gridi - 1); i++)
{
    for (j = 0; j <= (Variables.gridj - 1); j++)
    {
        uc[i, j] = 0.5 * (u[i, j] + u[i + 1, j]);
        vc[i, j] = 0.5 * (v[i, j] + v[i, j + 1]);
        pc[i, j] = 0.25 * (p[i, j] + p[i + 1, j] + p[i, j + 1] + p[i + 1, j + 1]);
    }
}


for (i = 0; i <= 8; i++)
{
    for (j = 0; j <= 8; j++)
    {
        train_data.Add(bn[i, j]);
    }
}


for (i = 0; i <= (Variables.gridi - 1); i+=Variables.mult)
{
    for (j = 0; j <= (Variables.gridj - 1); j+=Variables.mult)
```

```
    {

        train_label_x.Add(Variables.norm(uc[i, j], -1.0, 1.0));
        train_label_y.Add(Variables.norm(vc[i, j], -1.0, 1.0));


    }
}


// OUTPUT DATA
string data1 = "VARIABLES=\"X\",\"Y\",\"U\",\"V\",\"P\"\n";
data1 += "ZONE  F=POINT\n";
data1 += "I=" + Variables.gridj + ", J=" + Variables.gridi + "\n";



for (i = 0; i < (Variables.gridi); i++)
{
    for (j = 0; j < (Variables.gridj); j++)
    {
        double xpos, ypos;
        xpos = j * dx;
        ypos = i * dy;

        data1 += xpos + "\t" + ypos + "\t" + uc[i, j] + "\t" + vc[i, j] + "\t" + pc[i, j] + "\n";
    }
}

File.WriteAllText(file_name + ".plt", data1);


//DUMPING SIM VALUES

using (StreamWriter writer1 = File.AppendText("training_data.txt"))
{
    writer1.WriteLine(String.Join(",", train_data));
```

```
}
using (StreamWriter writer2 = File.AppendText("training_label_x_grid_size.txt"))
{
    writer2.WriteLine(String.Join(",", train_label_x));
}

using (StreamWriter writer2 = File.AppendText("training_label_y_grid_size.txt"))
{
    writer2.WriteLine(String.Join(",", train_label_y));
}

train_data = new List<double>();
train_label_x = new List<double>();
train_label_y = new List<double>();

        }
    }
}
```

C# Code for ANN Training

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace CSCFD
{
    class NeuralNetwork
    {

        public double TrainNow(int n)
        {
            double cost = 0;


            List<double> pixels = Variables.train_data[n];
            List<double> targets = Variables.train_label[n];



            List<double> hdl = new List<double>();
            List<double> opl = new List<double>();
            List<double> tgl = new List<double>();


            for (int s = 0; s < Variables.train_label_count; s++)
            {
                double target = targets[s];

                tgl.Add(target);
            }

            for (int d = 0; d < Variables.neurons_in_hidden_layer; d++)
```

```
{
    double hdn = 0;

    for (int k = 0; k < Variables.train_data_count; k++)
    {

        hdn += pixels[k] * Variables.w1[d][k];

    }

    hdn = Variables.sigmoid(hdn);

    hdl.Add(hdn);
}

for (int s = 0; s < Variables.train_label_count; s++)
{
    double opn = 0;

    for (int d = 0; d < Variables.neurons_in_hidden_layer; d++)
    {
        opn += hdl[d] * Variables.w2[s][d];
    }

    opn = Variables.sigmoid(opn);

    opl.Add(opn);

    cost += 0.5 * Math.Pow((opn - tgl[s]), 2);

}




for (int d = 0; d < Variables.neurons_in_hidden_layer; d++)
{
```

```
                double inter_value = 0;

                for (int s = 0; s < Variables.train_label_count; s++)
                {
                    inter_value += (opl[s] - tgl[s]) * opl[s] * (1 - opl[s]) * Variables.w2[s][d];
                }

                for (int k = 0; k < Variables.train_data_count; k++)
                {
                    Variables.w1[d][k] -= Variables.learning_rate * inter_value * hdl[d] * (1 - hdl[d]) * pixels[k];
                }
            }



            for (int s = 0; s < Variables.train_label_count; s++)
            {
                for (int d = 0; d < Variables.neurons_in_hidden_layer; d++)
                {
                    Variables.w2[s][d] -= Variables.learning_rate * (opl[s] - tgl[s]) * opl[s] * (1 - opl[s]) * hdl[d];
                }
            }

            cost /= Variables.train_label_count;

            return cost;

        }
    }
}
```

C# Code for ANN Testing

```
using System;
using System.Collections;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.IO;

namespace CSCFD
{
    class Test
    {

        public void TestNow(List<int[]> objects, string tp, string f)
        {

            int[] layers = new int[] { 81, 500, 1000, Variables.nnoutput };
            string[] activation = new string[] { "relu", "relu", "sigmoid" };
            double[] learning_ratesx = new double[] { 0, 0, 0 };
            double[] learning_ratesy = new double[] { 0, 0, 0 };

            NeuralNetwork netxGF = new NeuralNetwork(layers, activation, learning_ratesx);
            NeuralNetwork netyGF = new NeuralNetwork(layers, activation, learning_ratesy);

            NeuralNetwork netxLS = new NeuralNetwork(layers, activation, learning_ratesx);
            NeuralNetwork netyLS = new NeuralNetwork(layers, activation, learning_ratesy);

            netxGF.Load("wb_xvel_" + type + "_GF");
            netyGF.Load("wb_yvel_" + type + "_GF");

            netxLS.Load("wb_xvel_" + type + "_LS");
            netyLS.Load("wb_yvel_" + type + "_LS");
```

```
Console.WriteLine("Loading done.");


dt1 = DateTime.Now;

double[] vlxGF = netxGF.FeedForward(train_data.ToArray());
double[] vlyGF = netyGF.FeedForward(train_data.ToArray());

dt2 = DateTime.Now;

span = dt2 - dt1;

ms = (int)span.TotalMilliseconds;

double[,] ut = new double[Variables.gridi, Variables.gridj];
double[,] vt = new double[Variables.gridi, Variables.gridj];


int counter = 0;

for (int p = 0; p < (Variables.gridi); p++)
{
    for (int q = 0; q < (Variables.gridj); q++)
    {
        ut[p, q] = Variables.dnorm(vlxGF[counter], -1.0, 1.0);
        vt[p, q] = Variables.dnorm(vlyGF[counter], -1.0, 1.0);

        counter++;

    }
}

string data4 = "";
string data5 = "";

double total_gf_errorU = 0;
double total_gf_errorV = 0;
```

```
double total_ls_errorU = 0;
double total_ls_errorV = 0;

//ANN GF OUTPUT

data = "VARIABLES=\"X\",\"Y\",\"U\",\"V\"\n";
data += "ZONE  F=POINT\n";
data += "I=" + Variables.gridj + ", J=" + Variables.gridi + "\n";

data2 = "VARIABLES=\"X\",\"Y\",\"U (Diff)\",\"V (Diff)\"\n";
data2 += "ZONE  F=POINT\n";
data2 += "I=" + Variables.gridj + ", J=" + Variables.gridi + "\n";

data4 = "VARIABLES=\"Grid Points\",\"U (Diff)\"\n";
data4 += "ZONE  F=POINT\n";
data4 += "I=" + Variables.gridj * Variables.gridi + "\n";

data5 = "VARIABLES=\"Grid Points\",\"V (Diff)\"\n";
data5 += "ZONE  F=POINT\n";
data5 += "I=" + Variables.gridj * Variables.gridi + "\n";

int ecounter = 1;

for (int p = 0; p < (Variables.gridi); p++)
{
    for (int q = 0; q < (Variables.gridj); q++)
    {
        double xpos, ypos;
        xpos = q * dx;
        ypos = p * dy;

        data4 += ecounter + "\t" + rel_change(ut[p, q], uc[p, q]) + "\n";
        data5 += ecounter + "\t" + rel_change(vt[p, q], vc[p, q]) + "\n";
        ecounter++;

        total_gf_errorU += rel_change(ut[p, q], uc[p, q]);
        total_gf_errorV += rel_change(vt[p, q], vc[p, q]);
```

```
    }
  }
  total_gf_errorU = total_gf_errorU / (Variables.gridi * Variables.gridj);
  total_gf_errorV = total_gf_errorV / (Variables.gridi * Variables.gridj);

  for (int p = 0; p < (Variables.gridi); p++)
  {
    for (int q = 0; q < (Variables.gridj); q++)
    {
      double xpos, ypos;
      xpos = q * dx;
      ypos = p * dy;

      data += xpos + "\t" + ypos + "\t" + ut[p, q] + "\t" + vt[p, q] + "\n";
    }
  }
  foreach (int[] ob in objects)
  {
    object_x = ob[0] * Variables.div;

    object_y = ob[1] * Variables.div;

    for (i = object_y + 1; i <= (object_width + object_y) - 1; i++)
    {
      for (j = object_x + 1; j <= (object_length + object_x) - 1; j++)
      {
        ut[i, j] = 1.0;
        vt[i, j] = 1.0;
      }
    }
  }

  for (int p = 0; p < (Variables.gridi); p++)
```

```
{
    for (int q = 0; q < (Variables.gridj); q++)
    {
        double xpos, ypos;
        xpos = q * dx;
        ypos = p * dy;

        data2 += xpos + "\t" + ypos + "\t" + rel_change(ut[p, q], uc[p, q]) + "\t" + rel_change(vt[p, q], vc[p,
                 q]) + "\n";
    }
}

data += "TEXT X=5 Y=93 T=\"" + type + " ANN GF Results (" + ms + " ms)\"\n";
data2 += "TEXT X=5 Y=93 T=\"" + type + " ANN GF Differences\"\n";
data4 += "TEXT X=5 Y=93 T=\"" + type + " ANN GF Differences\"\n";
data5 += "TEXT X=5 Y=93 T=\"" + type + " ANN GF Differences\"\n";

foreach (int[] ob in objects)
{
    data += "GEOMETRY X=" + (13.2 + (8.52 * ob[0])) + ", Y=" + (11 + (9.65 * ob[1])) + ", T=SQUARE,
            FC=BLACK\n8.5\n";
    data2 += "GEOMETRY X=" + (13.2 + (8.52 * ob[0])) + ", Y=" + (11 + (9.65 * ob[1])) + ", T=SQUARE,
             FC=BLACK\n8.5\n";
}

File.WriteAllText(file_name + "_ANN_GF.plt", data);
File.WriteAllText(file_name + "_ANN_GF_Error.plt", data2);
File.WriteAllText(file_name + "_ANN_GF_Error_LineU.plt", data4);
File.WriteAllText(file_name + "_ANN_GF_Error_LineV.plt", data5);


Console.WriteLine("GF ANN Done.");

//ANN LS OUTPUT

dt1 = DateTime.Now;
```

```
double[] vlxLS = netxLS.FeedForward(train_data.ToArray());
double[] vlyLS = netyLS.FeedForward(train_data.ToArray());

dt2 = DateTime.Now;

span = dt2 - dt1;

ms = (int)span.TotalMilliseconds;

ut = new double[Variables.gridi, Variables.gridj];
vt = new double[Variables.gridi, Variables.gridj];

counter = 0;

for (int p = 0; p < (Variables.gridi); p++)
{
    for (int q = 0; q < (Variables.gridj); q++)
    {
        ut[p, q] = Variables.dnorm(vlxLS[counter], -1.0, 1.0);
        vt[p, q] = Variables.dnorm(vlyLS[counter], -1.0, 1.0);

        counter++;

    }
}


data = "VARIABLES=\"X\",\"Y\",\"U\",\"V\"\n";
data += "ZONE  F=POINT\n";
data += "I=" + Variables.gridj + ", J=" + Variables.gridi + "\n";

data2 = "VARIABLES=\"X\",\"Y\",\"U (Diff)\",\"V (Diff)\"\n";
data2 += "ZONE  F=POINT\n";
data2 += "I=" + Variables.gridj + ", J=" + Variables.gridi + "\n";

data4 = "VARIABLES=\"Grid Points\",\"U (Diff)\"\n";
data4 += "ZONE  F=POINT\n";
```

```
data4 += "I=" + Variables.gridj * Variables.gridi + "\n";

data5 = "VARIABLES=\"Grid Points\",\"V (Diff)\"\n";
data5 += "ZONE  F=POINT\n";
data5 += "I=" + Variables.gridj * Variables.gridi + "\n";

ecounter = 1;

for (int p = 0; p < (Variables.gridi); p++)
{
    for (int q = 0; q < (Variables.gridj); q++)
    {
        double xpos, ypos;
        xpos = q * dx;
        ypos = p * dy;

        data4 += ecounter + "\t" + rel_change(ut[p, q], uc[p, q]) + "\n";
        data5 += ecounter + "\t" + rel_change(vt[p, q], vc[p, q]) + "\n";
        ecounter++;

        total_ls_errorU += rel_change(ut[p, q], uc[p, q]);
        total_ls_errorV += rel_change(vt[p, q], vc[p, q]);

    }
}

total_ls_errorU = total_ls_errorU / (Variables.gridi * Variables.gridj);
total_ls_errorV = total_ls_errorV / (Variables.gridi * Variables.gridj);

for (int p = 0; p < (Variables.gridi); p++)
{
    for (int q = 0; q < (Variables.gridj); q++)
    {
        double xpos, ypos;
        xpos = q * dx;
        ypos = p * dy;
```

```
        data += xpos + "\t" + ypos + "\t" + ut[p, q] + "\t" + vt[p, q] + "\n";
    }
}

foreach (int[] ob in objects)
{
    object_x = ob[0] * Variables.div;

    object_y = ob[1] * Variables.div;

    for (i = object_y + 1; i <= (object_width + object_y) - 1; i++)
    {
        for (j = object_x + 1; j <= (object_length + object_x) - 1; j++)
        {
            ut[i, j] = 1.0;
            vt[i, j] = 1.0;
        }
    }
}

for (int p = 0; p < (Variables.gridi); p++)
{
    for (int q = 0; q < (Variables.gridj); q++)
    {
        double xpos, ypos;
        xpos = q * dx;
        ypos = p * dy;

        data2 += xpos + "\t" + ypos + "\t" + rel_change(ut[p, q], uc[p, q]) + "\t" + rel_change(vt[p, q], vc[p,
                 q]) + "\n";
    }
}

data += "TEXT X=5 Y=93 T=\"" + type + " ANN LS Results (" + ms + " ms)\"";
data2 += "TEXT X=5 Y=93 T=\"" + type + " ANN LS Differences\"";
data4 += "TEXT X=5 Y=93 T=\"" + type + " ANN LS Differences\"\n";
data5 += "TEXT X=5 Y=93 T=\"" + type + " ANN LS Differences\"\n";
```

```
            foreach (int[] ob in objects)
            {
                data += "GEOMETRY X=" + (13.2 + (8.52 * ob[0])) + ", Y=" + (11 + (9.65 * ob[1])) + ", T=SQUARE,
                        FC=BLACK\n8.5\n";
                data2 += "GEOMETRY X=" + (13.2 + (8.52 * ob[0])) + ", Y=" + (11 + (9.65 * ob[1])) + ", T=SQUARE,
                         FC=BLACK\n8.5\n";
            }

            File.WriteAllText(file_name + "_ANN_LS.plt", data);
            File.WriteAllText(file_name + "_ANN_LS_Error.plt", data2);
            File.WriteAllText(file_name + "_ANN_LS_Error_LineU.plt", data4);
            File.WriteAllText(file_name + "_ANN_LS_Error_LineV.plt", data5);


            Console.WriteLine("LS ANN Done.");

            Console.WriteLine("GF error U " + total_gf_errorU);
            Console.WriteLine("GF error V " + total_gf_errorV);
            Console.WriteLine("LS error U " + total_ls_errorU);
            Console.WriteLine("LS error V " + total_ls_errorU);

            train_data = new List<double>();
            train_label_x = new List<double>();
            train_label_y = new List<double>();




        }

    }
}
```