MINOR PROJECT

ON

BUILDING A SURROGATE MODEL FOR 2D STEADY FLOWANALYSIS USING CNN

Submitted by:

G. Mohith Sai **221710801011**

B.V.D Naga Teja **221710801046**

Yathindra Abhinav.S **221710801049**

Under the Supervision of:

Mr. J. Ramesh

Asst. Professor

DEPARTMENT OF MECHANICAL ENGINEERING

GITAM SCHOOL OF TECHNOLOGY

GITAM, HYDERABAD

December 2020

DEPARTMENT OF MECHANICAL ENGINEERING

GITAM SCHOOL OF TECHNOLOGY

GITAM, HYDERABAD

CERTIFICATE

This is to certify that the thesis entitled **"BUILDING A SURROGATE MODEL FOR 2D STEADY FLOW ANALYSIS USING CNN"** submitted to GITAM University, Hyderabad by **G. Mohith Sai, B.V.D. Naga Teja, Yathindra Abhinav.S** for the award of **Minor Project** in **MechanicalEngineering** is a record of bonafide research work carried out by them. The results embodiedin this thesis have not been submitted to any other University or Institute for the award of anydegree.

Supervisor	**Head of the Department**
Mr. J. Ramesh	Dr. P. Eshwaraiah
Asst. Professor,	Professor
GITAM School of Technology	Department of Mechanical Engineering
Hyderabad Campus.	GITAM School of Technology

Project coordinator

Mr. B. Bhasker
Asst. Professor,
GITAM School of Technology
Hyderabad Campus.

DECLRARTION

We, hereby declare that the work entitled **“BUILDING A SURROGATE MODEL FOR 2D STEADY FLOW ANALYSIS USING CNN”** submitted to the department of Mechanical Engineering, GITAM University is a record of an original work carried out by me under the guidance of **J.Ramesh**. This report is submitted in partial fulfilment of requirement for the award of Minor Project in BACHELOR OF TECHNOLOGY in MECHANICAL ENGINEERING. We declare that the work reported in this report has not been reported and will not be submitted to any university for the award of Degree.

Signature

G. Mohith Sai	221710801011
B.V.D. Naga Teja	221710801046
Yathindra Abhinav.S	221710801049

ABSTRACT

In aerodynamics related design, analysis, and optimization problems, flow fields are simulated using computational fluid dynamics (CFD) solvers. However, CFD simulation is usually a computationally expensive, memory demanding, and time-consuming, repeatable process. These drawbacks of CFD restrict opportunities for design space exploration and prohibit interactive design.

We propose a general and flexible approximation model for real-time prediction of uniform steady laminar flow in a 2D domain based on convolutional neural networks (CNNs). We explored alternatives for the geometry representation and the network architecture of CNNs. We show that convolutional neural networks can estimate the velocity field two orders of magnitude faster than a GPU-accelerated CFD solver and four orders of magnitude faster than a CPU-based CFD solver at the cost of a low error rate.

This approach can provide immediate feedback for real-time design iterations at the early stage of design. Compared with existing approximation models in the aerodynamics domain, CNNs enable an efficient estimation for the entire velocity field. Furthermore, designers and engineers can directly apply the CNN approximation model in their design space exploration algorithms without training extra lower-dimensional surrogate models.

ACKNOWLEDGMENT

Apart from my effort, the success of this Minor Project largely depends on the encouragement and guidance of many others. I take this opportunity to express my gratitude to the people who have helped me in the successful competition of this Minor Project.

We would like to thank J. Ramesh Sir for great guidance and encouragement for doing this project and Burra Bhaskar Sir for making us improve our working skills in this minor project.

We would like to thank respected Eshwaraiah P, Head of the Department of MECHANICAL Engineering, for giving me such an excellent opportunity to expand my knowledge for my own branch and giving me guidelines to present a Minor Project report. It helped me a lot to realize what we study for.

We would like to thank the respected faculties who helped me to make this Minor Project a successful accomplishment.

We would also like to thank my teammates, who helped me to make my work more organized and well-stacked till the end.

CONTENTS

Page No

LIST OF FIGURES

LIST OF TABLES

CHAPTER-1

INTRODUCTION

1.1 Computational Fluid Dynamics:

Computational methods have emerged as powerful techniques for exploring physical and chemical phenomena and solving real engineering problems. In traditional CFD methods, Navier-Stokes equations solve mass, momentum, and energy conservation equations on discrete nodes (the finite difference method, FDM), elements (the finite element method, FEM), or volumes (the finite volume method, FVM). In other words, the nonlinear partial differential equations are converted into a set of nonlinear algebraic equations, which are solved iteratively. Traditional CFD simulation suffers from long response time, predominantly because of the complexity of the underlying physics and the historical focus on accuracy.

1.1.1 Finite Volume Method:

The finite volume method (FVM) is a method for representing and evaluating partial differential equations in the form of algebraic equations. In the finite volume method, volume integrals in a partial differential equation that contain a divergence term are converted to surface integrals, using the divergence theorem. These terms are then evaluated as fluxes at the surfaces of each finite volume. Because the flux entering a given volume is identical to that leaving the adjacent volume, these methods are conservative. Another advantage of the finite volume method is that it is easily formulated to allow for unstructured meshes. This method is used in many computational fluid dynamics packages. "Finite volume" refers to the small volume surrounding each node point on a mesh.

Finite volume methods can be compared and contrasted with the finite difference methods, which approximate derivatives using nodal values, or finite element methods, which create local approximations of a solution using local data, and construct a global approximation by stitching them together. In contrast, a finite volume method evaluates exact expressions for the average value of the solution over some volume, and uses this data to construct approximations of the solution within cells.

The basis of the finite volume method is the integral conservation law. The essential idea is to divide the domain into many control volumes and approximate the integral conservation law on each of the control volumes. For example, as shown in Figure cell i lies between the points at $x_{i-\frac{1}{2}}$ and $x_{i+\frac{1}{2}}$. Note that the points do not have to be equally-spaced.

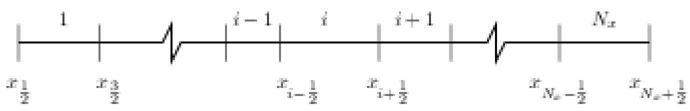


Fig 1.1: Mesh and notation for one-dimensional finite volume method.

The finite volume method is used for the discretization of conservation laws. We gave in the above section two examples of such conservation laws. Let us now present the discretization of general conservation laws by finite volume schemes. As suggested by its name, a conservation law expresses the conservation of a quantity $q(x,t)$. For instance, the conserved quantities may be the energy, the mass, or the number of moles of some chemical species. Let us first assume that the local form of the conservation equation may be written as

$$q_t(x,t) + divF(x,t) = f(x,t)$$

At each point x and each time t where the conservation of q is to be written. In the equation, t denotes the partial time derivative of the entity within the parentheses, div represents the space divergence operator: $divF = \partial F_1/\partial x_1 + \cdots + \partial F_d/\partial x_d, whereF = (F_1, \dots, F_d)^t$ denotes a vector function depending on the space variable x and on the time t, x_i is the i-th space coordinate, for i = 1, . . ., d, and d is the space dimension, i.e. d = 1, 2 or 3; the quantity F is a flux which expresses a transport mechanism of q; the "source term" f represents a possible volumetric exchange, due for instance to chemical reactions between the conserved quantities.

Thanks to the physicist's work, the problem can be closed by introducing constitutive laws which relate q, F, f with some scalar or vector unknown $u(x,t)$, the function of the space variable x and of the time t. For example, the components of u can be pressures, concentrations, molar fractions of the various chemical species by unit volume. . . The quantity q is often given using a known function $\overline{q}$ of $u(x,t)$, of the space variable x and of the time t, that is $q(x,t) = \overline{q}(x,t,u(x,t))$. The quantity F may also be given through a function of the space variable x, the time variable t and of the unknown $u(x,t)$ and (or) by means of the gradient of u at point (x,t). The transport equation, for example, $q(x,t) = u(x,t), F(x,t) = vu(x,t) andf(x,t) = f(x)$; so is the stationary diffusion equation for example, with $q(x,t) =$

$u(x), F(x,t) = -\nabla u(x), and f(x,t) = f(x)$. The source term f may also be given using a function of x, t and $u(x,t)$.

1.1.2 Surrogate Modelling in CFD:

Compared with physical-based simulation and approximation methods, our approach falls into the category of data driven surrogate models (or supervised learning in machine learning). The knowledge of fluid dynamics behaviour can be extracted from large data sets of simulation data by learning the relationship between an input feature vector extracted from geometry and the ground truth data output from a full CFD simulation. Then, without the computationally expensive iterations that a CFD solver requires to allow the flow to converge to its steady state, we can directly predict the steady flow behaviour in a fraction of the time.

The number of iterations and runtime in a CFD solver is usually dependent on the complexity of geometries and boundary conditions, but they are irrelevant to those factors in the prediction process. Note that only the prediction runtime of surrogate models has a significant impact on the design processes. The training time of the surrogate models is irrelevant.

1.2 Convolution Neural Networks:

The name "convolutional neural network" indicates that the network employs a mathematical operation called convolution. Convolutional networks are a specialized type of neural networks that use convolution in place of general matrix multiplication in at least one of their layers.

In deep learning, a convolutional neural network (CNN, or ConvNet) is a class of deep neural networks, most commonly applied to analysing visual imagery. They are also known as shift invariant or space invariant artificial neural networks (SIANN), based on their shared-weights architecture and translation invariance characteristics. They have applications in image and video recognition, recommender systems, image classification, medical image analysis, natural language processing, brain-computer interfaces, and financial time series.

CNNs are regularized versions of multilayer perceptron. Multilayer perceptrons usually mean fully connected networks, i.e., each neuron in one layer is connected to all neurons in the next layer. The "fully-connectedness" of these networks makes them prone to overfitting data. Typical ways of regularization include adding some form of magnitude measurement of weights to the loss function.

1.2.1 Architecture:

A convolutional neural network consists of an input and an output layer and multiple hidden layers. The hidden layers of a CNN typically consist of a series of convolutional layers that convolve with a multiplication or other dot product. The activation function is commonly a ReLU layer. It is subsequently followed by additional convolutions such as pooling layers, fully connected layers, and normalization layers, referred to as hidden layers because their inputs and outputs are masked by the activation function and final convolution.

Convolutional:

When programming a CNN, the input is a tensor with shape (number of images) x (image height) x (image width) x (input channels). Then after passing through a convolutional layer, the image becomes abstracted to a feature map, with shape (number of images) x (feature map height) x (feature map width) x (feature map channels). A convolutional layer within a neural network should have the following attributes:

- Convolutional kernels defined by width and height (hyper-parameters).
- The number of input channels and output channels (hyper-parameter).
- The depth of the Convolution filter (the input channels) must be equal to the number of channels (depth) of the input feature map.

Convolutional layers convolve the input and pass its result to the next layer. This is similar to the response of a neuron in the visual cortex to a specific stimulus. Each convolutional neuron processes data only for its receptive field. Although fully connected feedforward neural networks can be used to learn features and classify data, it is not practical to apply this architecture to images. A very high number of neurons would be necessary, even in a shallow (opposite of deep) architecture, due to the huge input sizes associated with images, where each pixel is a relevant variable. For instance, a fully connected layer for a (small) image of size 100 x 100 has 10,000 weights for each neuron in the second layer. The convolution operation brings a solution to this problem as it reduces the number of free parameters, allowing the network to be deeper with fewer parameters. For instance, regardless of image size, tiling regions of size 5 x 5, each with the same shared weights, requires only 25 learnable parameters. By using regularized weights over fewer parameters, the vanishing gradient and exploding gradient problems seen during backpropagation in traditional neural networks are avoided.

Pooling:

Convolutional networks may include local or global pooling layers to streamline the underlying computation. Pooling layers reduce the data dimensions by combining the outputs of neuron clusters at one layer into a single neuron in the next layer. Local pooling combines small clusters, typically 2 x 2. Global pooling acts on all the neurons of the convolutional layer. Also, pooling may compute a max or average. Max pooling uses the maximum value from each of a cluster of neurons at the prior layer. Average pooling uses the average value from each of a cluster of neurons at the prior layer.

Fully Connected:

Fully connected layers connect every neuron in one layer to every neuron in another layer. It is, in principle, the same as the traditional multilayer perceptron neural network (MLP). The flattened matrix goes through a fully connected layer to classify the images.

Weights:

Each neuron in a neural network computes an output value by applying a specific function to the input values coming from the receptive field in the previous layer. The function that is applied to the input values is determined by a vector of weights and a bias (typically real numbers). Learning, in a neural network, progresses by making iterative adjustments to these biases and weights.

The vector of weights and the bias are called filters and represent particular input features (e.g., a particular shape). A distinguishing feature of CNNs is that many neurons can share the same filter. This reduces memory footprint because a single bias and a single vector of weights are used across all receptive fields sharing that filter instead of each receptive field having its own bias and vector weighting.

1.2.2 Distinguishing Features:

In the past, traditional multilayer perceptron (MLP) models have been used for image recognition. However, due to the full connectivity between nodes, they suffered from the curse of dimensionality, and did not scale well with higher resolution images. A 1000×1000-pixel image with RGB colour channels has 3 million weights, which is too high to process efficiently at scale with full connectivity feasibly.

For example, in CIFAR-10, images are only of size 32×32×3 (32 wide, 32 high, 3 colour channels), so a single fully connected neuron in a first hidden layer of a regular neural network would have 32*32*3 = 3,072 weights. A 200×200 image, however, would lead to neurons that have 200*200*3 = 120,000weights.

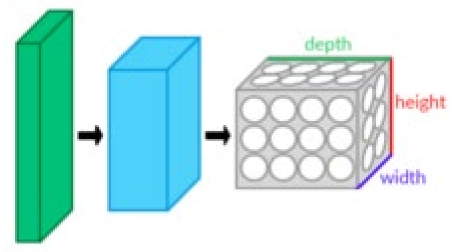


Fig 1.2: CNN layers arranged in 3 dimensions

Convolutional neural networks are biologically inspired variants of multilayer perceptrons that are designed to emulate the behaviour of a visual cortex. These models mitigate the challenges posed by the MLP architecture by exploiting the strong spatially local correlation present in natural images. As opposed to MLPs, CNNs have the following distinguishing features:

- Local connectivity: following the concept of receptive fields, CNNs exploit spatial locality by enforcing a local connectivity pattern between neurons of adjacent layers.
- Shared weights: In CNNs, each filter is replicated across the entire visual field. These replicated units share the same parameterization (weight vector and bias) and form a feature map.
- Pooling: In a CNN's pooling layers, feature maps are divided into rectangular sub-regions, and the features in each rectangle are independently down-sampled to a single value, commonly by taking their average or maximum value.

Together, these properties allow CNNs to achieve better generalization on vision problems. Weight sharing dramatically reduces the number of free parameters learned, thus lowering the memory requirements for running the network and allowing the training of larger, more powerful networks.

1.2.3 Building Blocks:

A CNN architecture is formed by a stack of distinct layers that transform the input volume into an output volume (e.g., holding the class scores) through a differentiable function. A few different types of layers are commonly used. These are further discussed below.

Convolutional Layer:

The convolutional layer is the core building block of a CNN. The layer's parameters consist of a set of learnable filters (or kernels), which have a small receptive field, but extend through the full depth of the input volume. During the forward pass, each filter is convolved across the width and height of the input volume, computing the dot product between the entries of the filter and the input and producing a 2-dimensional activation map of that filter. As a result, the network learns filters that activate when it detects some specific type of feature at some spatial position in the input.

Stacking the activation maps for all filters along the depth dimension forms the full output volume of the convolution layer. Every entry in the output volume can thus also be interpreted as an output of a neuron that looks at a small region in the input and shares parameters with neurons in the same activation map.

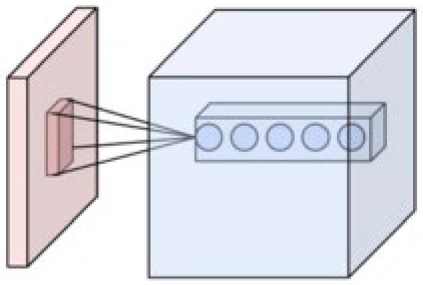


Fig 1.3: Neurons of a convolutional layer (blue), connected to their receptive field (red)

Spatial Arrangement:

Three hyperparameters control the size of the output volume of the convolutional layer: the depth, stride, and zero-padding. The depth of the output volume controls the number of neurons in a layer that connects to the same region of the input volume. These neurons learn to activate for different features in the input. For example, if the first convolutional layer takes the raw image as input, then other neurons along the depth dimension may activate in the presence of various oriented edges, or blobs of colour.

Stride controls how depth columns around the spatial dimensions (width and height) are allocated. When the stride is 1, then we move the filters one pixel at a time. This leads to heavily overlapping receptive fields between the columns, and also to large output volumes. When the stride is 2, then the filters jump 2 pixels at a time as they slide around. Similarly, for any integer $S > 0,$ a stride of S causes the filter to be translated by S units at a time per output.

In practice, stride lengths of $S \geq 3$ are rare. The receptive fields overlap less, and the resulting output volume has smaller spatial dimensions when stride length is increased.

The spatial size of the output volume can be computed as a function of the input volume size W, the kernel field size of the convolutional layer neurons K, the stride with which they are applied S, and the amount of zero padding P used on the border. The formula for calculating how many neurons "fit" in a given volume is given by

$$\frac{W - K + 2P}{S} + 1$$

If this number is not an integer, then the strides are incorrect, and the neurons cannot be tiled to fit across the input volume symmetrically. In general, setting zero padding to be $P = (K - 1)/2$ when the stride is $S = 1$ ensures that the input volume and output volume will have the same size spatially. However, it's not always essential to use all of the neurons of the previous layer. For example, a neural network designer may decide to use just a portion of padding.

Pooling Layer:

Another important concept of CNNs is pooling, which is a form of nonlinear down-sampling. There are several nonlinear functions to implement pooling, among which max pooling is the most common. It partitions the input image into a set of non-overlapping rectangles and, for each such sub-region, outputs the maximum.

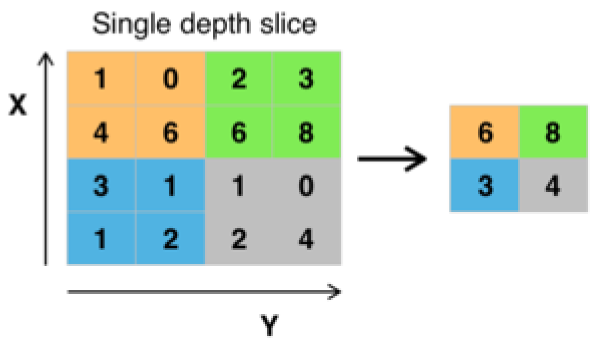


Fig 1.4: Max pooling with a 2x2 filter and stride = 2

The pooling layer operates independently on every depth slice of the input and resizes it spatially. The most common form is a pooling layer with filters of size 2×2 applied with a stride of 2 downsamples at every depth slice in the input by 2 along both width and height, discarding 75% of the activations:

$$f_{X,Y}(S) = max_{a,b=0}^{1} S_{2X+a,2Y+b}$$

In this case, every max operation is over 4 numbers. The depth dimension remains unchanged.

In addition to max pooling, pooling units can use other functions, such as average pooling or ℓ2-norm pooling. Average pooling was often used historically but has recently fallen out of favour compared to max pooling, which performs better in practice.

Due to the aggressive reduction in the representation's size, there is a recent trend towards using smaller filters or discarding pooling layers altogether. "Region of Interest" pooling (also known as RoI pooling) is a variant of max pooling, in which output size is fixed, and input rectangle is a parameter. Pooling is a crucial component of convolutional neural networks for object detection based on Fast R-CNN architecture.

ReLU Layer:

ReLU is the abbreviation of the rectified linear unit, which applies the non-saturating activation function $f(x) = max(0, x)$. It effectively removes negative values from an activation map by setting them to zero. It increases the nonlinear properties of the decision function and the overall network without affecting the receptive fields of the convolution layer.

Other functions are also used to increase nonlinearity, for example, the saturating hyperbolic tangent $f(x) = tanh, f(x) = |tanh|$, and the sigmoid function $\sigma(x) = (1 + e^{-x})^{-1}$. ReLU is often preferred to other functions because it trains the neural network several times faster without a significant penalty to generalization accuracy.

Fully Connected Layer:

Finally, after several convolutional and max pooling layers, the high-level reasoning in the neural network is done via fully connected layers. Neurons in a fully connected layer have connections to all activations in the previous layer, as seen in regular (non-convolutional) artificial neural networks. Their activations can thus be computed as an affine transformation, with matrix multiplication followed by a bias offset (vector addition of a learned or fixed bias term).

Loss Layer:

The "loss layer" specifies how training penalizes the deviation between the predicted (output) and true labels and is usually the final layer of a neural network. Various loss functions may be used. Softmax loss is used for predicting a single class of K mutually exclusive classes.

Sigmoid cross-entropy loss is used for predicting K independent probability values in $[0,1]$. Euclidean loss is used for regressing to real-valued labels $(-\infty, \infty)$.

1.2.4 Choosing Hyperparameters:

CNNs use more hyperparameters than a standard multilayer perceptron (MLP). While the usual rules for learning rates and regularization constants still apply, the following should be kept in mind when optimizing.

Number of Filters:

Since feature map size decreases with depth, layers near the input layer will tend to have fewer filters while higher layers can have more. To equalize computation at each layer, the product of feature values v_a with the pixel position is kept roughly constant across layers. Preserving more information about the input would require keeping the total number of activations (number of feature maps times number of pixel positions) non-decreasing from one layer to the next. The number of feature maps directly controls the capacity and depends on the number of available examples and task complexity.

Filter Shape:

Common filter shapes found in the literature vary greatly and are usually chosen based on the dataset. The challenge is, thus, to find the right level of granularity to create abstractions at the proper scale, given a particular dataset, and without overfitting.

Max Pooling Shape:

Typical values are 2×2. Huge input volumes may warrant 4×4 pooling in the lower layers. However, choosing larger shapes will dramatically reduce the signal's dimension and may result in excess information loss. Often, non-overlapping pooling windows perform best.

1.2.5 Regularization Methods:

Regularization is a process of introducing additional information to solve an ill-posed problem or to prevent overfitting. CNNs use various types of regularization.

1.2.5.1 Empirical:

Dropout:

Because a fully connected layer occupies most of the parameters, it is prone to overfitting. One method to reduce overfitting is a dropout. At each training stage, individual nodes are either "dropped out" of the net with probability $1p$ or kept with probability p, so that a reduced network is left; incoming and outgoing edges to a dropped-out node are also removed. Only the reduced network is trained on the data in that stage. The removed nodes are then reinserted into the network with their original weights. In the training stages, the probability that a hidden node will be dropped is usually 0.5; for input nodes, however, this probability is typically much lower, since information is directly lost when input nodes are ignored or dropped.

At the testing time after training has finished, we would ideally like to find a sample average of all possible 2^n dropped-out networks unfortunately, this is unfeasible for large values of n. n. However, we can find an approximation by using the full network with each node's output weighted by a factor of p, so the expected value of the output of any node is the same as in the training stages. This is the biggest contribution of the dropout method: although it effectively generates 2^n neural nets, and as such, allows for model combination. At test time, only a single network needs to be tested. By avoiding training all nodes on all training data, dropout decreases overfitting. The method also significantly improves training speed. This makes the model combination practical, even for deep neural networks. The technique reduces node interactions, leading them to learn more robust features that better generalize to new data.

Stochastic Pooling:

A major drawback to dropout is that it does not have the same benefits for convolutional layers, where the neurons are not fully connected. In stochastic pooling, the conventional deterministic pooling operations are replaced with a stochastic procedure, where the activation within each pooling region is picked randomly according to a multinomial distribution, given by the activities within the pooling region. This approach is free of hyperparameters and can be combined with other regularization approaches, such as dropout and data augmentation.

Artificial Data:

Since the degree of model overfitting is determined by both its power and the amount of training it receives, providing a convolutional network with more training examples can reduce overfitting.

1.2.5.2 Explicit:

Early Stopping:

One of the simplest methods to prevent overfitting of a network is to simply stop the training before overfitting has had a chance to occur. It comes with the disadvantage that the learning process is halted.

Number of Parameters:

Another simple way to prevent overfitting is to limit the number of parameters, typically by limiting the number of hidden units in each layer or limiting network depth. For convolutional networks, the filter size also affects the number of parameters. Limiting the number of parameters restricts the predictive power of the network directly, reducing the complexity of the function that it can perform on the data, and thus limits the amount of overfitting. This is equivalent to a "zero norm".

Weight Decay:

A simple form of added regularizer is weight decay, which adds error, proportional to the sum of weights (L1 norm) or squared magnitude (L2 norm) of the weight vector, to each node's error. The level of acceptable model complexity can be reduced by increasing the proportionality constant, thus increasing the penalty for large weight vectors.

Max Norm Constraints:

Another form of regularization is to enforce an absolute upper bound on the magnitude of the weight vector for every neuron and use projected gradient descent to enforce the constraint. In practice, this corresponds to performing the parameter update as normal, and then enforcing the constraint by clamping the weight vector $\bar{w}$ of every neuron to satisfy $||w^{-}||_2 < c$. Typical values of c are in the order of 3–4. Some papers report improvements when using this form of regularization.

1.2.6 Applications:

Some of the practical applications of CNN are:

- Image Recognition
- Video Analysis
- Natural Language Processing
- Anomaly Detection
- Drug Discovery
- Time Series Forecasting

1.3 Image Segmentation:

In digital image processing and computer vision, image segmentation is the process of partitioning a digital image into multiple segments (sets of pixels, also known as image objects). The goal of segmentation is to simplify and/or change the representation of an image into something more meaningful and easier to analyze. Image segmentation is typically used to locate objects and boundaries (lines, curves, etc.) in images. More precisely, image segmentation is the process of assigning a label to every pixel in an image such that pixels with the same label share specific characteristics.

The result of image segmentation is a set of segments that collectively cover the entire image or a set of contours extracted from the image (see edge detection). Each of the pixels in a region are similar concerning some characteristic or computed property, such as color, intensity, or texture. Adjacent regions are significantly different with respect to the same characteristics.

1.3.1 Classes of Segmentation Techniques:

There are three classes of segmentation techniques.

- Classical approaches
- AI-based techniques
- Techniques not falling into the above two categories.

1.3.2 Groups of Image Segmentation:

- Semantic segmentation is an approach detecting, for every pixel, belonging class of the object. For example, when all people in a figure are segmented as one object and background as one object.

- Instance segmentation is an approach that identifies, for every pixel, a belonging instance of the object. It detects each distinct object of interest in the image. For example, when each person in a figure is segmented as an individual object.

1.3.3 Thresholding:

The simplest method of image segmentation is called the thresholding method. This method is based on a clip-level (or a threshold value) to turn a gray-scale image into a binary image.

The key to this method is to select the threshold value (or values when multiple-levels are chosen). Several popular methods are used in industry, including the maximum entropy method, balanced histogram thresholding, Otsu's method (maximum variance), and k-means clustering.

1.3.4 Compression-Based Methods:

Compression based methods postulate that the optimal segmentation is the one that minimizes, over all possible segmentations, the coding length of the data. The connection between these two concepts is that segmentation tries to find patterns in an image, and any regularity in the image can be used to compress it. The method describes each segment by its texture and boundary shape. Each of these components is modeled by a probability distribution function, and its coding length is computed as follows:

1. The boundary encoding leverages the fact that regions in natural images tend to have a smooth contour. This prior is used by Huffman coding to encode the difference chain code of the contours in an image. Thus, the smoother a boundary is, the shorter coding length it attains.
2. Texture is encoded by lossy compression in a way similar to the minimum description length (MDL) principle, but here the length of the data given the model is approximated by the number of samples times the entropy of the model. The texture in each region is modeled by a multivariate normal distribution whose entropy has a closed form expression. An interesting property of this model is that the estimated entropy bounds the true entropy of the data from above. This is because among all distributions with a given mean and covariance, normal distribution has the largest entropy. Thus, the true coding length cannot be more than what the algorithm tries to minimize.

For example, when the textures in an image are similar, such as in camouflage images, stronger sensitivity, and thus lower quantization is required.

1.3.5 Edge Detection:

Edge detection is a well-developed field on its own within image processing. Region boundaries and edges are closely related, since there is often a sharp adjustment in intensity at the region boundaries. Edge detection techniques have therefore been used as the base of another segmentation technique. The edges identified by edge detection are often disconnected. To segment an object from an image, however, one needs closed region boundaries. The desired edges are the boundaries between such objects or spatial-taxons.

1.3.6 Dual Clustering Method:

This method is a combination of three characteristics of the image: partition of the image based on histogram analysis is checked by high compactness of the clusters (objects), and high gradients of their borders. For that purpose, two spaces have to be introduced: one space is the one-dimensional histogram of brightness $H = H(B)$; the second space is the dual 3-dimensional space of the original image itself $B = B(x, y)$. The first space allows measuring how compactly the brightness of the image is distributed by calculating a minimal clustering kmin. Threshold brightness T corresponding to kmin defines the binary (black-and-white) image – bitmap $b = \varphi(x, y)$, where $\varphi(x, y) = 0$, if $B(x, y) < T$, and $\varphi(x, y) = 1$, if $B(x, y) \geq T$. The bitmap b is an object in dual space. On that bitmap a measure has to be defined reflecting how compact distributed black (or white) pixels are. So, the goal is to find objects with good borders. For all T the measure $MDC = G/(k \times L)$ has to be calculated (where k is the difference in brightness between the object and the background, L is the length of all borders, and G is mean gradient on the borders). Maximum of MDC defines the segmentation.

1.3.7 Model-Based Segmentation:

The central assumption of model-based approaches is that the structures of interest tend to a particular shape. Therefore, one can seek a probabilistic model that characterizes the shape and its variation. When segmenting an image, constraints can be imposed using this model as a prior. Such a task may involve:

1. Registration of the training examples to a common pose
2. Probabilistic representation of the variation of the registered samples

3. Statistical inference between the model and the image. Other essential methods in the literature for model-based segmentation include active shape models and active appearance models.

1.3.8 Multi-Scale Segmentation:

Image segmentations are computed at multiple scales in scale space and sometimes propagated from coarse to fine scales; see scale-space segmentation. Segmentation criteria can be arbitrarily complex and may take into account global as well as local criteria. A common requirement is that each region must be connected in some sense.

1.3.8.1 One-Dimensional Hierarchical Signal Segmentation:

Witkin's seminal work in scale space included the notion that a one-dimensional signal could be unambiguously segmented into regions, with one scale parameter controlling the segmentation scale.

A key observation is that the zero-crossings of the second derivatives (minima and maxima of the first derivative or slope) of multi-scale-smoothed versions of a signal form a nesting tree, which defines hierarchical relations between segments at different scales. Specifically, slope extrema at coarse scales can be traced back to corresponding features at fine scales when a slope maximum and slope minimum annihilate each other at a larger scale. The three segments that they separated merge into one segment and defining the hierarchy of segments.

1.3.8.2 Image Segmentation and Primal Sketch:

There have been numerous researches works in this area, out of which a few have now reached a state where they can be applied either with an interactive manual intervention (usually with application to medical imaging) or fully automatically.

1.3.9 Semi-Automatic Segmentation:

In one kind of segmentation, the user outlines the region of interest with the mouse clicks, and algorithms are applied so that the path that best fits the image's edge is shown. Techniques like SIOX, Livewire, Intelligent Scissors, or IT-SNAPS are used in this kind of segmentation. In an alternative kind of semi-automatic segmentation, the algorithms return a spatial-taxon (i.e., foreground, object-group, object, or object-part) selected by the user or designated via prior probabilities.

1.3.10 Trainable Segmentation:

Most of the aforementioned segmentation methods are based only on color information of pixels in the image. Humans use much more knowledge when performing image segmentation, but implementing this knowledge would cost considerable human engineering and computational time, and would require a vast domain knowledge database, which does not currently exist. Trainable segmentation methods, such as neural network segmentation, overcome these issues by modeling the domain knowledge from a dataset of labeled pixels.

An image segmentation neural network can process small areas of an image to extract simple features such as edges. Another neural network, or any decision-making mechanism, can then combine these features to label the areas of an image accordingly. A type of network designed this way is the Kohonen map.

U-Net is a convolutional neural network that inputs an image and outputs a label for each pixel. U-Net was initially developed to detect cell boundaries in biomedical images. U-Net follows classical autoencoder architecture. As such, it contains two sub-structures. The encoder structure follows the traditional stack of convolutional and max pooling layers to reduces the receptive field as it goes through the layers. It is used to capture the context in the image. The decoder structure utilizes transposed convolution layers for upsampling so that the end dimensions are close to that of the input image. Skip connections are placed between convolution and transposed convolution layers of the same shape to preserve details that would have been lost otherwise.

In addition to pixel-level semantic segmentation tasks which assign a given category to each pixel, modern segmentation applications include instance-level semantic segmentation tasks in which each individual in a given category must be uniquely identified, as well as panoptic segmentation tasks which combines these two tasks to provide a more complete scene segmentation.

1.3.11 Applications:

Some of the practical applications of image segmentation are:

- Content-based image retrieval.
- Machine vision.
- Medical imaging, including volume rendered images from computed tomography and magnetic resonance imaging.

- Object Detection
- Recognition Tasks
- Traffic control systems
- Video surveillance

Several general-purpose algorithms and techniques have been developed for image segmentation. To be useful, these techniques must typically be combined with a domain's specific knowledge to solve the domain's segmentation problems effectively.

CHAPTER-2

LITERATURE REVIEW

2.1 Literature Survey:

[1] In the review paper, they have proposed a general CNN-based approximation model for predicting the velocity field in 2D non-uniform steady laminar flow. They have shown that CNN-based surrogate models can estimate the velocity field two times faster than a GPU-accelerated CFD solver or four times faster magnitude than a CPU-based CFD solver with a low error rate.

In traditional CFD methods, Navier-Stokes equations solve mass, momentum, and energy conservation equations on discrete nodes, but this method they haven't used for the simulation; instead, they have used the Lattice Boltzmann Method (LBM). This is because the traditional Navier-Strokes equation solution takes a long response time, predominantly because of the complexity of the underlying physics and the historical focus on accuracy. So, LBM has been emerged as an alternative for traditional CFD simulation due to its ability to operate on complex geometry shapes and to its trivially parallel implementation nature.

[6] In this paper, they have used LBM to generate all the training data for the CNNs, and they have compared the speed of the CNN surrogate model with both traditional CPU and GPU accelerated LBM solvers. Compared with physical-based simulation and approximation methods, our approach falls into the category of data-driven surrogate models (or supervised learning in machine learning). The basic idea of using a surrogate model is it is relatively cheap compared with the traditional CFD method as it requires high CPU and GPU configurations for getting solutions, and it also takes a lot of time to produce results. So, these surrogate models are less expensive. The CFD surrogate models are constructed from physical experiments.

In our review paper, we use CNNs to model large-scale nonlinear general CFD analysis for a restricted class of flow conditions. Their approach applies CNNs to model large-scale nonlinear general CFD analysis for a restricted class of flow conditions. So, in the paper, they have used the CNNs based CFD prediction to get two to four orders of magnitude speedup compared with the LBM method CFD solution at a cost of low error rate.

Since CNN's have been proven successful in geometry representation learning and per-pixel prediction in images, their surrogate models have three key components.

1. In the first component, they have adopted a signed distance function (SDF) as a flexible and general geometric representation for CNNs.
2. In the second component, they have multiplied convolutional encoding layers to extract abstract and high-level geometric representations.
3. Finally, multiple convolutional decoding layers map the abstract geometric representations into the CFD velocity field.

In the paper, they have used SDF, which is sampled on a 2-D Cartesian grid, as the geometry representation. They have used SDF because it is a universal representation for different geometry shapes and works efficiently with neural networks. To validate the efforts of computing SDF values, they have compared SDF geometry representation with a simple binary representation, where a grid value is 1 if and only if it is within or on the boundary of geometry shapes. This type of binary representation is easy to compute, and the values in it don't provide any global information.

[2] In this paper, they have used the deep neural network as real-time surrogate models to substitute time-taking LBM CFD simulation. LBM represents velocity fields as regular Cartesian grids, or lattices, where each lattice cell represents the velocity field at that location. In LBM, geometry is represented by assigning an identifier to each lattice cell. This specific identifier defines whether a cell lies on the boundary or in the fluid domain. In the paper, the encoding part uses SDF as an input, and it is stacked with a convolutional layer that extracts geometry features from the SDF image (SDF representation). In the decoding part, they used the inverse convolution process, which does the reverse of what the convolution layer does. Deconvolution layers multiply each input value by a filter elementwise and sum over the resulting output windows. It is a convolutional layer with forward and backward passes reversed to be simple.

In the network Architecture part, they have used a shared-encoding and separated-decoding architecture for our CFD surrogate model. In our shared-encoding and separated-decoding architecture, multiple convolutional layers are used to extract an abstract geometry representation, and numerous decoding parts use the abstract geometry representation to generate the various components of the CFD velocity field.

In the experiments, they have evaluated the proposed method of 2D geometry domains. In the 2D domain, they have trained the CNNS using a collection of 2D geometric primitives and evaluate the prediction over a validation dataset of 2D primitives and a test dataset of 2D car shapes. So, they have first compared the SDF representation with the binary representations and show that SDF outperforms binary representations in CFD prediction. After that, the CNN-based surrogate model outperforms a patch-based linear regression baseline in predicting CFD for both 2D geometry shapes. Finally, they have compared the proposed method's time cost with the traditional LBM solution to highlight the proposed method's speedup.

They have evaluated the effectiveness of SDF geometric representation compared to the binary representations in the same CNN architectures. In the experiments, the binary representations use 256 × 128 binary matrices to represent the geometry. Its element is 1 if and only if the corresponding position is on the geometry boundary or inside the geometry. When we compare the results, the errors we have got using SDF are smaller than using binary representations. So deep neural networks are the same, and the results show that the SDF representations are more effective than the binary representations.

The primary motivation of our surrogate model is that CNN prediction of non-uniform steady laminar flow is considerably faster than traditional LBM solvers. Furthermore, CNN utilizes GPU to evaluate the CFD results. LBM is well suited to massively parallel architectures, as each cell in the lattice can be updated independently at every time step.

The results of the time cost for CNNs and the time cost of the shared encoding and separated encoding are:

1. In the results, they have got that the average time cost decreases significantly as the batch size becomes.
2. The separated encoding takes more time than the shared encoding on different batch sizes. The prediction accuracy of shared and separated encoding architectures is close, but the shared encoding outperforms the separated encoding in terms of time cost.

So, when they have compared speedups of shared encoding CNNs with LBM on CPU and GPU, the results are shown below.

1. GPU accelerated CNN model achieves up to 12K speedup compared to traditional LBM solvers running on a single CPU core.
2. The CNN model achieves up to 292 speedups compared to the GPU-accelerated LBM solver.
3. The speedup increases as batch size increases because the overtime in using GPU is amortized.

So, even though for many domains, such as architectural design, low Reynolds number flows are usually sufficient, they would like to explore higher Reynolds number flows in the future to extend the approach to other design optimization areas.

2.2 Motivation for The Present Work:

While traditional CFD methods produce high-accuracy results, they are computationally expensive and the time required to obtain results is often measured in hours, or even days for complex prototypes. In many domains, CFD analysis becomes one of the most intensive and time-consuming processes; therefore, it is typically used only for final design validation.

During the early design stages, designers often need to quickly iterate over multiple design alternatives to make preliminary decisions and they usually do not require high fidelity simulations. Significant efforts have been made to make conceptual design software environments interactive, so that designers can get immediate or real-time feedback for continuous design iterations. One of the alternatives to high accuracy CFD simulation is the use of fast approximation models, or surrogates.

In the design optimization practice, multiple design alternatives are explored, evaluated and optimized at the same time. Guiding this process with fluid dynamics-based performance is difficult due to the slow feedback from conventional CFD solvers. The idea of speed-accuracy trade-offs also leads to practical design space exploration and design optimization in the aerodynamics domain by using fast surrogate models.

Data-driven surrogate models become more and more practical and important because many design, analysis and optimization processes generate high volume CFD physical or simulation data. Over the past years, deep learning approaches have shown great successes in learning representations from data, where features are learned end-to-end in a compositional hierarchy. This is in contrast to the traditional approach which requires the hand-crafting of features by domain experts. For data that have strong spatial and/or temporal dependencies, Convolutional Neural Networks (CNNs) have been shown to learn invariant high-level features that are informative for supervised tasks.

In particular, we propose a general CNN-based approximation model for predicting the velocity field in 2D non-uniform steady laminar flow. We show that CNN based surrogate models can estimate the velocity field two orders of magnitude faster than a GPU-accelerated

CFD solver or four orders of magnitude faster than a CPU-based CFD solver at a cost of a low error rate. Another benefit of using a CNN-based surrogate model is that we can collect training data from either physical or simulation results. The knowledge of fluid dynamics behaviour can then be extracted from data sets generated by different CFD solvers, using different solution methods. Traditional high accuracy simulation is still an important step to validate the final design, but fast CFD approximation models have wide applications in the conceptual design and the design optimization process.

CHAPTER-3

OBJECTIVES AND METHODOLOGY

3.1 Objectives:

To prepare a surrogate model for 2-dimensional steady flow for fluid simulations using CNN.

3.2 Methodology:

- For training and testing of the surrogate model, we have created SDF images of 6 shapes such as circle, ellipse, rectangle, and square in a domain with variation in positions.
- For creating the dataset of steady-state fluid flow simulations, we have used ANSYS with the same model as in corresponding SDF images with the fluid over the boundary in the domain.
- For creating a surrogate model, we have used the UNet architecture model, and we have trained and tested using the dataset created using the above methods.
- Then by changing various parameters in the CNN code, we have optimized the model by reducing the prediction errors.

3.2.1 Geometry Representation:

Geometry can be represented in multiple ways, such as boundaries and geometric parameters. However, those representations are not useful for neural networks since the vectors' semantic meaning varies. In this minor project, we use a Signed Distance Function (SDF) sampled on a Cartesian grid as the geometry representation. SDF provides a universal representation for different geometry shapes and works efficiently with neural networks.

Given a discrete representation of a 2D geometry on a Cartesian grid, to compute the signed distance function,

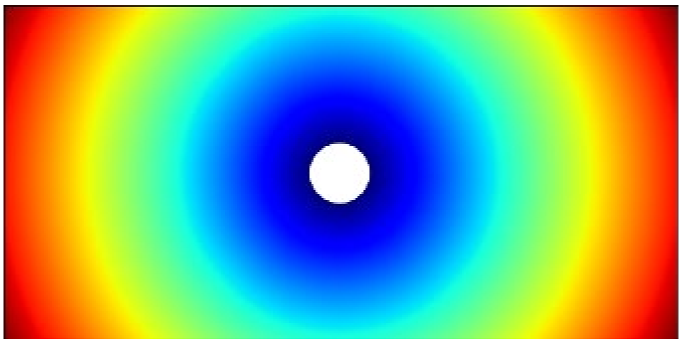


Fig 3.1: SDF Image Sample

We first create the zero-level set, which is a set of points (i,j) that give the geometry boundary (surface) in a domain $\Omega \subset R^2$:

$$Z = \{(i,j) \in R^2 : f(i,j) = 0\}$$

where f is the level set function s.t. $f(i,j) = 0$ if and only if $(i.j)$ is on geometry shape boundary $f(i,j) > 0$ if an only if (i,j) is outside the geometry shape boundary, $f(i,j) < 0$ if and only if (i,j) is inside the geometry shape boundary.

A signed distance function D (i, j) associated with a level set function $f(i,j)$ is defined by

$$D(i,j) = min_{(\hat{i},\hat{j})}|(i,j) - (\hat{i},\hat{j})|signf(i,j)$$

$D(i,j)$ is an oriented distance function, and it measures the distance of a given point (i,j) from the nearest boundary of a closed geometrical shape Z, with the sign determined by whether (i,j) is inside or outside the shape. Also, every point outside the boundary has a value equal to its distance from the interface (shown in the fig 3.1).

3.2.2 UNet Model Architecture:

Here for our model, we are using UNet architecture for building a surrogate model for our CFD Analysis. Generally, the UNet model is more accurate for the greyscale images, so we changed our data set images into greyscale.

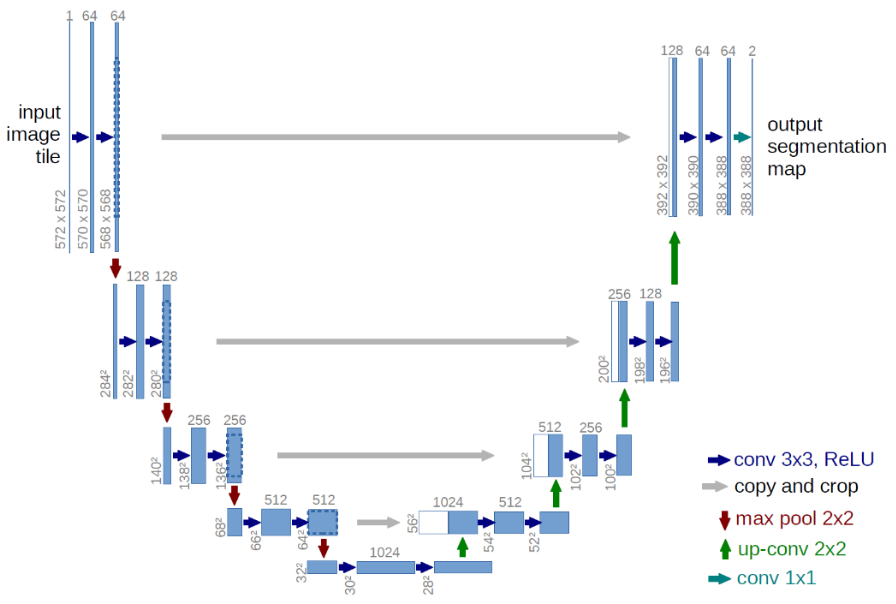


Fig 3.2: UNet Architecture

CHAPTER-4
NUMERICAL ANALYSIS

4.1 Numerical Analysis:

CNNs have been proven successful in geometry representation learning and per-pixel prediction in images. The other motivation for adopting CNNs is their memory efficiency. So here, for solving the problems, we used ANSYS 19.2 software. For training the CNN, we require different shapes. We used SDF images. We used a total of 6 different images and with a set of 11 images. Then we auto traced SDF images in Solid Works 2018 and imported them into Ansys software, and then with tools of FLUENT, we solved our problems.

Our surrogate models have three key components. First, we adopt signed distance functions as a flexible and general geometric representation for convolutional neural networks. Second, we use multiple convolutional encoding layers to extract abstract and high-level geometric representations. Finally, multiple convolutional decoding layers map the abstract geometric representations into the computational fluid dynamics velocity field. In the rest of this section, we describe the critical components in turn.

4.1.1 CFD Simulation:

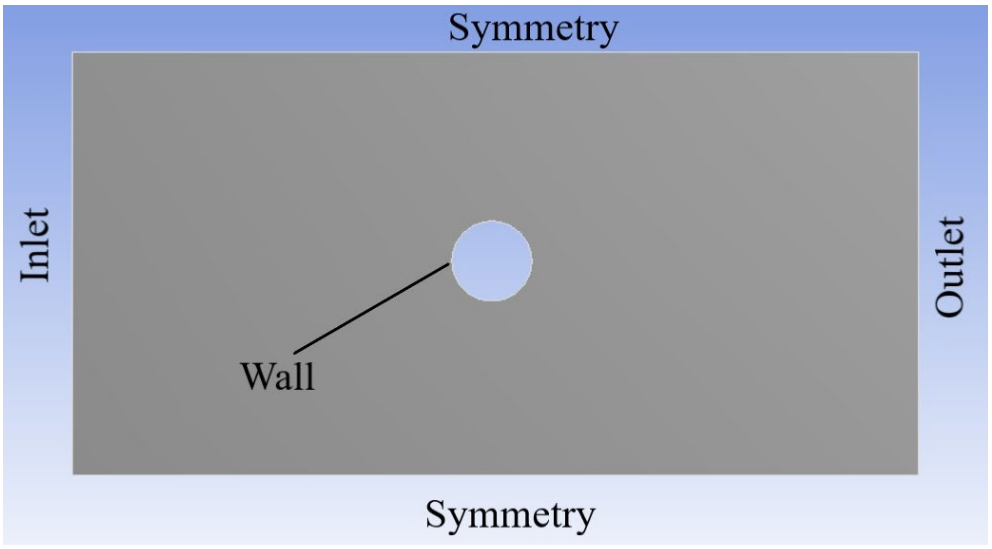


Fig 4.1: Demonstration of Boundaries

So, our boundary conditions for the problem are Inlet, Outlet, Wall, and Symmetry. At the inlet, the velocity of the fluid will be given. At the outlet, the outlet pressure condition is given. The wall has the no-slip condition, which means at the surface of the wall, there is no velocity of the fluid, which more or less conveys the fluid particles at the surface of the wall are sticking to it. Symmetry boundary conditions are used when the physical geometry of interest, and the expected pattern of the flow/thermal solution, have mirror symmetry. They can also be used to model zero-shear slip walls in viscous flows. ANSYS FLUENT assumes a zero flux of all quantities across a symmetry boundary.

Here for our problem, we used quadrilateral mesh and gave appropriate conditions for better mesh quality and less skewness. A mesh is usually considered to having higher quality than another mesh if it improves at least one of the most crucial simulation properties; Time to convergence, stability, or accuracy without affecting the others negatively. Skewness is defined as the difference between the shape of the cell and the shape of an equilateral cell of equivalent volume.

Table 4.1: Properties of Fluid (Air)

Property	Value
Density	1.225 kg/m^3
Viscosity	1.7894e-05 kg/m-s

In Fluent, we used laminar flow as we defined our problem to be laminar flow. And the energy equations were turned off as we are only concentrating on flow parameters velocity and pressure but not the temperature. Then for the algorithm, we used coupled algorithm.

Selecting Coupled from the Pressure-Velocity Coupling drop-down list indicates that you are using the pressure-based coupled algorithm, described in Coupled Algorithm in the Ansys Theory Guide. This solver offers some advantages over the pressure-based segregated algorithm. The pressure-based coupled algorithm obtains a more robust and efficient single-phase implementation for steady-state flows. It is not available for cases using the Non-Iterative Time Advancement option (NITA).

We made 66 simulations for the models with appropriate boundary conditions, and here are few results.

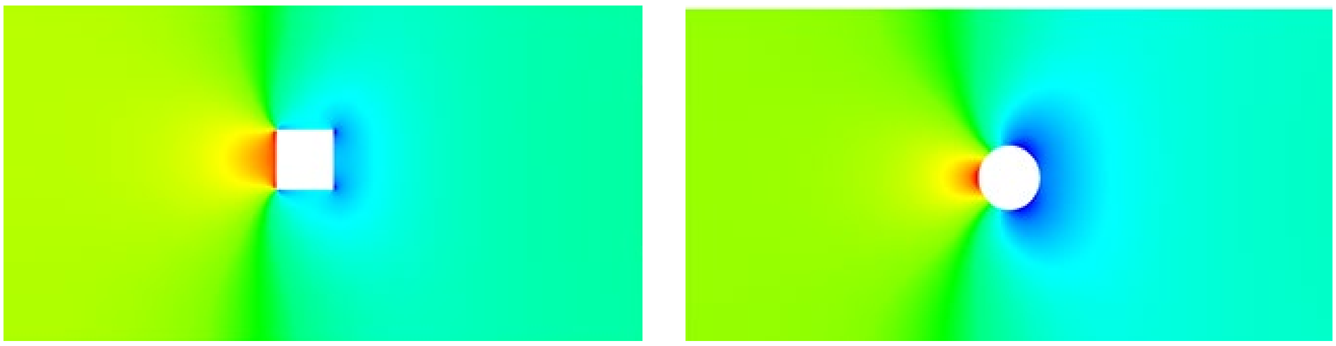


Fig 4.2: Pressure Contours

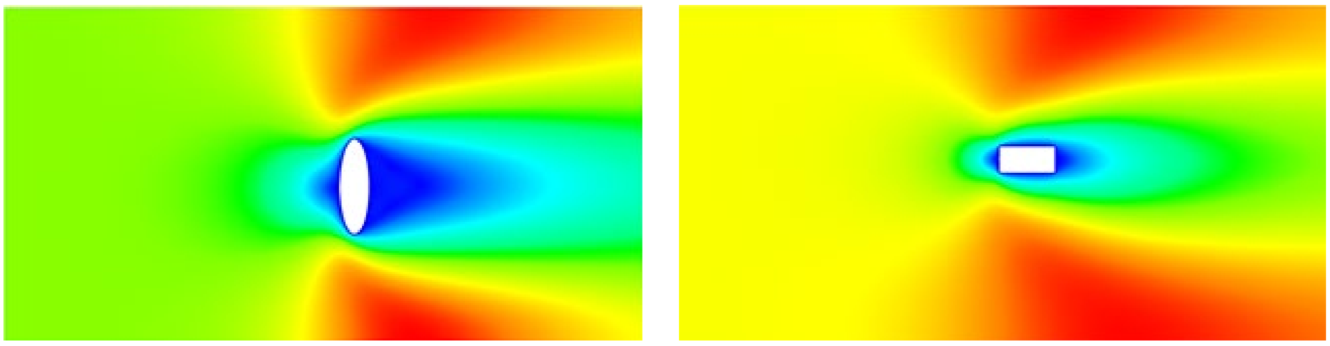


Fig 4.3: Velocity Contours

4.1.2 CNN:

So, for making a surrogate model for CFD using CNN we have opted for Image Segmentation. U-Net is our CNN model. For, encoding process we have used MobileNetV2 as pretrained encoder for grey scale images which would help us in gaining a better accuracy. For MobileNetV2 we are using an arbitrary value alpha = 0.35 and later tune that value to decrease loss and increase accuracy.

Here we are using SIGMOID activation function for more non linearity of the inputs. Here we have 4 blocks of encoding so we should have 4 blocks of decoding. To connect the encoding and decoding layers we are using the skip function. To avoid truncation errors, we are using normalization functions to input images so that it gives only 0's and 1's. ReLU function is used in the encoding and decoding blocks for better accuracy.

4.1.3 UNet Model:

For our model we are using an input shape of (None, 256, 256, 3) and with an output shape of (None, 256, 256, 1). We used a batch size of 2, with a number of epochs of 30. For our model, we have a total number of parameters as 416,209, and trainable parameters are

409,025. In total, we have four convolution blocks and four deconvolution blocks with 319 layers in CNN. For our model, we have used an autoencoder for better accuracy. So, MobileNetV2 is a pre-trained encoder with higher accuracy, so we opted for that pre-trained encoder.

We trained our CNN model to minimize the Dice loss. Dice loss is based on the Sorensen-Dice coefficient or Tversky index, which attaches similar importance to false positives and false negatives, and is more immune to the data-imbalance issue. With the proposed training objective, we observe a significant performance boost on a wide range of data imbalanced NLP tasks.

$$D = \frac{2\sum_i^N p_i\, g_i}{\sum_i^N p_i^2 + \sum_i^N g_i^2}$$

$$f(x) = 1 - D$$

D - Dice coefficient, pi - pixel values of prediction gi - pixel values of ground truth $f(x)$ - Dice loss.

CHAPTER-5

RESULTS AND DISCUSSIONS

5.1 CNN Results:

We trained our UNet model with the dataset that we have. In any CNN model, the metrics are most important for an increase in accuracy and validation.

5.1.1 Pressure UNet:

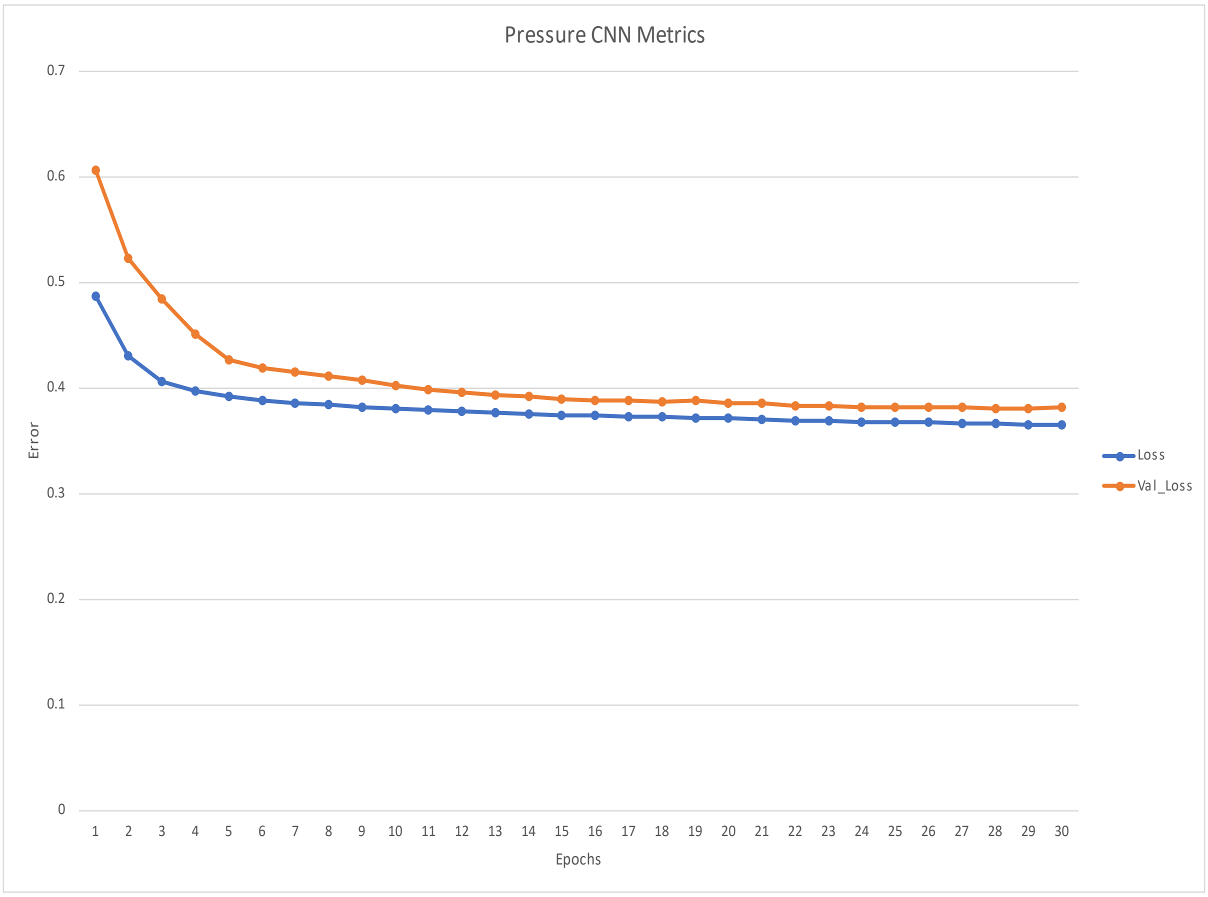


Fig 5.1: Metrics of the Pressure CNN

Our main aim is to reduce the loss of the CNN model. In the graph, the Pressure UNet training loss was a bit lower than the validation loss for our CNN validation. We have used six images from our data set, so it was a bit more than the training loss.

5.1.2 Velocity UNet:

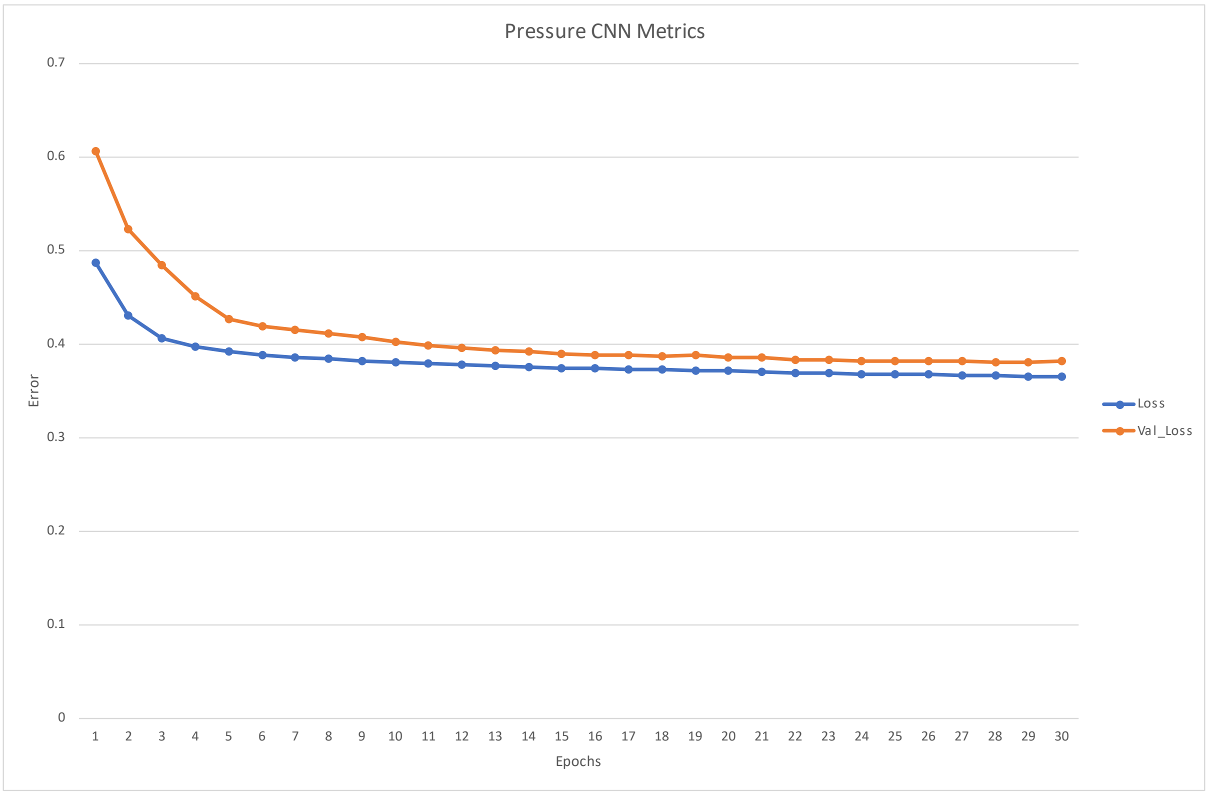
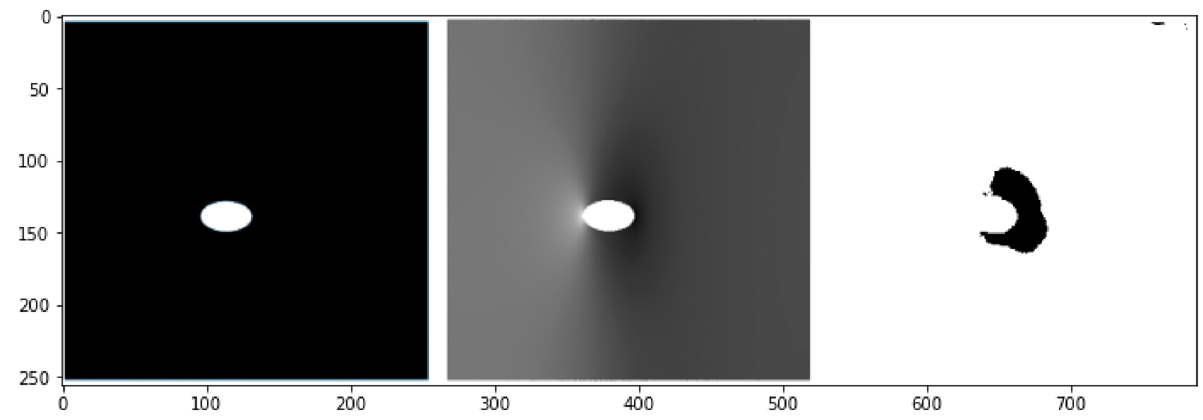


Fig 5.2: Metrics of the Velocity CNN

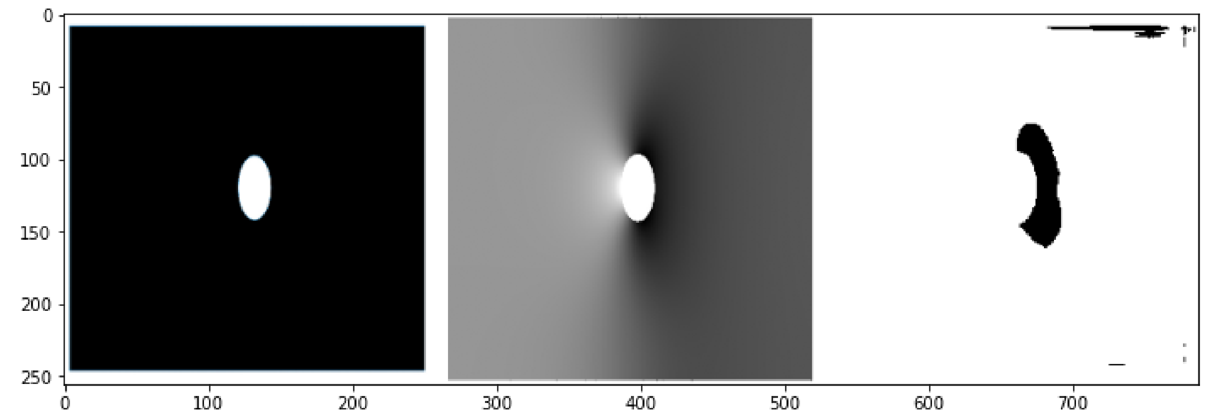
Our main aim is to reduce the loss of the CNN model. In the graph, the velocity CNN metrics, and you can see the Velocity UNet training loss was a bit lower than the validation loss for validation of our CNN. We have used six images from our data set, so it was a bit more than the training loss. We can see that the Velocity UNet loss was minimal than that of the Pressure UNet as the patterns were more recognizable than the pressure contours.

5.1.3 Predicted Images:

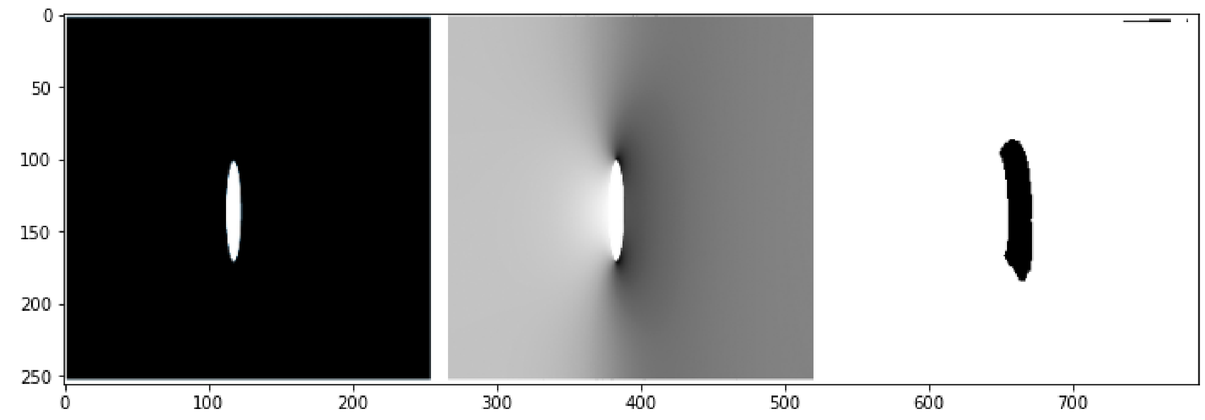
Here below, you can see the predicted images of Pressure UNet and Velocity UNet. The main drawback of the greyscale images is it can only detect the region from the contour, which is in black color. So, in predicted images we can see only the black region, and it was not accurate as if it would be with the colored contour and input.



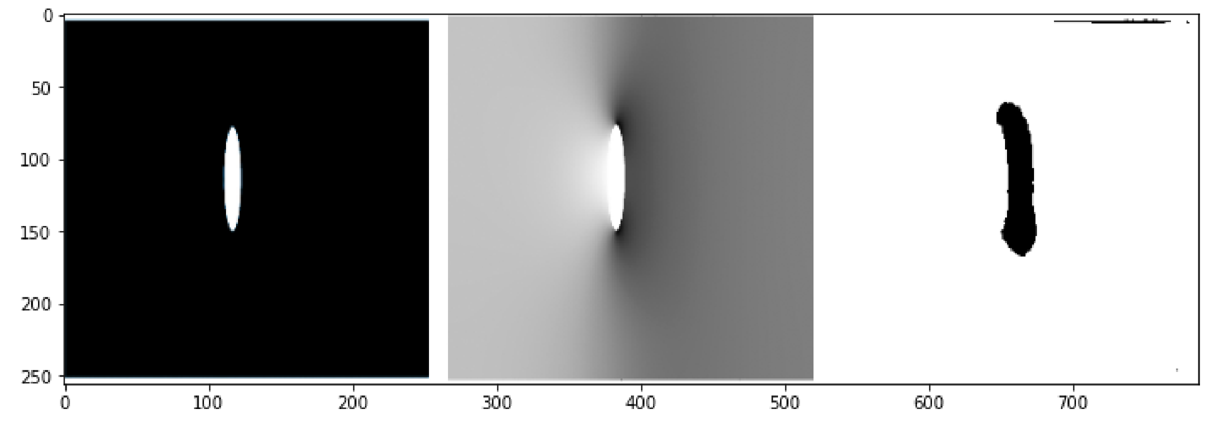
0
50
100
150
200
250
0
100
200
300
400
500
600
700



0
50
100
150
200
250
0
100
200
300
400
500
600
700



0
50
100
150
200
250
0
100
200
300
400
500
600
700



0
50
100
150
200
250
0
100
200
300
400
500
600
700

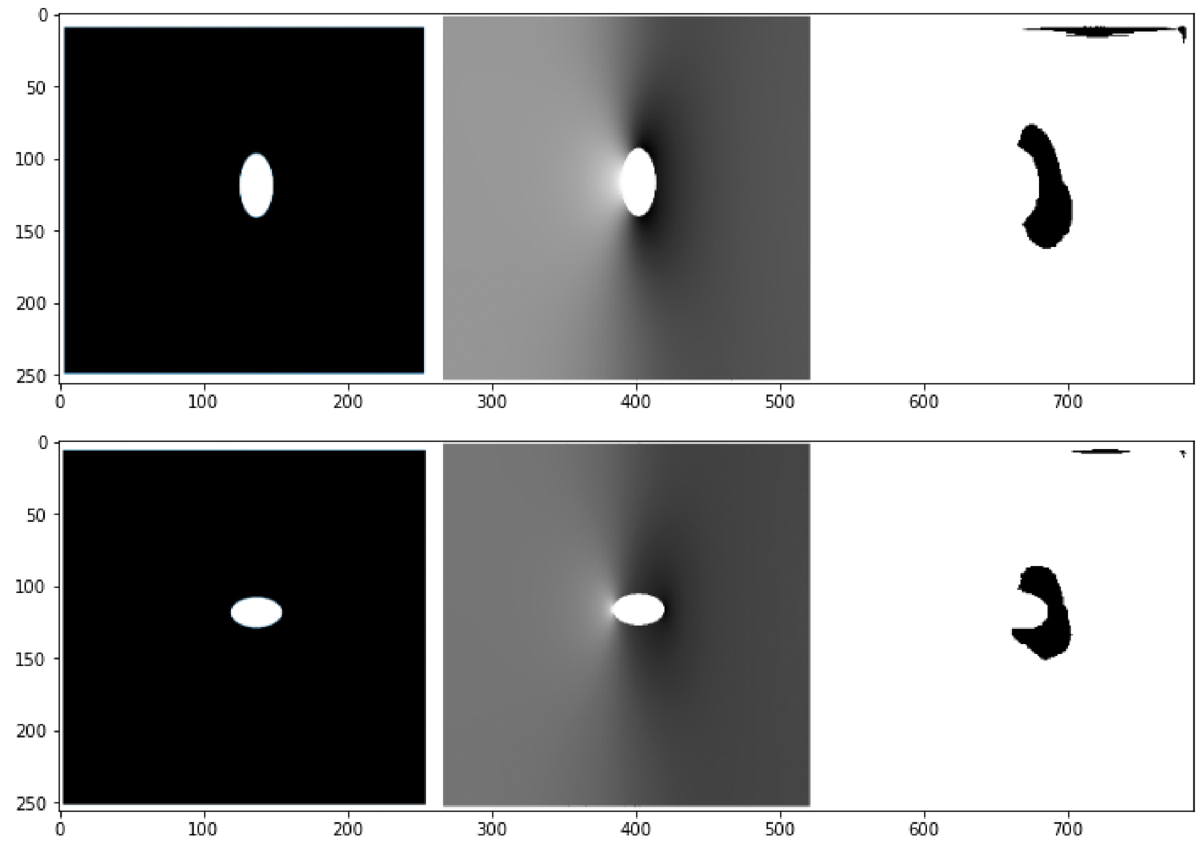
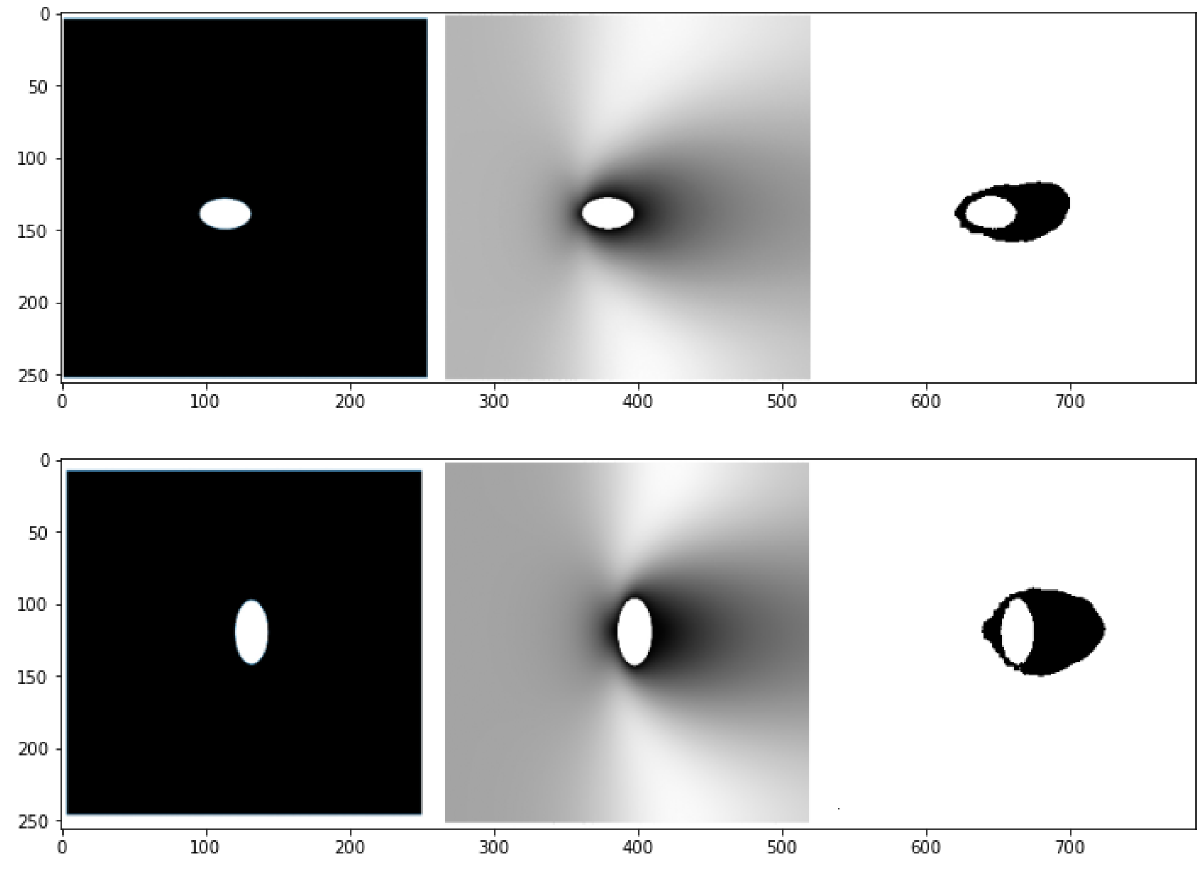


Fig 5.3: Pressure UNet Predicted Images



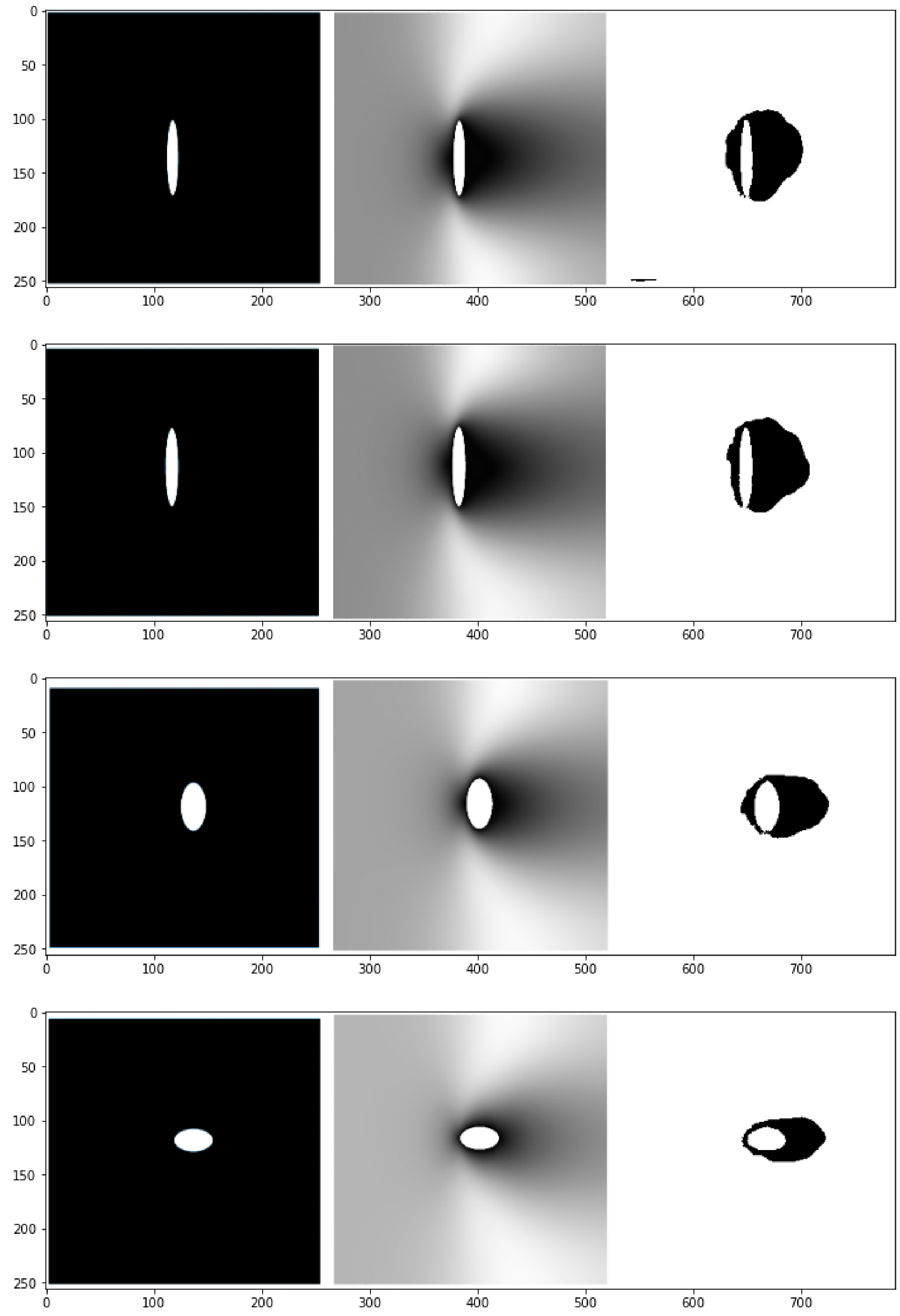


Fig 5.4: Velocity UNet Predicted Images

5.1.3 Computational Time:

Another main aim of our mini project is minimizing the computation time required to solve the CFD models by building a surrogate model using CNN. Basically, in CNNs, the number of cores plays a significant role in computation time. So, for testing our CNN model with CPU based, we have used an Intel Core i7-10750H processor, with a processor base frequency - 2.60 GHz Max Turbo Frequency - 5.00 GHz, Number of cores – 6, Number of threads – 12 and cache - 12MB. The results are demonstrated below:

Table 5.1: CPU Computational Time

UNet Model	Execution time
Pressure UNet	0:10:09.863590
Velocity UNet	0:10:30.907419

Here, you can see it took almost 11 minutes to run the model and compile it. We know that CNNs are much faster with GPUs than the CPUs as there many CUDA cores present in GPUs, which makes the CNN work quicker and accurate. So, for testing our CNN model with GPU based, we have used Nvidia GeForce RTX 2060 Super, which has 1650Mhz clock speed, 2175 CUDA cores, and 8GB system memory of GPU.

Table 5.2: GPU Computational Time

UNet Model	Execution time
Pressure UNet	0:00:44.765289
Velocity UNet	0:00:44.834898

So, you can see it took almost 45 seconds to run the model and compile. So, we were able to reduce the computational time with some information loss and less accurate results. As we have a dataset with 66 images, the accuracy was somewhat low, but if we had a bigger dataset, our model would have accuracy.

CHAPTER-6
CONCLUSION

Therefore, we successfully able to build a surrogate model for 2-D steady state laminar flow. As our dataset is minimal, the prediction was very moderate. So, we want to increase our data set size and increase qualitative predictions in future work.

Due to technical problems, despite preparing the dataset in colored images, we had to change the dataset into grayscale and output the predicted images in grayscale. Next time we want to do this with all the colored images to have better accuracy. In this project, we applied deep Convolutional Neural Networks to Computation Fluid Dynamics modeling.

Our primary motivation is to provide a lightweight (fast and less-accurate) surrogate model for exterior flow. Even though for many domains, such as architectural design, low Reynolds number flows are usually sufficient, we intend to explore higher Reynolds number flows in the future to extend the approach to other areas. It would also be worthwhile investigating whether we could use the results from our approximation models as an initial setup to warm start high-accuracy CFD simulations.

Since the predictions are relatively close representations of the final, fully converged results, the number of iterations required to converge to a steady state could be significantly reduced. Therefore high-accuracy traditional CFD methods could be made to converge much more quickly.

REFERENCES:

Journals Papers:

[1] M. Ahmed and N. Qin. Surrogate-Based Aerodynamic Design Optimization: Use of Surrogates in Aerodynamic Design Optimization. Aerospace Sciences and Aviation Technology, ASAT-13, 2009.

[2] V. Balabanov, A. Giunta, O. Golovidov, B. Grossman, H. Mason, T. Watson, and T. Haftkalf. Reasonable design space approach to response surface approximation. Journal of Aircraft, 1999.

[3] M. Batill, A. Stelmack, and S. Sellar. Framework for multi-disciplinary design based on response-surface approximations. Journal of Aircraft, 1999.

[4] Y. Bengio. Learning deep architectures for AI, Foundations and trends in Machine Learning. 2009.

[5] L. Chittka, P. Skorupski, and E. Raine. Speed-accuracy tradeoffs in animal decision making. Trends in Ecology and Evolution, 2009.

[6] D. Eigen, C. Puhrsch, and R. Fergus. Depth map prediction from a single image using a multi-scale deep network. In Advances in neural information processing systems, 2014.

[7] H. Fang, M. Rais-Rohani, Z. Liu, and M. Horstemeyer. A comparative study of metamodeling methods for multi-objective crashworthiness optimization. Computers & Structures, 2005.

[8] A. Giunta. Aircraft Multidisciplinary Design Optimization Using Design of Experiments Theory and Response Surface Modeling Methods. 1997.

[9] S. Gupta, R. Girshick, P. Arbelaez, and J. Malik. Learning rich features from RGB-D images for object detection and segmentation. In Computer Vision, ECCV. 2014.

[10] D. Hartmann, M. Meinke, and W. Schroder. Differential equation based constrained reinitialization for level set methods. Journal of Computational Physics, 2008.

[11] V. Heuveline and J. Latt. The OpenLB project: an open source and object-oriented implementation of lattice Boltzmann methods. International Journal of Modern Physics, 2007.

[12] S. Jeong, M. Murayama, and K. Yamamoto. Efficient optimization design method using kriging model. Journal of Aircraft, 2005.

[13] S. J. Leary, A. Bhaskar, and A. J. Keane. Global approximation and optimization using adjoint computational fluid dynamics codes. AIAA journal, 2004.

[14] D. Tran, L. Bourdev, R. Fergus, L. Torresani, and M. Paluri. Deep end2end voxel2voxel prediction. arXiv preprint arXiv:1511.06681, 2015.

[15] S. Wilkinson, G. Bradbury, and H. S. Reduced-Order Urban Wind Interference, Simulation: Transactions of the Society for Modeling and Simulation International. 2015.

[16] G. Russo and P. Smereka. A remark on computing distance functions. Journal of Computational Physics, 2000.

[17] M. Mawson, G. Leaver, and A. Revell. Real-time flow computations using an image-based depth sensor and GPU acceleration. 2013.

[18] G. R. McNamara and G. Zanetti. Use of the boltzmann equation to simulate lattice-gas automata. Physical Review Letters, 1988.

[19] S. Patankar. Numerical heat transfer and fluid flow. CRC Press, 1980.

[20] R. Socher, B. Huval, B. Bath, C. D. Manning, and A. Y. Ng. Convolutional-recursive deep learning for 3d object classification. In Advances in Neural Information Processing Systems, 2012.

Books:

[1] J. Anderson. Computational Fluid Dynamics. 1995.

[2] G. Forsythe and W. Wasow. Finite-Difference Methods for Partial Differential Equations. 1960.

[3] J. Schmidhuber. Deep learning in neural networks: An overview. Neural Networks, 2015.

Conference Proceedings:

[1] D. Daberkow and D. N. New Approaches to Conceptual and Preliminary Aircraft Design: A Comparative Assessment of a Neural Network Formulation and a Response Surface Methodology. World Aviation Conference, 1998.

[2] A.Dosovitskiy, P. Fischer, E. Ilg, P. Hausser, C. Hazrba, V. Golkov, P.v.d. Smagt, D. Cremers, and T. Brox. Flownet: Learning optical flow with convolutional networks. In IEEE International Conference on Computer Vision (ICCV), 2015.

[3] Y. Jia, E. Shelhamer, J. Donahue, S. Karayev, J. Long, R. Girshick, and T. Darrell. Cae: Convolutional architecture for fast feature embedding. In Proceedings of the ACM International Conference on Multimedia. 2014.

[4] Y. LeCun, L. Bottou, Y. Bengio, and P. Haner. Gradient-based learning applied to document recognition. Proceedings of the IEEE, 1998.