Development of a Surrogate Model for FEM Error Prediction using Deep Learning

Siddharth Jain

Thesis submitted to the Faculty of the

Virginia Polytechnic Institute and State University

in partial fulfillment of the requirements for the degree of

Master of Science

in

Aerospace Engineering

Rakesh K. Kapania, Chair

Daniel C. Hammerand

Heng Xiao

May 9, 2022

Blacksburg, Virginia

Keywords: Deep Learning, Convolution Neural Networks, Finite Element Analysis,

Surrogate Model, Adaptive Remeshing

Development of a Surrogate Model for FEM Error Prediction using Deep Learning

Siddharth Jain

(ABSTRACT)

This research is a proof-of-concept study to develop a surrogate model, using deep learning (DL), to predict solution error for a given model with a given mesh. For this research, we have taken the von Mises stress contours and have predicted two different types of error indicators contours, namely (i) von Mises error indicator (MISESERI), and (ii) energy density error indicator (ENDENERI). Error indicators are designed to identify the solution domain areas where the gradient has not been properly captured. It uses the spatial gradient distribution of the existing solution for a given mesh to estimate the error. Due to poor meshing and nature of the finite element method, these error indicators are leveraged to study and reduce errors in the finite element solution using an adaptive remeshing scheme. Adaptive re-meshing is an iterative and computationally expensive process to reduce the error computed during the post-processing step. To overcome this limitation we propose an approach to replace it using data-driven techniques. We have introduced an image processing-based surrogate model designed to solve an image-to-image regression problem using convolutional neural networks (CNN) that takes a 256×256 colored image of von Mises stress contour and outputs the required error indicator. To train this model with good generalization performance, we have developed four different geometries for each of the three case studies: (i) quarter plate with a hole, (b) simply supported plate with multiple holes, and (c) simply supported stiffened plate. The entire research is implemented in a three phase approach, phase I involves the design and development of a CNN to perform training on stress contour images with their

corresponding von Mises stress values volume-averaged over the entire domain. Phase II involves developing a surrogate model to perform image-to-image regression and the final phase III involves extending the capabilities of phase II and making the surrogate model more generalized and robust. The final surrogate model used to train the global dataset of 12,000 images consists of three auto encoders, one encoder-decoder assembly, and two multi-output regression neural networks. With the error less than 1% for the training data set, the neural network shows good memorization and generalization performance. Our final surrogate model takes 15.5 hours to train and less than a minute to predict the error indicators on testing datasets. Thus, this present study can be considered a good first step toward developing an adaptive remeshing scheme using deep neural networks.

Development of a Surrogate Model for FEM Error Prediction using Deep Learning

Siddharth Jain

(GENERAL AUDIENCE ABSTRACT)

This research is a proof-of-concept study to develop an image processing-based neural network (NN) model to solve an image-to-image regression problem. In finite element analysis (FEA), due to poor meshing and nature of the finite element method, these error indicators are used to study and reduce errors. For this research, we have predicted two different types of error indicator contours by using stress images as inputs to the NN model. In popular FEA packages, adaptive remeshing scheme is used to optimize mesh quality by iteratively computing error indicators making the process computationally expensive. To overcome this limitation we propose an approach to replace it using convolutional neural networks (CNN). Such neural networks are particularly used for image based data. To train our CNN model with good generalization performance we have developed four different geometries with varying load cases. The entire research is implemented in a three phase approach, phase I involves the design and development of a CNN model to perform initial level training on small image size. Phase II involves developing an assembled neural network to perform image-to-image regression and the final phase III involves extending the capabilities of phase II for more generalized and robust results. With the error of less than 1% in the neural network training shows good memorization and generalization performance. Our final surrogate model takes 15.5 hours to train and less than a minute to predict the error indicators on testing datasets. Thus, this present study can be considered a good first step toward developing an adaptive remeshing scheme using deep neural networks.

Dedication

I want to dedicate my thesis to my parents who are always proud of me and to my brother who always inspires me to do something different and become a better version of myself.

Acknowledgments

I would like to thank my advisor Dr. Rakesh Kapania and my committee member Dr. Daniel C. Hammerand for their invaluable guidance in the course of my graduate program. The ideas and strategies provided by both of them helped me a lot in finding solutions to challenges during the entire course of research. I admire the discussions we had in the process of this research and the huge learning experience while working on this thesis. These discussions helped me to become a better researcher. I have learned new skills and techniques that will be of great help throughout my career. I especially appreciate the efforts taken by Dr. Hammerand for taking time from his busy schedule to be my committee member and guide my research work through weekly and bi-weekly meetings. I really appreciate Dr. Heng Xiao for serving on my graduate committee and spending time to go through my thesis and providing his valuable feedback. I would also like to thank the Virginia Tech Advanced Research Computing (ARC) for providing the necessary computational resources and technical support that have contributed to the results reported in this research. Finally, I would like to thank the Department of Aerospace Engineering for providing me an opportunity to be a part of their master's program, my family for believing in me and supporting me throughout my graduate studies and my colleagues for being there with me.

Contents

List of Figures

List of Tables

List of Abbreviations

AE	Auto Encoder
ARC	Advanced Research Computing
CNN	Convolutional Neural Networks
DCNN	Deep Convolutional Neural Networks
DL	Deep Learning
ENDENERI	Energy Density Error Indicator
FC	Fully Connected
FEA	Finite Element Analysis
FEM	Finite Element Method
GPU	Graphics Processing Units
HPC	High Performance Computers
MISESERI	von Mises Error Indicator
ML	Machine Learning
MLP	Multi Layer Perceptron
NN	Neural Networks
PDEs	Partial Differential Equations

ReLU	Rectified Linear Units
RNN	Recurrent Neural Networks
TPC	Tensor Processing Units

Chapter 1

Introduction

As more and more industries and technology startups are flourishing, the demand for computational analysis has shown a rapid increase. Specifically, in mechanical and aerospace engineering, computational stress analysis is fundamentally critical to the design and fabrication of structural components and products. While designing automobiles, aircraft or marine structures, high-rise buildings, bridges, or highways, finite element methods (FEM) have fundamentally revolutionized the way we do scientific modeling and engineering analysis. Finite element analysis (FEA) simulations made the design iterations independent of machine shop and manufacturing schedules making each new design tested virtually in hours, instead of waiting days or weeks for a hard copy to be tested. Commercial availability of FEM packages has resulted in a wide range of applications not limited to the aforementioned examples but also in biological processes for medical diagnosis, dentistry, surgery planning, electromagnetics, semi-conductor circuit, additive manufacturing, chip designing. FEM has become the computational workhorse and is being used in every conceivable problem that can be described by partial differential equation (PDEs).

Being an approximate method, there can be multiple sources of error while running an FEM simulation. One common and a primary source of error is associated with inappropriate domain discretization. Since the mesh quality strongly affects the solution accuracy, it is crucial to know the correct meshing parameters for the problem under study to generate a good quality mesh. The information about optimal meshing parameters is seldom known

to us and generally, requires several analysis iterations to acquire. Mesh designing and generation is one of the most critical aspects of engineering simulation especially when it comes to any structural design and analysis problem. Solution accuracy for such analysis using numerical methods like FEM requires a large amount of memory and longer simulation run-time. The overall mesh quality and distribution of element sizes dominate the solution computational cost and convergence for both linear and nonlinear problems. Too many elements may result in long solver run-times and too few leads to inaccurate results [9]. *A priori* mesh or initial mesh, even when generated with the best practice guidelines, such initial meshes cannot be guaranteed to result in an accurate solution [17]. Continuous improvements in pre- and post-processing formulations when combined with graphical user interfaces (GUIs) has led to more interactive problem implementation and visualization. From the early stage ($1941 - 1965$) and the golden age ($1966 - 1991$) of finite elements, there have been many improvements in the algorithm to improve accuracy. One major breakthrough was the implementation of the Zienkiewicz-Zhu error estimator (1992). This *a posteriori* error estimator enables the quality control of a finite element solution with an optimal use of computational resources by specifying prime areas for mesh adaptivity [18].

According to the universal approximation theorem, a neural network (NN) has a kind of universality that means it can be designed and trained to approximate any given continuous function to the desired accuracy. In [23] Selmic and Lewis (2002) discussed a novel NN architecture that provides an approximate solution to piecewise continuous functions as well. Such scientific discoveries and engineering innovations enticed FE researchers to study various forms of machine learning (ML) and its subset deep learning (DL) methods for solving FEM problems. This has become a state-of-the-art technology in this modern era of FEM. There can be two different approaches to train a neural network, (i) Offline Learning: also known as batch learning where a NN model is built using the entire batch of data available

all at once, and (ii) Online Learning: where the training is being done in real-time as the data comes in. Online learning or evaluations makes the NN more adaptable because it makes no assumption about the distribution of the data. With any change in user behavior, NN also adapts on-the-fly to keep pace with the trends in real-time. Such training is also very data efficient as there is no need to store data, because once data has been consumed it is no longer required.

In the scientific community, ML-based surrogate models and data-driven techniques have received considerable attention because of fast online evaluations allowing researchers to obtain results much faster. These techniques allow the accumulation of large datasets and fast searches for correlations and advanced computations. This produces many data-driven methods where the obtained knowledge is processed to feed the simulation models to deduce the results and improve prognosis. The availability of a large amount of data has encouraged the application of machine learning that not only identifies hidden and unexpected patterns, but also can be applied to learn and understand the process that generates the data. The patterns learned are used to analyze unknown data such that it can be grouped or mapped to the unknown groups [5].

Many surrogate models have been developed that combines both offline and online learning, called 'coupled learning' or two-stage learning to reduce the FE computational cost. In such data-driven methods during the offline stage, a database is generated by the FEM algorithms and the final solution is evaluated during the online stage. In 2016, Z. Liu *et al.* [19] developed a two-scale data-driven approach to predict the behavior of general heterogeneous materials under irreversible processes such as inelastic deformation. Their research involved two major contributions, (i) the k-means clustering algorithm to group material subdomains with similar mechanical behavior; (ii) their new analysis method: "self-consistent clustering analysis." Their methodology was found to be accurate, achieved good convergence rate

under refinement, was computationally efficient, and involved a minimal amount of effort for both the offline and online stages.

In computational solid mechanics, there have been early attempts to use neural regression for FEA. In [22], Nie *et al.* developed two end-to-end DL models, one with a single input channel and the other with multiple input channels. They aim to predict the stress fields in a cantilevered beam structure with a linear isotropic elastic material. They designed and implemented CNN on a dataset of 28 different geometries each varying in four different categories, geometry contour, hole shape, size, and location. To read and extract information from the given geometry and then generate a corresponding stress field image, they used a baseline architecture of the encoder-decoder neural network (NN) structure. Their particular approach takes the following input: structure geometry, external loads, and displacement boundary conditions. With such given information, the CNN model can output the predicted stress field.

Similarly, in another study [6], Guodong and Krzysztof used machine learning to build a surrogate model to solve a regression problem. They discussed how adjoint-based methods provide an approach to quantify the output error and to guide the mesh adaptation. They presented an application of machine learning techniques to overcome the liability of obtaining an adaptive solution that imposes both implementation and cost challenges. Their model is trained to predict the output error with a low-fidelity solution as input with the aim to generalize the error modeling knowledge from the available simulation data. Their methodology is developed on a rectangular computational domain without any geometry and solving an advection-diffusion problem. Their neural network is a supervised machine learning model trained using both the adaptive error indicator field and the total output error to capture both the local and global features related to numerical error. In both of these studies, they have leveraged a special type of architecture called encoder-decoder type CNN.

Our research directly works on a standard problem in solid mechanics and the machine learning model is trained with high quality colored images and a varying dataset. We discuss how data-driven techniques provide capabilities to improve mesh quality and improve solution accuracy for FEA. The motivation of this research is to develop a building block to find an alternate approach to perform adaptive re-meshing without any iterative steps and re-meshing rules. The next section will provide a high-level overview of adaptive re-meshing, available in Abaqus [25]; deep learning, and neural networks. The methodology section discusses how a deep learning-based surrogate model is developed as a proof of concept to showcase the abilities of machine learning to predict errors in each element for a structural simulation. It overcomes the challenge of excessive run times and iterative simulations to perform adaptive re-meshing. Results from the developed tool can be used as a recommendation to obtain information such as the location of the maximum error in *a priori* mesh, which will help the user to update the mesh accordingly. A unique aspect of *a priori* analysis is that the mesh quality metrics depend solely on the mesh geometry and are independent of the equations being solved and of the solution itself. [20]

1.1 Motivation

In computational analysis, adaptive remeshing is a powerful tool that can help in optimizing the mesh generation process. It allows the user to perform mesh refinement where it is of utmost importance or mesh coarsening where the spatial solution gradient is low. In [2] Ahmed and Singh (2007) performed a parametric study for sheet forming operation using FEA and implementing adaptive meshing techniques. They studied the influence of different parameters on the performance of adaptivity procedures at different stages during the analysis. In any commercial FEA package, adaptive remeshing is computed as a post-

processing step that makes it computationally expensive. Therefore they also studied and discussed the CPU time by varying adaptivity parameters (i) field variable recovery, (ii) refinement techniques, and (iii) accuracy limit. During the CPU time comparison analysis, it was found that the choice of post-processing of basic field variable leads to faster convergence, and the patch size of the recovery domain, enhances the recovery process of improved value. Their study shows how the entire FEA simulation took total of 22 hours under the first adaptivity parameter while the same simulation under refinement techniques took 7 hours to complete. Considering this significant difference in CPU time, we can infer that the total computation cost varies a lot with model size, type of simulation, type of error estimators, and adaptivity procedures. This research address this particular issue and presents a proof-of-concept approach to building an improved adaptive remeshing scheme. Our current version of the surrogate model can be considered as the building block based on artificial intelligence. Presumably given enough training, the developed deep neural network (DNN) could handle a broad range of structures under a broad range of loadings. A pre trained DNN could be used for existing and previous analyses assessment resulting in total cost savings.

1.2 Mesh Refinement and Adaptive Re-Meshing

In [14], Knupp defines mesh quality as "the characteristics of a mesh that permit a particular numerical PDE simulation to be efficiently performed, with fidelity to the underlying problem and with the accuracy required for the problem." Knupp also discussed how the mesh geometry is dependent on the type of calculation required to be solved. In FEA, the primary objective of a dense mesh is to provide accuracy to the solution, therefore it is appropriate to consider error analyses to monitor mesh quality. We know that increasing the mesh density over a region in the domain, called h-refinement, is one of the methods

to improve solution quality. Similarly, *p*-refinement is another technique where we keep on using the same element size but instead, we increase the order of the polynomial used in the mesh definition. Adaptive re-meshing is an iterative approach available in popular FE packages that perform such refinements automatically and provides the user with a better quality mesh that guarantees the accurate solution up to the desired level. This cyclic process mainly consists of four following steps (i) running analysis on a given mesh, (ii) computing the error of the desired solution (iii) comparing the errors against the desired error limits, and (iv) if required, refine or coarsen the mesh in areas of higher or lower error locations depending on the solution gradient distribution over a given mesh.

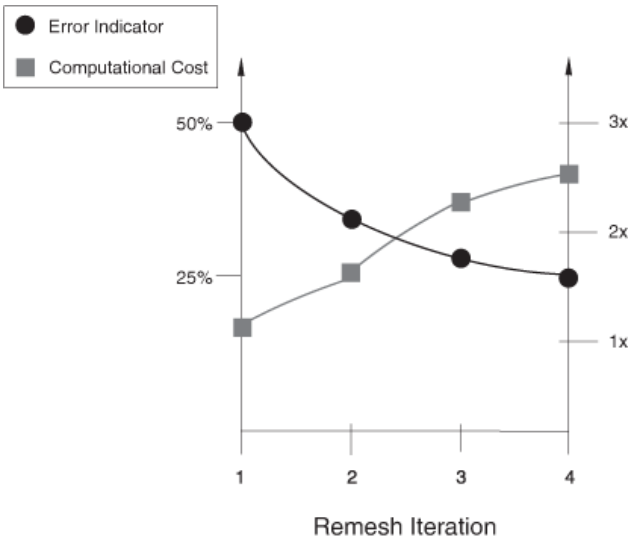


Figure 1.1: Convergence profile for adaptive meshing procedure [25].

To perform adaptive re-meshing a user needs to define remeshing rules such as specific steps for refinement, the total number of iterations, the sizing method, or any other constraints on the remeshing calculations. It works on calculating error values in the base solution at every step of the analysis, until it satisfies the given constraints, making the whole process computationally expensive. These error values or error indicators are simply a measure to indicate the error using the existing solution (like the jumps in stress bands) for a given mesh.

Abaqus/Standard provides these error indicator output variable to aid in understanding the spatial distribution of the discretization error. They are designed to identify the solution domain areas where the gradient has not been properly captured. By predicting the regions of smoothed or unsmoothed areas of spatial solution gradients, it determine where to coarsen or refine a mesh. At each iteration step, the solver tends to find the location where to refine or coarsen a mesh and then generates a new mesh in the specified region based on computed element sizes. From the adaptive meshing procedure shown in 1.2 and convergence profile shown in Figure 1.1, the solution error indicator decreases monotonically to the desired 25% error indicator. Accompanying this, the error indicator decreases with a moderate increase in computational cost. A recent trend in the development of the finite element technique is the use of adaptive procedures based upon error estimators. There are 3 main types of error estimators for adaptive procedures, (i) the residual type introduced by Babuska and Rheinholdt (1978), (ii) the interpolation type propounded by Erikson and Johnson (1988), and (iii) the post-processing type proposed by Zienkiewicz-Zhu (1987). The post-processing type of error estimators relies upon a "recovery" of higher order finite element solution. The error can be evaluated in any appropriate norms. Since the finite element solution minimizes the error in the energy norm, the magnitude of the error in the energy norm is a good measure of the overall quality of the solution, but can miss refining cases where there are steep solution gradients over small solution regions.

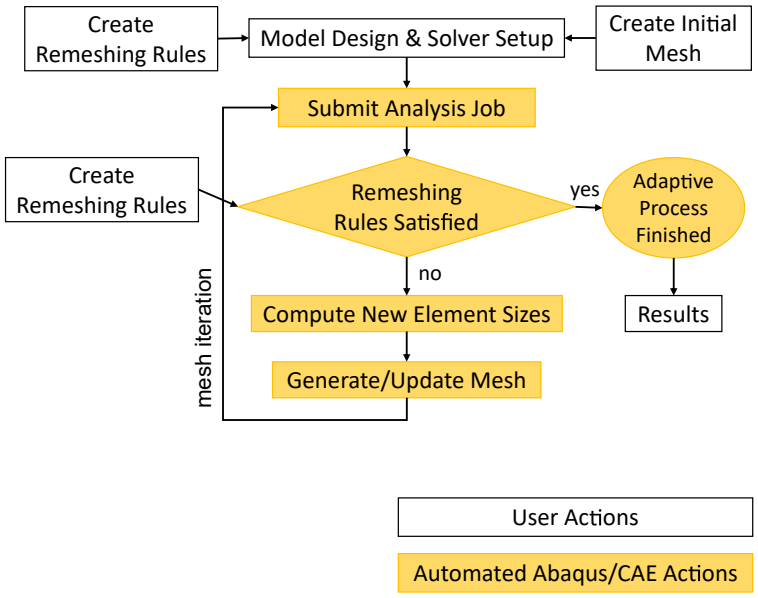


Figure 1.2: Outline of Adaptive Remeshing procedure in Abaqus/Standard.

1.3 Deep Learning

Typically if machine learning is being used somewhere, it is either solving one or more of these problems: (i) recommendation problem (e.g., which movie to watch next based on a past watchlist); (ii) classification problem (e.g., identifying objects or images); (iii) regression problem (e.g., predicting house prices based on certain parameters and criteria); (iv) clustering problem (e.g., filtering fake news); (v) ranking problem (e.g., credit risk ranking; and (vi) anomaly detection (e.g., detecting credit card fraud). Figure 1.3 is a multi-layer perceptron (MLP) model and provides a high-level understanding and basic definitions to get a rough idea of what machine learning is. There are three types of learning approaches to solve ML problems, (i) supervised learning where the output is fed to the network along with the input data to perform training, (ii) unsupervised learning where no explicit information is given to the DL model to perform training, and (iii) reinforcement learning where a reward based algorithm is designed for an "AI Agent" to make predictions. Here, an AI

agent is termed referred to as a collection of two components, (i) a policy which is a function approximator with tunable parameters such as a DNN, and (ii) the learning algorithm, that continuously updates the policy based on the actions, observations, and rewards.

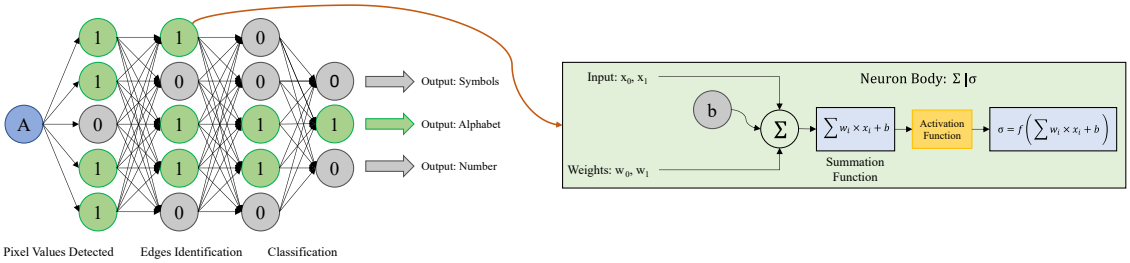


Figure 1.3: Deep Learning in its simplest form - with working of neurons in a multi perceptron model.

Mostly neural networks are broadly classified into two types: (i) feed forward network, and (ii) feed backward network. These are two different types of architectures used to establish a connection between individual nodes. In a feed forward network, the signal travels only in the forward direction, one way only. This type of network is considered a static network which means each input is associated with only one particular output whereas, the feed backward networks are dynamic which means, one input produces a series of outputs. The state of the feed backward network changes for many cycles only during the training portion until it reaches an equilibrium point as shown in Figure 1.4. For a given problem, with a set of training samples (X, Y); there can be an infinite number of different size networks that can learn to map the input patterns, X with output patterns, Y. To transfer the signal from one node to the next node, a non-linear mapping function F is established during the training phase. Here the network learns to correctly associate $Y = F(X; \theta)$ with input patterns X to output patterns Y by optimizing the parameter θ using a technique called back-propagation.

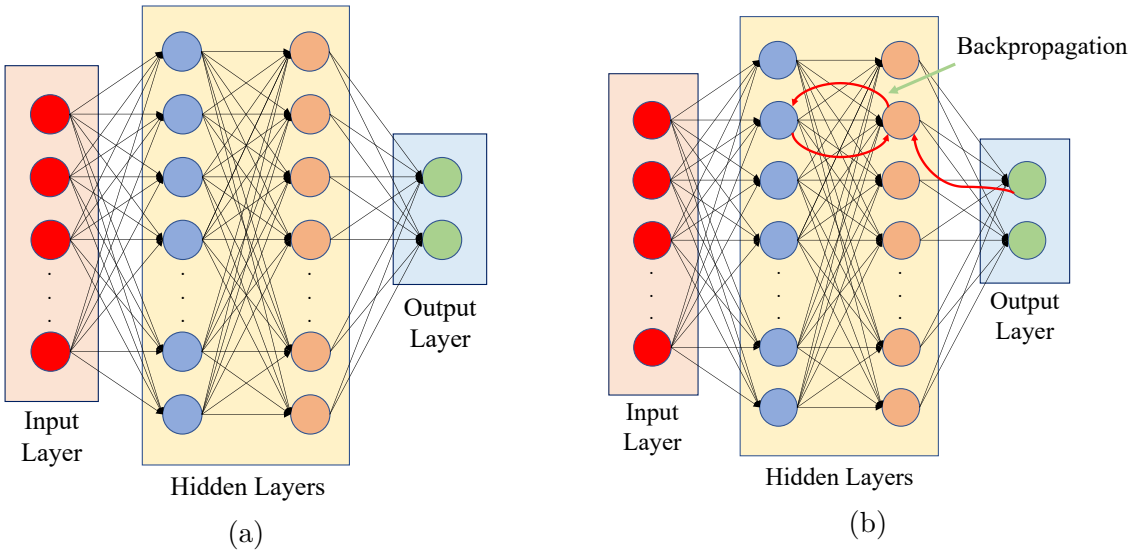


Figure 1.4: Data propagation of typical neural network: (a) feed forward network; (b) feed backward network.

Behind the huge success of ML in solving research problems, network size plays an important role. In [3] Bebis and Georgiopoulos (1994) discuss the importance of network size in feed forward networks. One of the key takeaways is that, the network size affects the generalization capabilities, network complexity, and learning time. While designing a network architecture the primary objective is to achieve both memorization and generalization capabilities. Research done by Google Inc. [7], Cheng *et al.*, discuss why these two are the important features of neural networks and how wider and deeper networks influence these features. Generalization can be loosely defined as the ability to identify the rules and making the system to make predictions and produce accurate results on patterns on the unknown data. Memorization, on the other hand, is a learning of the frequent co-occurrence of features or items and using the correlation available in a historical data. It is an obvious task in learning and can be performed by storing the input samples explicitly, or by identifying the concept behind the input data, and memorizing their general rules.

With a wider network architecture, it is possible to train a neural network with every possible

input value making them good at memorization but they perform poorly at generalization. On the other hand, a deep neural network (DNN) that consists of multiple hidden layers can effectively captures the natural hierarchy that is present everywhere in the given training dataset. One of the major advantages we have in DNN is that they can learn features at various levels of abstraction. That means, with every hidden layer, a DNN captures the low level features in the first layer and tries to improve the learned features as the input signal moves forward. This makes the network good at generalization because multiple layers learn all the intermediate features between the raw data and the high-level classification. Both wide and deep networks have their own advantages but they also come with their challenges during the training phase. By increasing the network size we increase the total number of parameters that the networks want to learn, hence increasing the chances of overfitting.

In [7], Cheng *et al.* proposed a network architecture that is both wide and deep and works great on a recommendation system resulting in good generalization and memorization capabilities. Our proposed method focuses on building a feed forward deep neural network using convolutional layers to obtain two different error indicators for a given von Mises stress contour. Such networks are a special type of trained DNN called convolutional neural network (CNN). Following chapters discusses the working of CNNs and how the amount and quality of data impacts the training performance.

Chapter 2

Convolutional Neural Networks

Representation learning, or commonly referred to as feature learning, is defined as learning representations from the raw data that makes it easier to perform feature detection when building regression or classification models [4]. With the idea of successive layers to extract information, CNNs have shown great success in image recognition and processing. They have various applications such as face recognition, object detection, traffic identification, and powering computer vision in autonomous systems. Considering the bigger picture, a CNN comprises of four different layers: (i) Convolutional Layer, (ii) Activation Layer or non-linearity layer, (iii) Pooling Layer or sub-sampling operation layer, and (iv) Fully Connected Layer or Ordinary neuron layer. Some of these layers do have some additional parameters, called 'Hyperparameters' to fine-tune the performance of the respective layer. Representation learning algorithms have also been applied to speech recognition, signal processing, or in even more unique cases such as to music for retrieving music information or audio tagging. However, the current work using CNN will mostly be related to object recognition with a focus on image data.

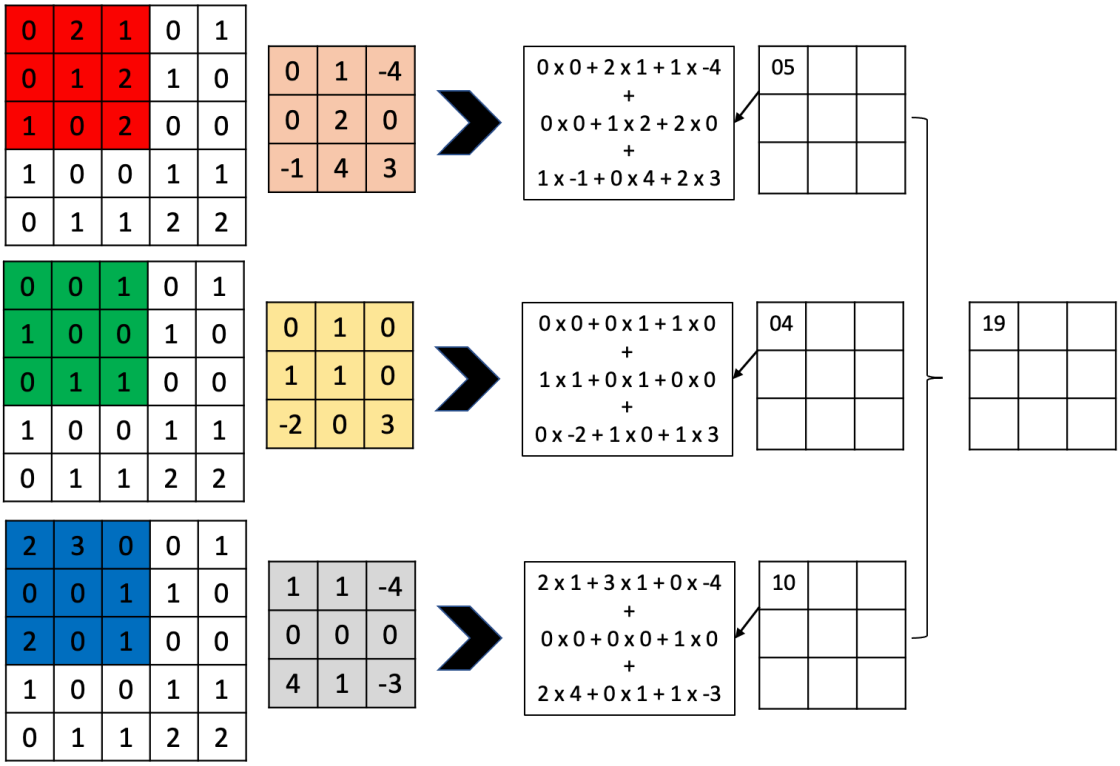


Figure 2.1: CNN operation on a colored image.

Images are a unique type of data representation that can provide information that is difficult to measure from or not available in the text data. To perform machine learning, CNNs are well-known architectures designed especially for such types of data. Over many years, a significant amount of research has been done on implementing these neural networks for image processing, image classification, and image segmentation. With advancements in the network structure, these neural networks are becoming more accurate while using a lesser amount of training time.

2.1 Previously developed DCNNs

This section discusses most of the popular DCNNs built to compete in ImageNet Large Scale Visual Recognition Competition (ILSVRC) competition. These networks are known to make mostly correct assumptions to predict the patterns and successfully classify them.

2.1.1 AlexNet

For image recognition, in 2012 a landmark paper [16] from Alex Krizhevsky introduced the first ever Deep Convolutional Neural Network, 'AlexNet' while he was working in Geoff Hinton's lab. This DCNN won the ImageNet Large Scale Visual Recognition Competition (ILSVRC). This dense network consisted of 5 convolutional layers, along with max pooling layers, three fully-connected layers, and lastly a layer of 1000-way softmax to perform classification. This network comprised of a total of 650,000 neurons and had 60 million parameters. It was used to classify the 1.2 million high resolution images in the ImageNet LSVRC-2010 contest into 1000 different classes. The reason behind the significant acceleration in the training phase that led AlexNet to its success was hidden in the non-linear activation function is used, "ReLU - Rectified Linear Unit". Looking at the network architecture, in Figure 2.2 we can notice that the network architecture is divided into two parts. It was designed in such a way that one half executes on one GPU and the other half executes on the second GPU. This parallel computing, leveraging multiple GPUs, increased the training speed. Also, Krizhevsky *et al.*, with the use of the data augmentation technique and implementation of the dropout layer successfully avoided the overfitting.

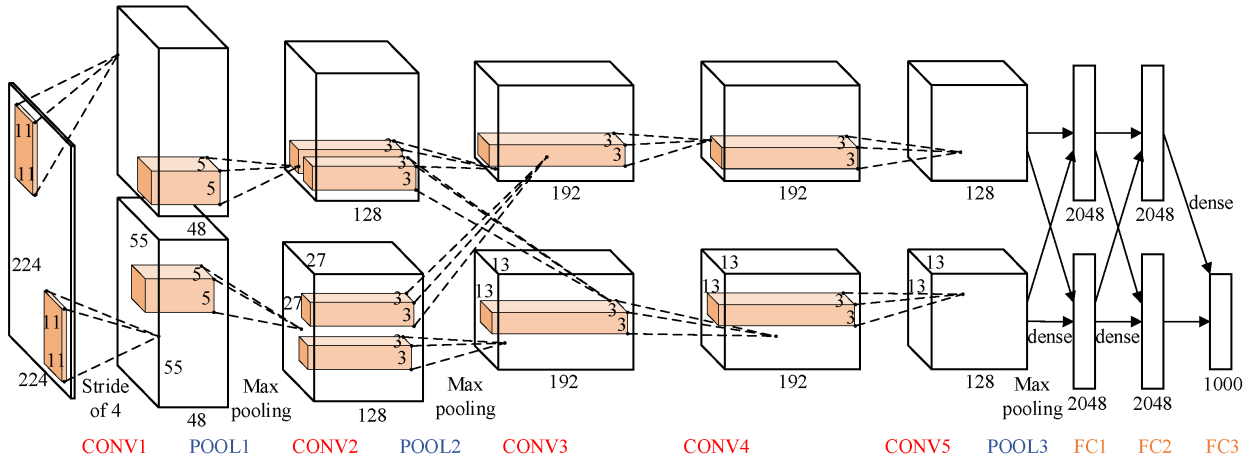


Figure 2.2: Network architecture of AlexNet [31].

2.1.2 VGG-16

Another popular DCNN that was introduced at the ILSVR competition in 2014 is VGG-16 and it is still considered as one of the excellent vision model architecture. Simonyan and Zisserman [24], from the Oxford Vision Geometry Group (VGG) achieved top results for image classification and localization with their VGG model. Their model architecture was able to achieve 92.7% top-5 test accuracy on the ImageNet dataset. It is also important to note that in two more years, the ImageNet database was also increased from 1.2 million to 14 million images. This means that VGG not only provides better classification accuracy as compared with AlexNet but also good generalization and memorization capability. Simonyan and Zisserman with their VGG-16 architecture were able to achieve an error score of 7.3% on a massive dataset of 14 million images. They were able to surpass AlexNet by using a smaller convolutional kernel of size 3×3 instead of the larger kernel size of 11×11 and 5×5 present in AlexNet. Whereas AlexNet consists of a total of 62 million parameters, VGG-16 consists of 138 million parameters because of the aforementioned changes and a deeper network of 16 layers.

2.1.3 InceptionNets

Another breakthrough was shown by another DCNN developed by Szegedy *et al.* [28] which won the 2014 ILSVR competition with their GoogleNet or InceptionNet. With the main purpose of dimension reduction, this network used 1×1 convolutions resulting in reducing the output channel of each convolutional block. The core building block of this DCNN was called 'Inception Module'. This module not only allowed the architecture to increase the depth but also to increase the width. However, unlike VGG, this network used convolutions of different kernel sizes and a total of nine inception modules stacked together, within between max pooling layers to halve the spatial dimensions. It uses global average pooling after the last inception module summing up to a total of 22 layers or 27 layers including the pooling layers. During the next several years, Szegedy *et al.* tried improving the InceptionNet using (i) spatially separable convolutions, (ii) factorizing 5×5 and 7×7 convolutions to two and three 3×3 sequential convolutions, respectively, (iii) adding another layer for batch normalization, and (iv) making inception modules more wider, therefore increasing the size of feature maps. The latest version of the network named, Inception v4, has a top-5 error rate of only 4.2%.

2.1.4 ResNet-50

First developed at Microsoft Research [11] by He et al (2016), ResNet was able to solve the saturated performance problem caused by stacking a larger number of convolution layers in a CNN. The idea behind this architecture was to identity the shortcut connections between two layers and to force the network to learn the residual between the input and the output instead of direct mapping. Another important modification in the ResNet-50 architecture is that there are no fully connected layers except for the one used prior to the softmax function.

This DCNN showed a top-5 error rate of only 3.57%, and with this level of performance, this architecture won the 2015 ILSVR competition.

2.2 Layers in CNN

Since CNN performance can be optimized by varying the depth and breadth of the hidden layers, it is important to understand each layer and its impact on feature learning algorithm. Every layer has its own hyperparameters and they can significantly impact the performance. Therefore, a detailed explanation of each layer along with their working principle, implementation and respective hyperparameters used in a CNN are discussed in this chapter.

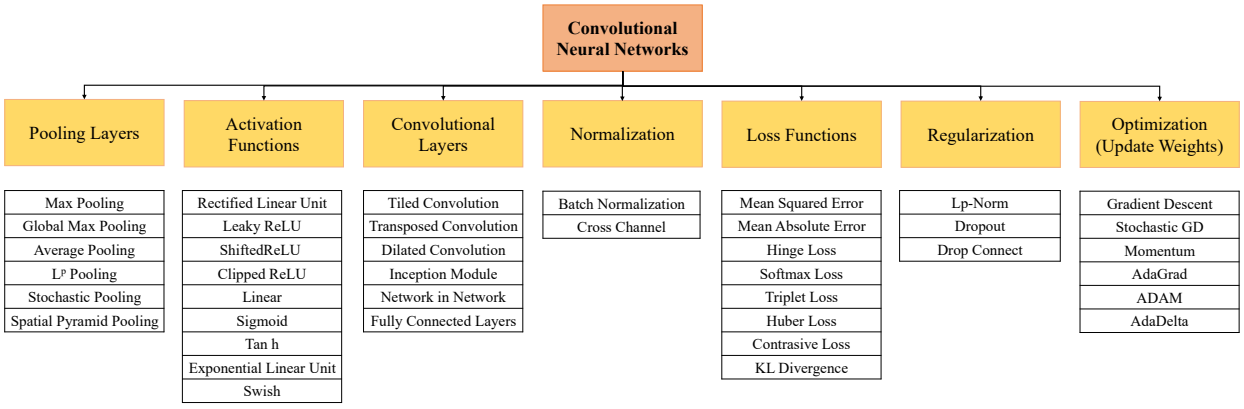


Figure 2.3: CNN Taxonomy.

2.2.1 Input Layer

It is the first layer while designing a CNN model. This layer takes the image data in the form of a multi-dimensional array, also called $rank - n$ tensor. It is necessary to specify the number of images, image size, and the number of channels in the input in this same order. For example, a rank-4 tensor for a sample input of 100 grayscale images of 256×256 input dimension would be, [100, 256, 256, 1]; similarly for a colored image, since it has three

different channels (each for a different color, RGB) the input dimension would be [100, 256, 256, 3]

2.2.2 Convolution Layer

The primary objective of a convolutional layer is to extract the necessary information from an image. It consists of neurons that connect the input layer with the next layer, which will further act as the input layer for the next layer. This kind of network architecture is called, feed forward network. For example, if there are sloping lines and arcs at a certain angle in an image this layer would aim to capture that data, referred to as a feature map, and passes it to the next layer. This is done by using a series of filters called convolutional kernels. A CNN allows the capability to either use pre-defined kernels or create new kernels that suit the input image. Equal to the number of applied filters, a respective feature map is generated, and while the kernel moves along the image matrix it has the same weights and biases for the entire convolution. Every feature map that is generated is the result of such convolutions, see Firgure 2.4 using a different set of weights and biases. Once the convolution weights and biases are assigned to each neuron, and an output is generated, the next step is to compare this output with the actual value. Based on the final output, and how close or far this is from the actual value (error), the values of the parameters are updated and this process is called, backpropagation (short for back propagation of error). The forward propagation process is repeated using the updated parameter values and the CNN model tries to adjust weights by adjusting these kernels through training.

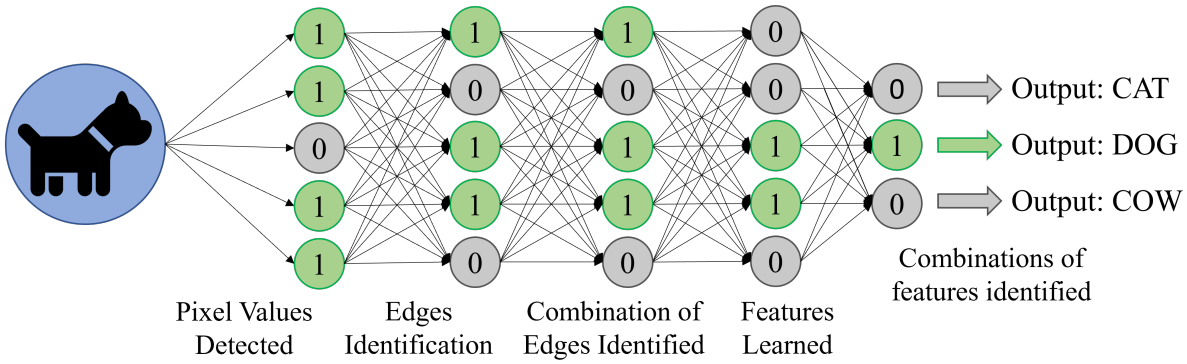
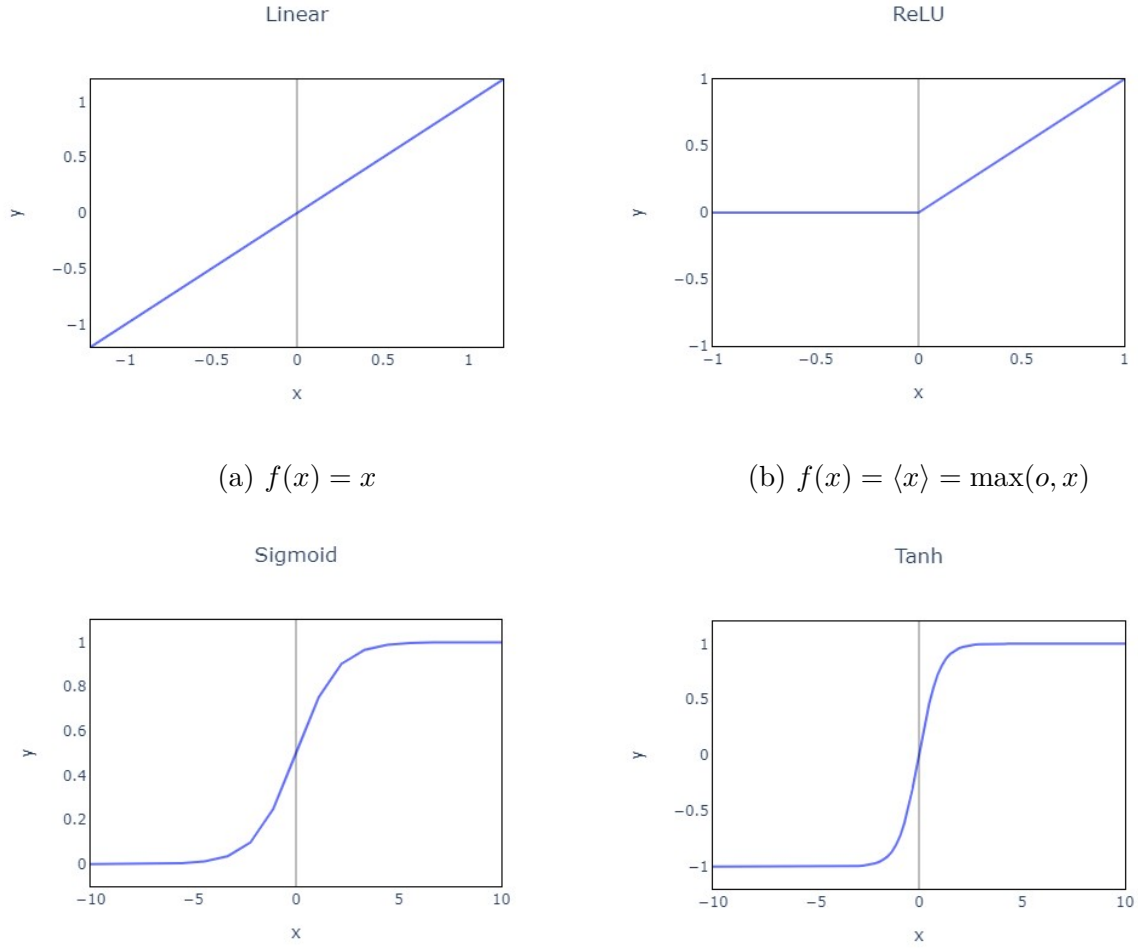


Figure 2.4: Schematic of CNN working of feature and edge extraction.

2.2.3 Activation Layer

An activation function is another aspect of training a neural network. This function is necessary to induce non-linearity in the trained models. Activation functions can be thought of as mathematical gates which allow information to pass through them after a threshold is achieved. There are many activation functions available that can be used to train a deep neural network, Figure 2.5 (b), shows the most popular activation functions used in machine learning. They can be of many different types and thus it is important to select them based on the type of prediction that the neural network needs to make. For this research, out of many available activation functions [10], we have used ReLU as the activation function. There are many advantages of using the ReLu function, through continuous implementations and research on DCNN, it was found that using ReLUs accelerates the training process when compared to other activation functions [16], [21], [30]. This is because of the $max(\cdot)$ operation that allows computing weights and biases much faster than other functions such as *sigmoid*, *tanh*. Another advantage of using *ReLu* is that it also induces the sparsity in the hidden units and thus allows a network to easily obtain a sparse representation, hence accelerating the training time.



(a) $f(x) = x$

(b) $f(x) = \langle x \rangle = \max(o, x)$

(c) $f(x) = \frac{1}{1+e^{-x}}$

(d) $f(x) = \tanh\ (x)$

Figure 2.5: Most commonly used activation functions in deep learning: (a) Linear activation; (b) Rectified Linear unit (ReLu) activation; (c) Sigmoid activation; (d) Tanh.

2.2.4 Pooling Layer

This layer is used to progressively reduce the spatial size of the representation to reduce the number of parameters and computations in the network. This layer operates independently on every depth slice of the input and resizes it spatially using either of the two popularly known settings: MAX pooling, which returns the maximum values of the divided region of

the input, and the AVERAGE pooling, which returns the average value of that region. An average pooling or max pooling layer will reduce the feature maps from a convolutional by one half in each dimension.

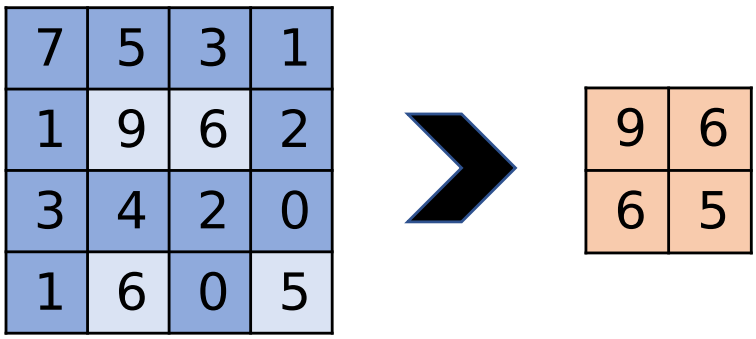


Figure 2.6: Max Pooling.

2.2.5 Fully Connected Layer

In this layer, every neuron is connected to all the neurons in the previous layer, thus called a fully connected (FC) layer. This layer aims to combine all the features learned by the previous set of layers across the image and extract a larger pattern. This layer works on the flattened input where all the connected neurons have full connections to all the activations and this layer is usually inserted at the end in a CNN architecture. The neuron takes the input and multiplies it by a weight matrix W and then adds to it a bias vector b.

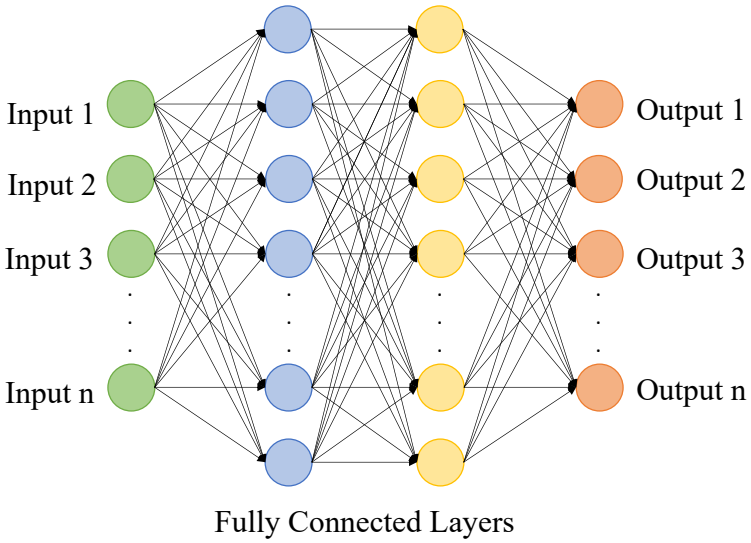


Figure 2.7: Fully Connected (FC) layer: all neuron connected to every single neuron in the next layer.

2.3 Working with Image Graphics

Computers interact with an image differently than we humans do. We interpret them based on objects, contrasts, and differentiating objects in the foreground with backgrounds. However, while using images as datatypes, a computer treats these images as a collection of a multi-dimensional array. In the field of machine learning and deep learning, these n-dimensional arrays are termed as tensors, for instance, 5×1 vector is the first order tensor, and 5×5 is a second order tensor. The success of machine learning algorithms generally depends on data representation, and we hypothesize that this is because different representations can entangle or hide the different explanatory features that bring variation in a dataset. A colored image of dimension 512×512 is a third order tensor: $512 \times 512 \times 3$ where the third dimension denotes the total number of channels: Red, Green, Blue. Removing channels or converting an image to grayscale (channel = 1) can drastically change the image.

There are three image datatypes in which a computer can read and distinguish them, (i) indexed images, (ii) color images, and (iii) grayscale images. The only difference among them is the structure of the tensor as shown in Figure 2.8. Grayscale and colored images are the standard image types where each element value in the array denotes the pixel intensity. However, indexed images have a unique structure, which consists of a data matrix X of $height \times width$, where each value of X is a pointer that denotes the location of RGB values in the color map, therefore must be an integer. A color map is an $m \times 3$ array of class double containing floating-point values in the range $[0, 1]$ and it is loaded automatically with indexed images. The color of each image pixel is determined by using the corresponding value of X as an index to the row in the color map, e.g., the value five in the Figure 2.8 (a) points to the fifth row in the color map and each column in the color map denotes values in order of Red, Green, and Blue.

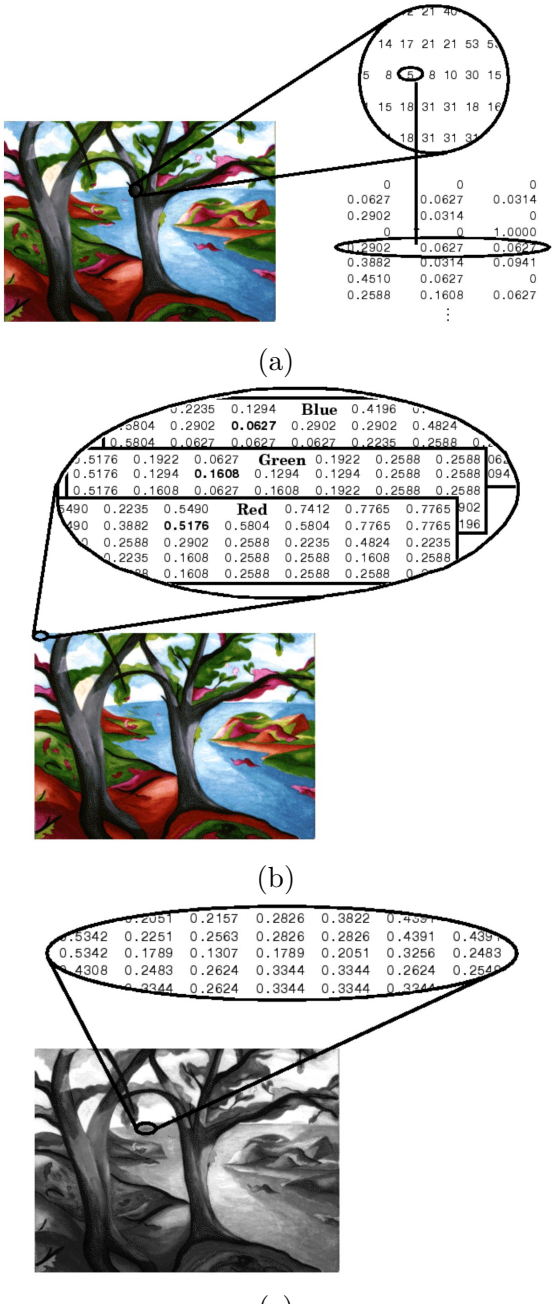


Figure 2.8: Image Types (a) indexed image - 2nd order tensor with 1st order tensor color map; (b) color image - 3rd order tensor; (c) grayscale image - 2nd order tensor. *https* : *//www.mathworks.com/help/matlab/creating_plots/image − types.html*

2.4 Transfer Learning

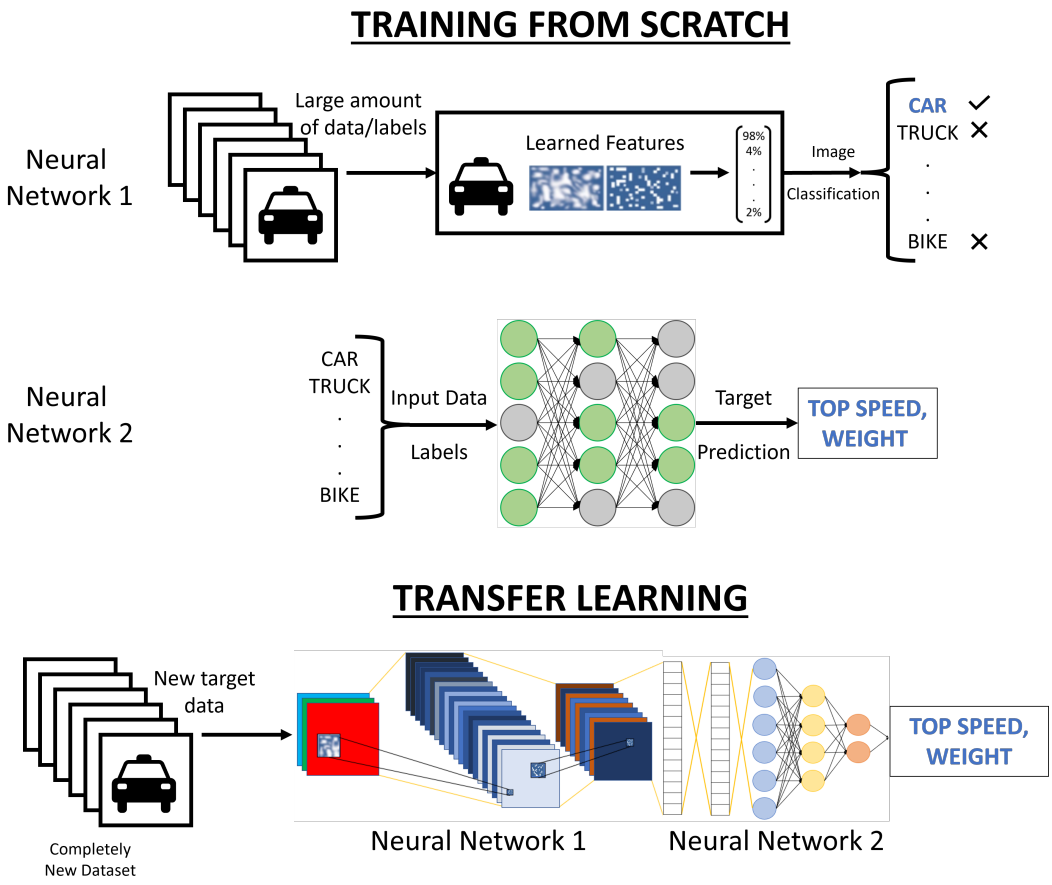


Figure 2.9: Schematic diagram for transfer learning.

Transfer learning is a machine learning method wherein a model developed for a task is reused as the starting point for a model for a second task. The basic premise is to take a deep learning neural network that is trained on a large dataset and transfer the knowledge to a smaller dataset, Figure 2.9 shows the schematic diagram of how transfer learning works. It is a popular approach in deep learning where pre-trained models are used as the starting point for computer vision and natural language processing tasks given the vast computing and time resources required to develop neural network models on these problems and from the huge jumps in a skill that they provide on related problems.

Chapter 3

Methodology

Our proposed approach shows an image processing based surrogate model to develop a data-driven adaptive remeshing scheme. In FEM packages adaptive remeshing, depending on the model, can sometimes take longer run times. Error indicators are used to find high error locations in the given mesh and based on the given remeshing rules adaptive remeshing algorithm performs the iterative steps to reduce the values in error indicator contours. Our surrogate model targets to predict these error indicator contours using a given stress image contour. A three Phase research is conducted to develop a proof of concept approach to build a generalized surrogate model that can predict two different types of error indicators: (i) von Mises error indicator (MISESERI) and (ii) energy density error indicator (ENDENERI). This chapter first discusses the data generation process and how using Abaqus/CAE we developed our input and output datasets. Later, we discuss the different components in the surrogate model and their respective function to predict error indicator. Focus during the phase I was to develop an image to vector regression network and to create a CNN that is able to learn features from a stress contour image and establish a relation with volume averaged von Mises stress values. We started with one of the most basic problem in solid mechanics, a quarter plate with a hole and subject to an applied load in x and y directions, and symmetrical boundary conditions at the left and the bottom edges of the top right quadrant of the plate. Achieving accurate results in phase I it gave us the confidence to move forward with phase II, that was extending the capabilities of a CNN network and

developing an image-image regression network. During this phase, we start with developing a deep neural network that can take the given image as an input and returns an error indicator image. Also, for the phase II neural network training, we improved the image quality from 64×64 used in phase I to 256×256. This change required us to add additional hidden layers to reduce the value of the validation loss and to use powerful graphical processing units (GPUs) to accelerate training. A total of four individual deep neural networks were designed to complete a surrogate model for MISESERI output variable, and two additional DNNs to include ENDENERI output variables. With a single type of geometry used during phase II, we obtained excellent results, but to even improve our surrogate model we developed phase III where we added two additional geometries to the dataset. Dataset for phase III includes a total of three case studies that include four different geometries.

3.1 Model Formulation

To both generate enough data and train a surrogate model with stress and error contours, we first need to define a problem and simulate the structural analysis using the FEA package. We used Abaqus/CAE and Abaqus/Standard to perform static structural analysis on three different case studies, (i) a quarter plate with a hole, (ii) a simply supported plate with multiple holes, and (iii) a simply supported plate with straight stiffeners. For each one of them, four different geometries were built. Leveraging Abaqus/CAE's Python interface, it became easier to develop a Python function to automate the model generation and execute the stress analysis simulation. To post process the results another function was designed to save the desired data in the portable network graphics (PNG) format. A medium sized mesh was used for all case studies and their sub geometries. This allowed us to capture significant errors in the mesh corresponding to the von Mises stress contours. Shell

elements in 3-D space available in Abaqus/Standard are based on three different formulations: general-purpose, thin-only, and thick-only. The three-dimensional "thick" and "thin" element types provide for arbitrarily large rotations but only small strains. The general-purpose shells allow the shell thickness to change with the element deformation. The S4 element in Abaqus/Standard is a general-purpose, fully integrated, finite-membrane-strain shell element. It has four integration points on each section point defined throughout the shell thickness at which Abaqus/Standard computes the desired output variable. For shell sections integrated during the analysis, one can define the number of section points through the thickness for integration. The default is five for Simpson's rule and three for Gauss quadrature [25]. Although there can be an infinite number of section points, we chose to go by the default number of five section points to model our geometry. Later, after applying the boundary conditions and loading along the plate top and right edges, we developed a result Python function to post-process the results. For phase I this function was developed in such a way that it was able to extract the von Mises stress values from every integration point and summed across all section points and volume averaged for every individual element available in the discretized domain.

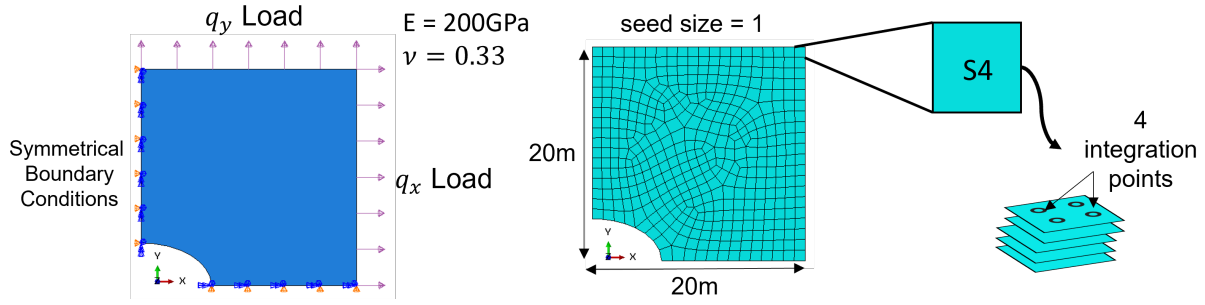


Figure 3.1: Example model for case I, with symmetrical boundary conditions and applied loads along with generated mesh.

The result from this phase I Python function was a von Mises stress contour along with a data file that consists of volume average stress values corresponding to each stress contour for

every load case applied for the first case study: Quarter plate with a hole. Later, for phase II and phase III, these Python functions were modified to include additional commands to extract the MISESERI and ENDENERI contours for each load case and save it in a local directory. Load parameterization and the contour limits for post-processing were kept the same for all case studies and discussed in the next section. Similarly, another Python function was developed to model and simulate the stress analysis for the second case study which is a simply supported plate with multiple holes. Random holes in circular and elliptical holes were created in the plate. The size and number of these holes vary across each of the four sub geometries developed to provide a more robust dataset for the neural network. Figures 3.2, 3.3, and 3.5 show the three different case studies used.

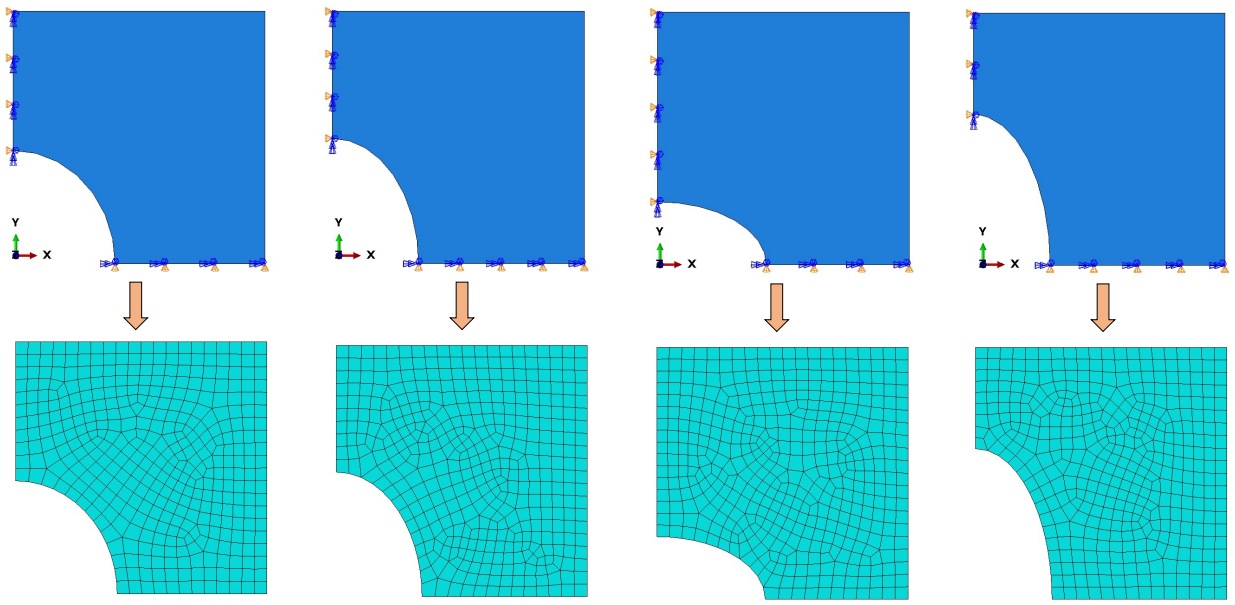


Figure 3.2: Case study 1: Right quadrant of a Plate with hole at the center: symmetrical boundary conditions (above) and meshed geometry (below).

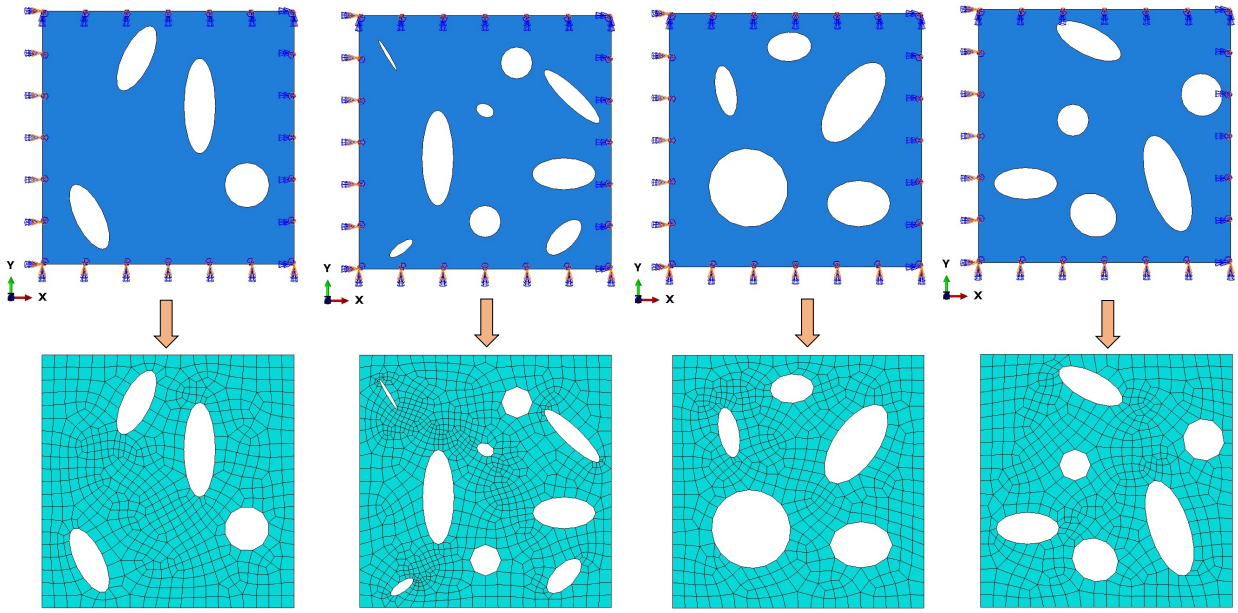
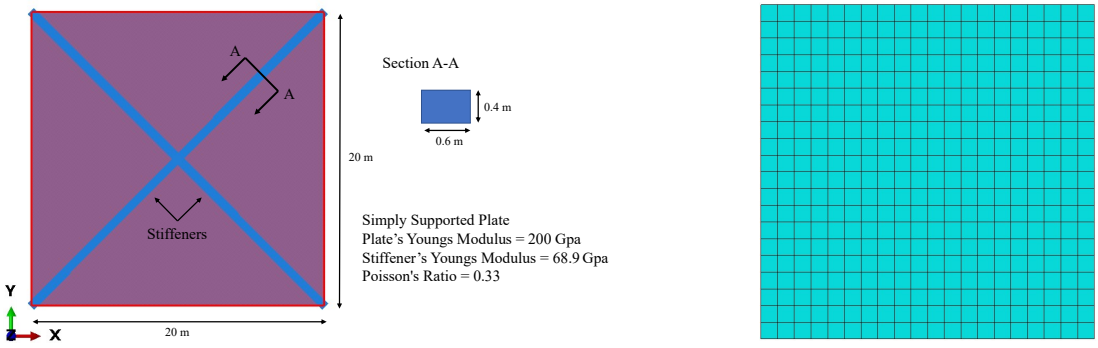


Figure 3.3: Case study 2: plate with multiple holes: simply supported boundary conditions (above) and meshed geometry (below).

To develop the model and geometry for the third case study which is a simply supported stiffened plate, an example on buckling analysis of stiffened plates from Zhao and Kapania [32] and Tamijani and Kapania [29] were found to be good resources. Figure 3.4 shows the cross-section of stiffeners and model properties for the stiffened plate model along with the meshed geometry.



(a) (b)

Figure 3.4: Case study 3: (a) cross section and stiffener alignment; (b) meshed stiffened plate

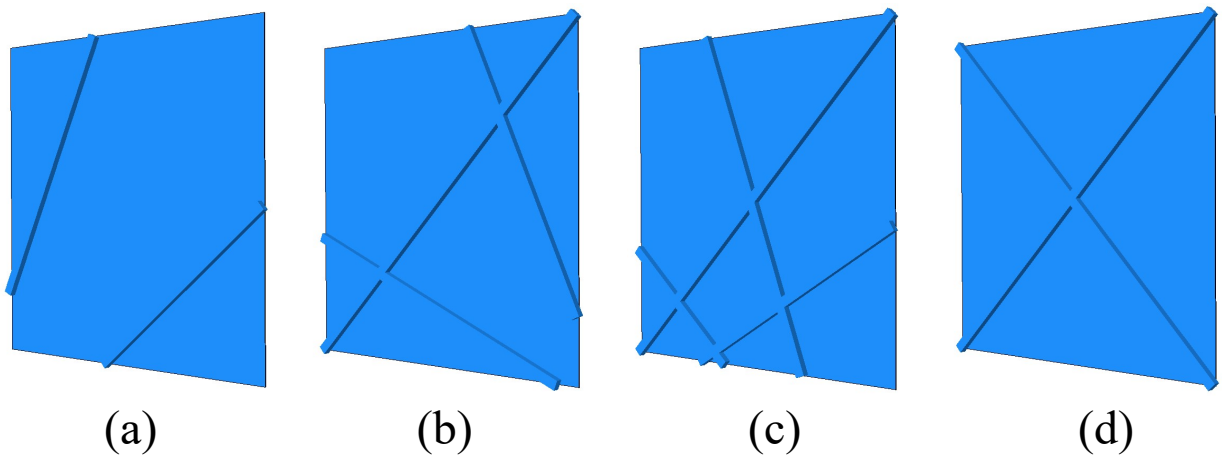


Figure 3.5: Case study 3: simply supported stiffened Plate Rendered assembly view for stiffener and plate.

3.2 Data Generation

The performance of machine learning methods is heavily dependent on the choice of data representation (or features) on which they are applied. For that reason, much of the actual effort in deploying machine learning algorithms goes into the design of pre-processing pipelines and data transformations that result in a representation of the data that can support effective machine learning. For our current regression problem in phase I, the main input data were the images of von Mises stress contours and the final output volume averaged values of von Mises stress, see Figure 3.6.

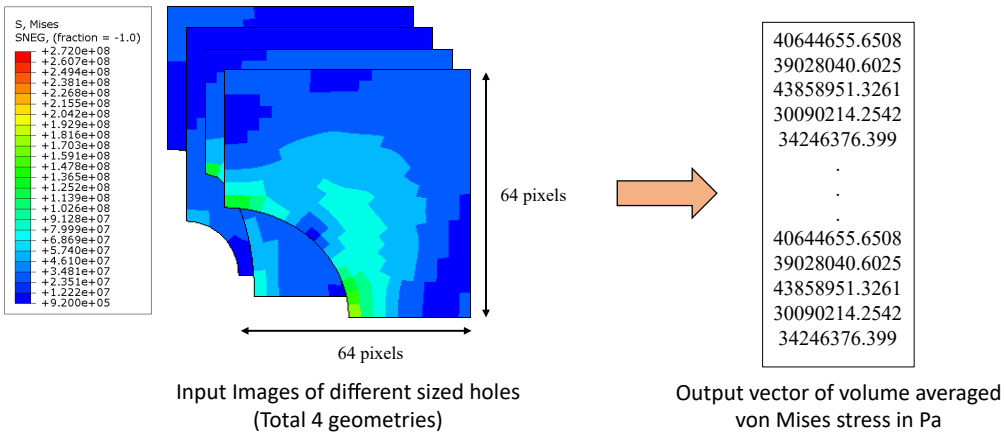


Figure 3.6: Collection of input images from 4 different geometries (left) and volume averaged von Mises stress vector as output (right).

For phase II and phase III, our datasets consist of all images both for input and output. The Python function developed to extract images from Abaqus/CAE saves images using quilt feature, which is saving single contour color per element. This was done to incorporate the mesh information inside the error indicator and von Mises stress images. With change in mesh density, the quilt feature not only allows to provide a per element value but also adequately represent the spatial gradients. Initial training of our surrogate model was done using von Mises stress contours as input and just the MISESERI error indicators as output, as shown in 3.7. Since for phase III we included two more case studies in addition to the plate with hole model with four distinguished geometries respectively, we had a total of 12 different geometries. Figures 3.8, 3.9, and 3.10, shows both input and two desired output error indicator image datasets for all the three different case studies having four different geometries respectively.

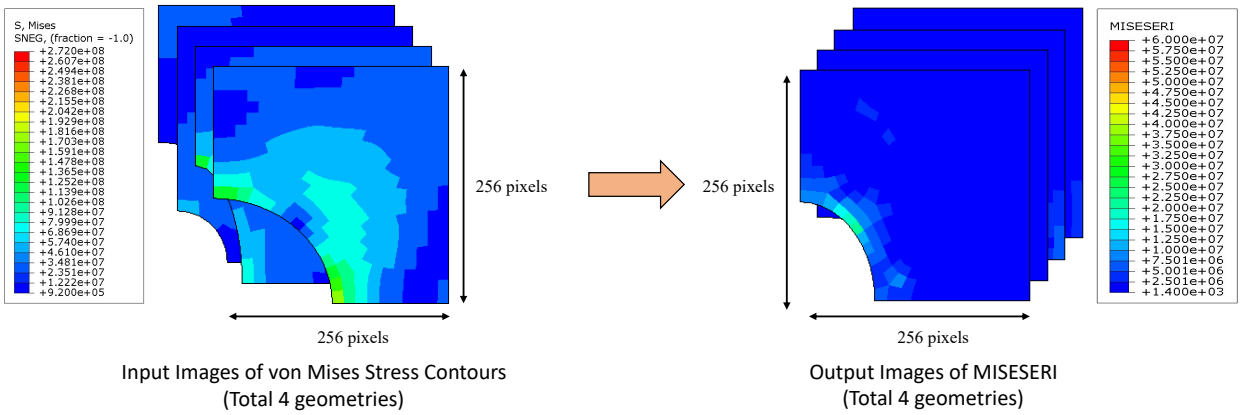


Figure 3.7: Collection of stress contours input images (left) and MISESERI images as output (right).

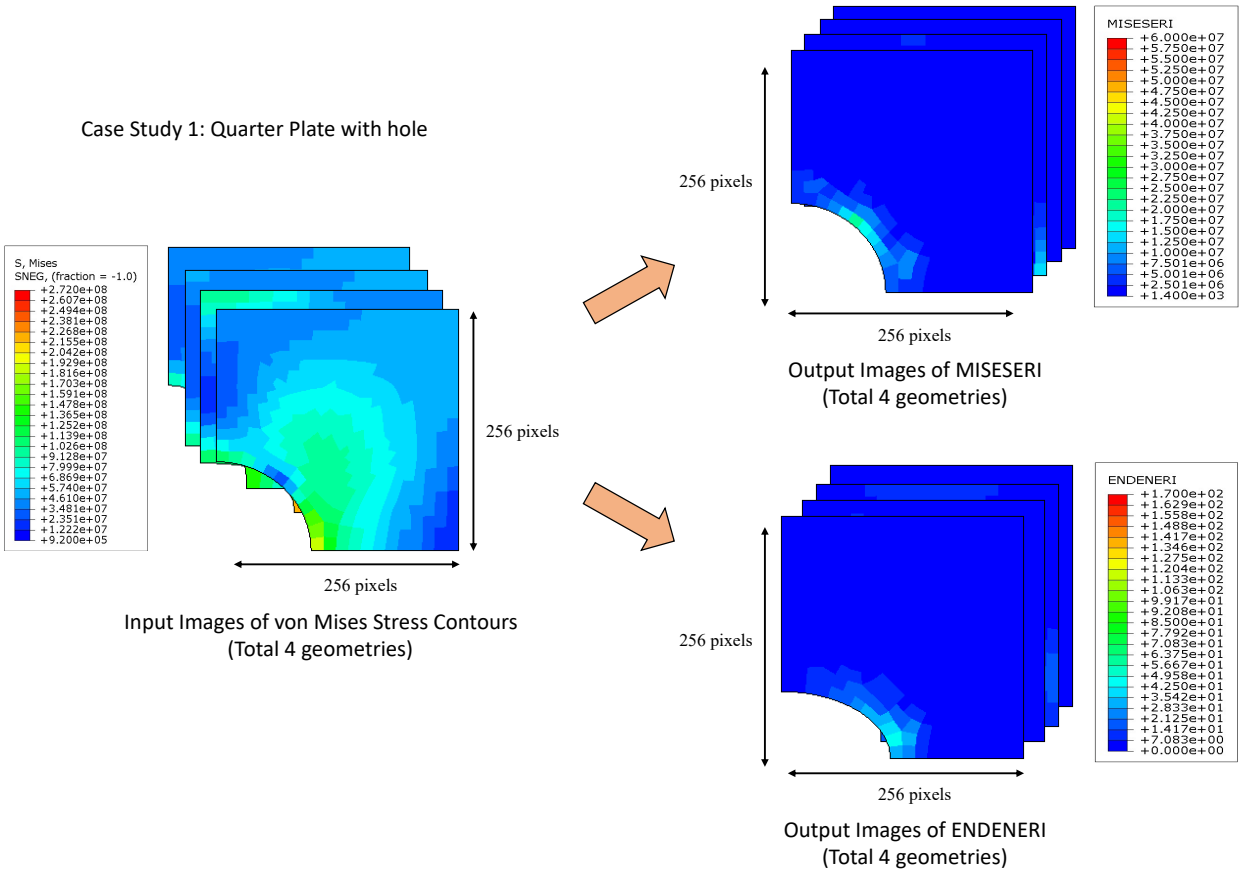


Figure 3.8: Case study 1: quarter plate with hole - input: von Mises stress contours; output 1: MISESERI images; output 2: ENDENERI images.

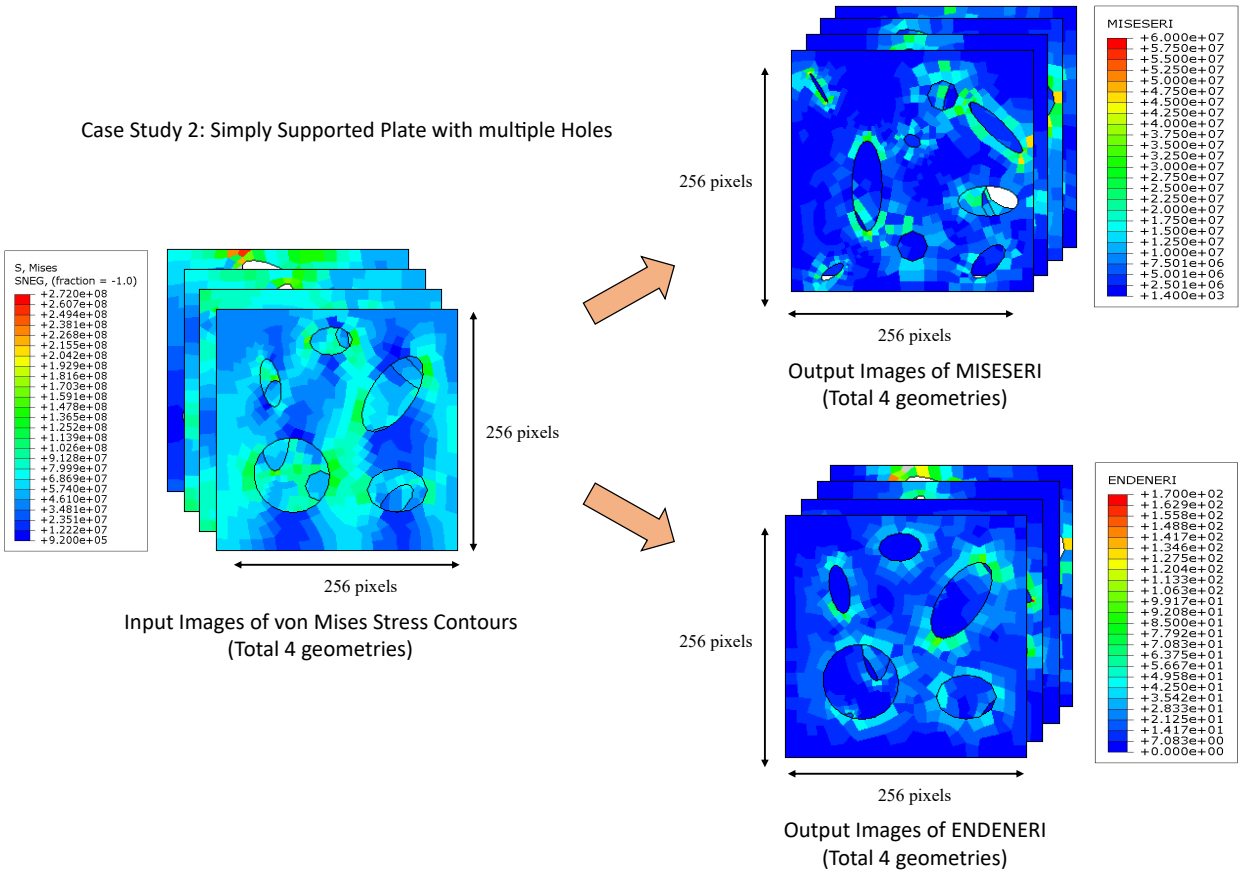


Figure 3.9: Case study 2: plate with multiple holes - input: von Mises stress contours; output 1: MISESERI images; output 2: ENDENERI images.

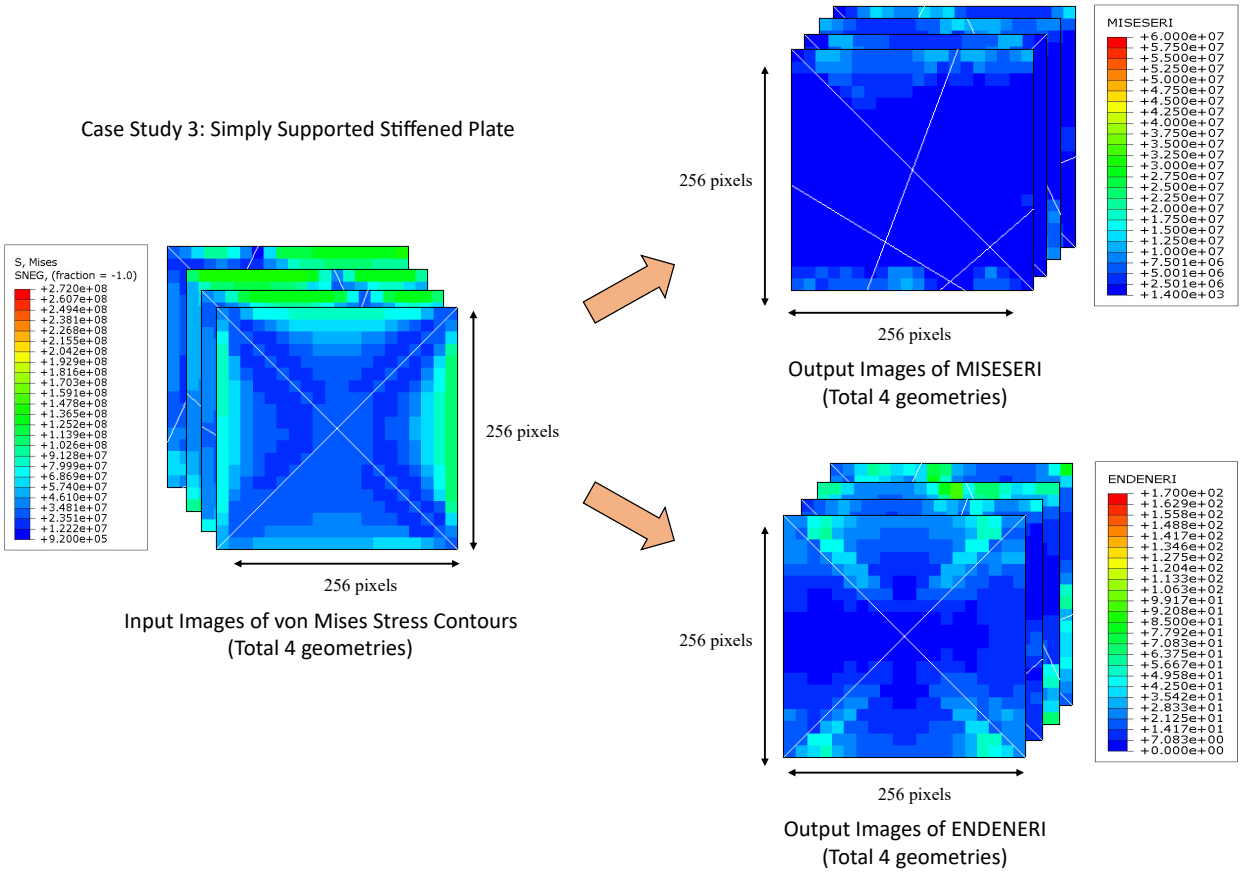


Figure 3.10: Stiffened plate - input: von Mises stress contours; output 1: MISESERI images; output 2: ENDENERI images.

Our three different case studies include four small datasets each with different hole dimensions for the first case study, a different number of holes for the second case study, and a different stiffener geometry for the third case study. A global dataset that combines all the data points from these four small datasets. Each smaller dataset contains a total of 500 colored images having the dimension size of 256×256 pixels for both the stress and the error images. The global dataset is a collection of 6000 images each for stress inputs and both of the error indicator outputs.

To have a diverse image dataset, a unique load case approach is designed and implemented in the model generation script that will leverage the load case tool available in Abaqus/S-

tandard. Loads are applied both in horizontal F_x and vertical F_y directions. For each data point (F_x and F_y) are constant due to the static problem and determined by the resultant force F and its direction given by θ: $F_x = Fcos\theta$ and $F_y = Fsin\theta$. Table 3.1 shows the four different load variations that are considered to simulate the load diverse conditions. From the four given options, we can choose any one of them to parameterize the load cases, we have selected option C.

	θ in 4 Quadrant				Coordinate Signs	
Case	I	II	II	IV	x	y
1	X				+1	+1
2		X			-1	+1
3			X		-1	-1
4				X	+1	-1

Table 3.1: Different cases for implementing load variations.

1. Option A: Case 1, and 2: $0 \leq \theta \leq \pi$
2. Option B: Case 3, and 4: $\pi \leq \theta \leq 2\pi$
3. Option C: Case 1, and 4: $-\frac{\pi}{2} \leq \theta \leq \frac{\pi}{2}$
4. Option D: Case 2, and 3: $\frac{\pi}{2} \leq \theta \leq \frac{3\pi}{2}$

3.3 Data Pre-Processing

In machine learning (ML), once the data is collected it is necessary to perform data pre-processing, to avoid training errors and higher loss values for training and validation datasets. Data pre-processing involves four sub steps such as data integration, data transformation,

dimensionality reduction, and data cleaning. The current analysis problem deals with volume averaged values of von Mises stress as output and our input is the colored images of von Mises stress contours. Normalization is a data transformation technique that is widely used for pre-processing algorithms. It is used when the numerical attributes of data are scaled up or down and we need to fit the data within a specified range. (a) Min-max normalization, (b) Z-Score normalization, and (c) Decimal scaling normalization are few popular normalization techniques. With a closer look at phase I input and output datasets, it can be observed that output values are in the range of $10^6 - 10^7$ Pa while our input is a fourth order tensor having values in between 0-255. Thus, training can be entirely impossible because both inputs and outputs datasets are not on the same scale and the difference is very large. To bring both input and output both on the same scale it is best to perform min-max normalization, shown in Eq. 3.1 which will transform the entire dataset in the range of 0-1. [15]

$$\bar{x} = \frac{x - x_{min}}{x_{max} - x_{min}} \tag{3.1}$$

3.4 Encoder-Decoder

Inspired by the success of these networks in neural machine translation [27], which is converting source text in one language to text in another language [12], encoder-decoder showed great progress in image processing as well. DCNNs designed as encoder-decoder networks were widely used for image segmentation, image denoising, and a highly popular application is in designing generative models (GANs). Such models are feed forward neural networks with a unique network architecture that is comprised of both convolutional layers as well as deconvolutional layers, both in a single model. One should not confuse encoder-decoder with

a separate type of neural network, it is a network architecture design that can be made using any type of known neural network such as CNN or recurrent neural network (RNN). There are two main tasks involved to perform successful training, down sampling, and up sampling. As the name suggests, the down sampling (Encoder Network) extracts the necessary information from the input by reducing the spatial dimensions.

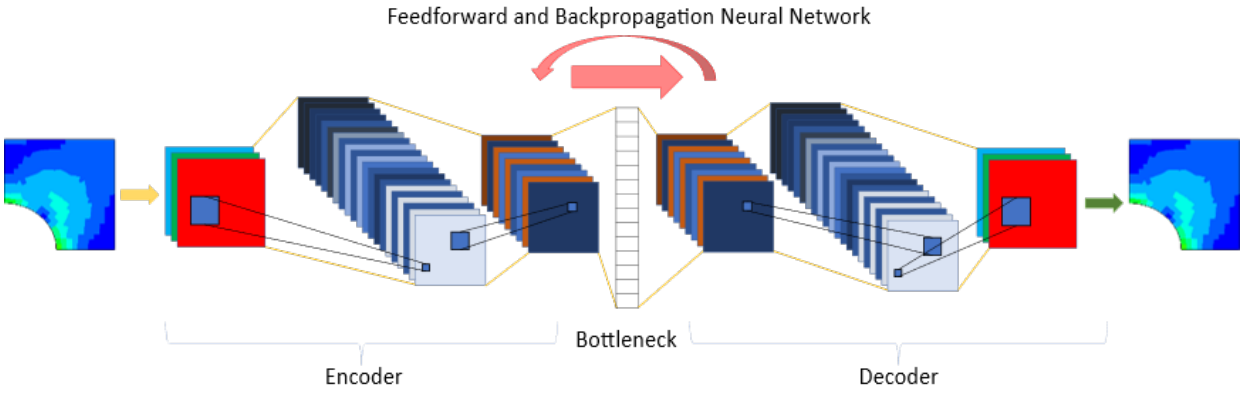


Figure 3.11: Working of autoencoder network architecture.

In DCNN, it is performed using multiple combinations of convolution, activation, pooling, and normalization layers. Similarly, in an up sampling network or commonly referred to as a decoder network, the encoded vector is translated back to the image format by making the spatial dimensions equal to that of the input image. The bottleneck layer contains the lowest possible encoded state of the input and acts as the input layer for the decoder. The simplest form of encoder-decoder is an auto encoder (AE), as shown in Figure 3.11 with an input and output layer connected with more than one hidden layer. AEs are popularly used to reconstruct the given input and show them as output. This type of learning is referred to as unsupervised learning where the model adjusts both the training weights and the biases by itself, without any specific output provided during the training. Our auto encoder serves the same purpose, it takes the stress images as input and tries to regenerate them. Since we designed two different auto encoders, we were successfully able to avoid the following two

problems, (i) a mix-up of the training weights with input/output, and (ii) mix-up of the contour limits.

Similar to an encoder network convolution layers are used to downsample the input, in the decoder network there are two common types of layers used, (i) upsample layer (UpSampling2D) and (ii) transpose convolutional layer (Conv2DTranspose) to up sample the encoded state vector. Where Conv2DTranspose layer can perform an inverse convolution operation, UpSampling2D simply doubles the dimensions of the input. Transpose convolutional layer tries to interpret the coarse input data to fill in some of the details while performing the upsample operation. This layer is more complex than the UpSampling2D layer because this layer combines both the UpSampling2D and Conv2D layers into one layer. From the following different techniques available, namely (i) nearest neighbors, (ii) bi-linear interpolation, (iii) bed of nails, and (iv) max-unpooling. The decoder model utilizes the max-unpooling technique. Here, the UpSampling2D layer takes the maximum value among all the values in the kernel. To perform max-unpooling, first, the index of the maximum value is saved for every max-pooling layer during the encoding step. A saved index is then used during the Decoding step where the input pixel is mapped to the saved index, filling zeros everywhere else. A major disadvantage of these upscaling techniques is that they are pre-defined techniques that do not depend on the input data. This prevents the network to learn and getting trained from the given input image. To overcome this limitation, Transposed Convolutions are considered to be a better choice, since they perform an inverse convolution operation.

3.5 Multi-Output Regression

Typically, regression refers to a predictive modeling problem that involves predicting a single output value for a specific value of the input. But our approach is a little different. With the

regression network as the bottleneck of the third encoder-decoder network, we aim to create a mapping model that can connect the encoded state vector of stress images to the encoded state vector of the error indicators. This problem is thus a multi-input and multi-output regression problem because, for each image in the input array, the encoder will try to create a multidimensional encoded vector, see Figure 3.12

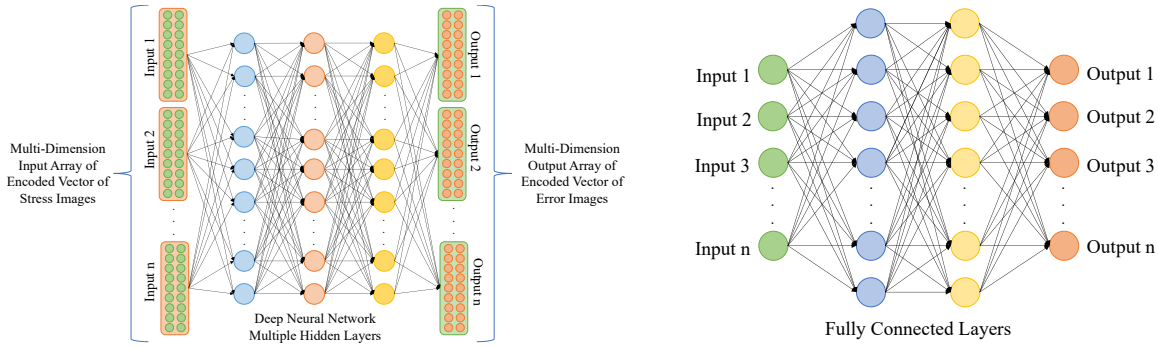


Figure 3.12: Multi-Output Regression Model compared with a) Generic Regression Model b) A Deep Regression Model implemented at bottleneck layer of the third encoder-decoder.

3.6 Surrogate Model

We have introduced a surrogate model for automatically predicting error as a part of a building block for a data-driven adaptive remeshing tool. This tool leverages Deep CNNs that take von Mises stress images as input and the output is the von Mises Error Indicator. Our model is a joint architecture of encoder-decoder contribution neural nets with the multi-output, fully connected regression network in the latent space or bottleneck layer. The Encoder part of the architecture performs feature extraction on the input images, this will result in optimizing training weights using the backpropagation technique. Figure 3.13 shows a simple CNN architecture designed to extract features from input von Mises stress images to perform training during Phase I. The FC regression network creates a mapping function

between the encoded state of both input and output. The final stage is the decoder network, it will translate the encodings back to the dimensions of the input image.

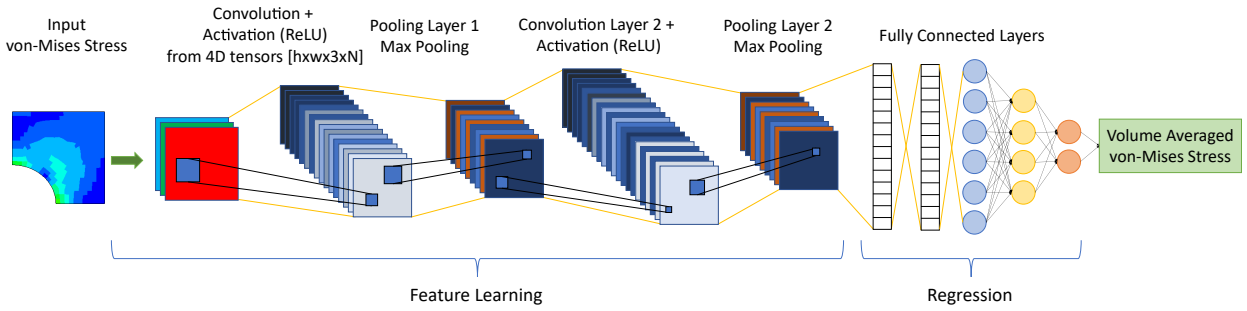


Figure 3.13: Convolution neural network architecture used for training for phase I.

Now, when the stress images are passed through the auto encoder it will optimize the training weights according to the stress image. Although, we do not need the decoder training weights for stress images, however, training the decoder network will act as a verification network for reducing the error. However, we do need both encoder and decoder networks for error images. Now, if the error images are also passed along with the stress images through the same auto encoder, it will optimize the training weights for the collective collection of both types of input and output images, see Figure 3.14. To avoid this mix-up of training weights, two different auto encoders were designed individually one each for stress and error images respectively.

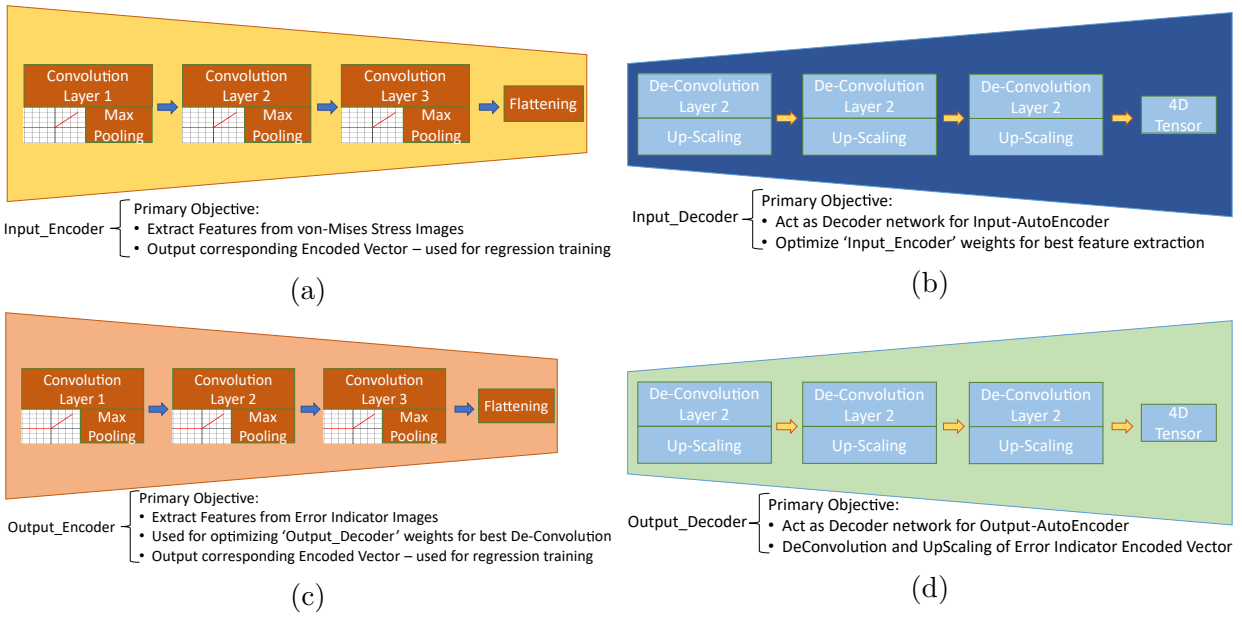


Figure 3.14: Individual components of surrogate model: (a) input encoder; (b) input decoder; (c) output encoder; (d) output decoder

The second auto encoder is designed exactly in the same way as the first auto encoder so that the dimensions of the encoded vector for output images are exactly the same as the dimensions of the encoded vector for the input image. Once the error images are passed through the second auto encoder, we obtained an encoded state vector of downsampled error images, which will be used in the next stage of the surrogate model. After a successful training of both the auto encoder, the next stage is designing a third encoder-decoder combination with a transfer learning regression model. This third neural net is just an assembly of the Encoder network from the first auto encoder used for the input image and the decoder network from the second auto encoder used for the output image and a multi-output regression model designed with FC layers, see Figure 2.7, in the bottleneck, see Figure 3.15. The bottleneck is a region in between the encoder and the decoder networks that is used to connect the flattened output of the encoder to the input layer of the decoder network. Encoders and decoders presented in this network, instead of employing their own weights, leverage the

weights from the earlier training of the input and the output auto encoders. However, the regression model in the bottleneck is trained separately to perform transfer learning. This would be a supervised learning model where the input is the encoded state of the stress images and the output is the encoded state of the error images.

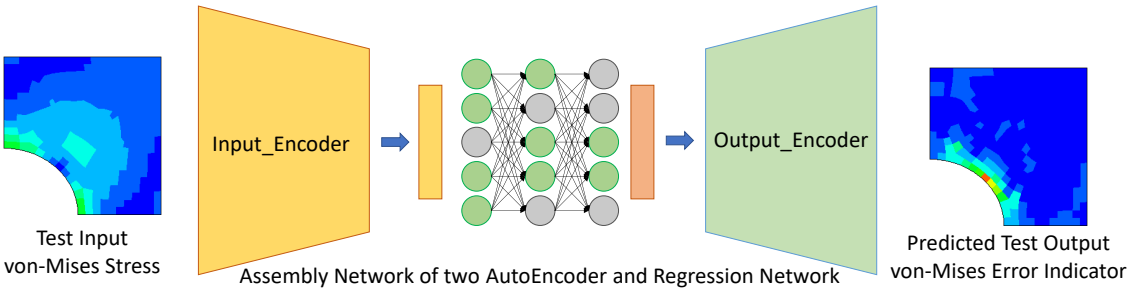


Figure 3.15: Assembly and final representation of the working model.

First, we have evaluated our approach on a single dataset and found good accuracy within the results, the ground truth and predicted error indicator images look exactly the same. The first training is done with a total of 1000 images out of which 80% are used as training images, 10% as validation, and the remaining 10% as testing datasets. To increase the generalization of our proof-of-concept (PoC), we designed more datasets by varying the size of the ellipse in our FEM model. With Abaqus Python scripting capabilities, it was very convenient to generate such datasets as per the requirement and to perform training on the deep learning models using the High Performance Computers available at Virginia Tech's - Advanced Research Computing (VT-ARC).

3.7 Training Frameworks

There are multiple framework available to design CNN networks, it all depends on the comfortability of the user. Our developed surrogate model is based on TensorFlow [1] and Keras libraries [8] to design CNN architectures.

3.7.1 Loss Function Minimization

The primary objective of the optimizer used in training a neural network is to update the weight parameters to minimize the loss function, thus providing more accuracy with the testing datasets. The mathematical function for optimizers can be as simple as subtracting the gradients from the weights, or can be very complex. The choice of an appropriate optimizer has a huge influence on the performance of neural network training. A good optimizer is the one which is faster, efficient, and improves the generalization of the neural network, meaning avoiding overfitting. To train our surrogate model we have used '*Adam*' (Adaptive Moment Estimation) [13] as the optimizer to reduce the loss function in both of the auto encoders. We also tested our CNN model with different available optimizers, e.g., *SGD* (Stochastic Gradient Descent) and it was found that *Adam* worked most efficiently. It combines the advantages of two popular optimizers, *AdaGrad* and *RMSProp* making it more robust and well suited to a wide range of non-convex optimization problems in the field of machine learning [13]. This optimizer is computationally efficient as it requires first-order gradients, that can be computed using very little memory.

3.7.2 The Loss Function

For any neural network, the error function, conventionally called a loss function is the most important component and it is highly dependent on the data type used for input and output and also on the type of machine learning problem. Since, the error for the current state of the model is estimated repeatedly at every epoch, the convergence of the optimization algorithm requires an appropriate choice for an error function. With each error evaluation during the training, the optimizer updates the weights to reduce the loss on the next evaluation. For regression problems, there are plenty of choices: (a) Mean Absolute Error (MAE), (b)

Mean Absolute Percentage Error (MAPE), (c) Mean Squared Error (MSE), (d) Root Mean Squared Error (RMSE), (e) Huber Loss, (f) Log-Cosh Loss and when working with image data MAE, MSE becomes the top choices. We are working with image data and our problem is regression analysis therefore from the list of available loss functions, Figure 2.3, we chose Mean Absolute Error (MAE) as our loss function to train both of the auto encoders. This was done for two reasons, (i) it is the simplest of the available loss functions to calculate the loss function, shown in Eq. 3.2, and (ii) as a result, it is computationally inexpensive.

$$MAE = \frac{1}{n}\sum_{i=1}^{n} |y_{predicted} - y_{actual}| \tag{3.2}$$

3.7.3 Accelerating Training Time

Training a CNN model is highly computational intensive which can lead to longer wait times. To accelerate this training instead of modifying the network architecture, parallel computation implementation is a very efficient way that can be achieved using GPUs to perform the training. In [26] Strigl *et al.*, performed 1000 learning iterations of a LeNet5 model on both GPUs and CPUs and found that with a graphic card involved, training with GPU was found to be 2 to 24 times faster than when done only with using a CPU. However, this training time depends on the network topology; however, for a given topology, a GPU based CNN scales much better than the one trained using CPU. This offers great capabilities to execute machine learning models at an accelerated rate. We leveraged an HPC node that consists of a CPU with 16 cores and a very powerful NVIDIA A1000 GPU for accelerating the surrogate model training.

Chapter 4

Results

The most important aspect of our approach is the efficiency of our CNN model. To understand the feasibility of the method proposed in this research, we designed a two phase approach where we first checked the capabilities of the CNN model and then confirmed if one can train the network with stress images and learns to encode the RGB values to predict volume averaged stress values. From the results of Phase I, the convolutional neural network was found to be an efficient approach to successfully solve an image to a vector regression problem. With exceptionally good results obtained in terms of the regression performance metrics, we proceeded to Phase II. The details of the two approaches are described next. Phase III was developed to extend the capabilities of phase II, with a focus to generalize the surrogate model.

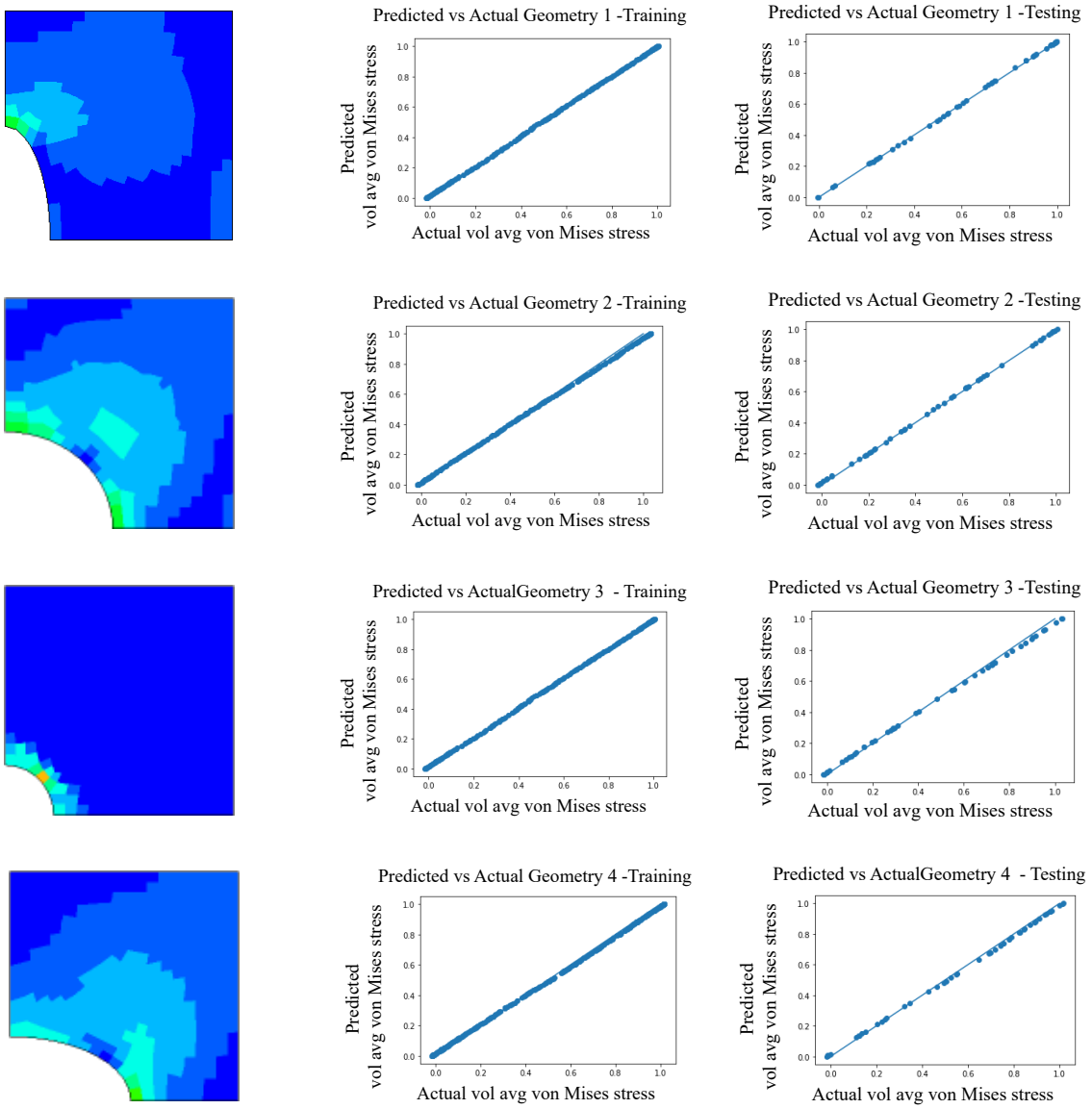


Figure 4.1: Regression and training plots for four different geometry cases.

4.1 Phase I

To understand the learning capabilities of CNNs, four different datasets are trained separately, and based on the results obtained, the hyperparameters are fine-tuned to obtain the best fit. For phase I, we only used the right quadrant of the plate with a hole model with

symmetrical boundary conditions on the left and bottom edges. Each dataset consists of 500 input images of von Mises stress contours and a corresponding vector of volume averaged von Mises stress as output. To vary the dataset, we parameterize the loads acting on the top and the right edges. It is found that for the validation dataset the mean absolute error varies from 0.7% to 1.3% and for the training dataset it varies from 0.8% to 1.5%. In Figure 4.1, we can see one input image and two corresponding regression plots for the training and testing dataset. These results are developed for all four geometries developed for the first case study. Each of these geometry cases is trained separately and two regression plots for training and testing datasets are obtained. These plots compare the predicted volume averaged von Mises stress value using the trained neural network with actual volume averaged stress value.

For each geometry, we can see that the data points follow the $x = y$ line, also called the regression line. Such results indicate that the training is successfully done and the neural network weights are perfectly optimized to capture the maximum number of features. While training for 500 epochs it took close to two hours to complete the training. A separate plot loss curve on the training and validation set was also obtained for each geometry case to monitor the training progress. The hyperparameters were tuned accordingly to avoid the overfitting and underfitting of the model. The true power of a neural network depends on its performance on a generalized dataset. Therefore, a single dataset is made by combining all the four geometry cases, and the neural network training is done again to monitor the performance. Figure 4.2 shows the overall performance of the model and it can be seen that almost all the data points follow the regression line, which means that the predicted volume averaged stress values are equal to the actual values. From this, we can infer that our CNN model is working well on all the datasets and can be further used for Phase II. These results not only validate our CNN model but also show the generalization capability of our neural

network. From these results, we can also conclude that the current CNN model architecture and the associated training hyperparameters are adequate to be used as an encoder model for Phase II.

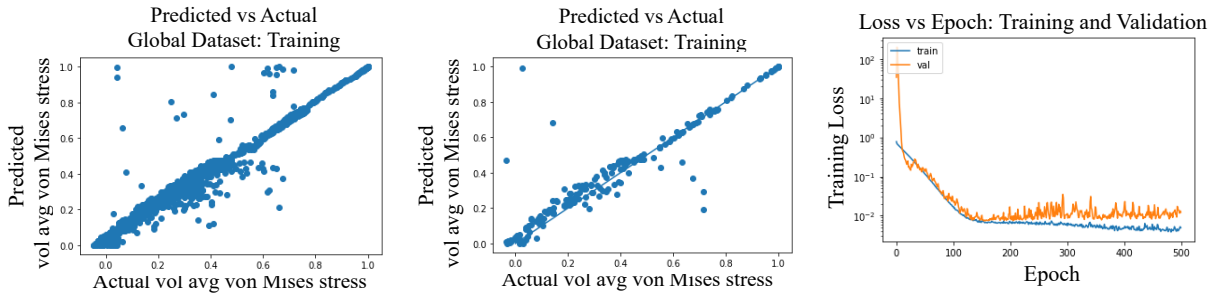
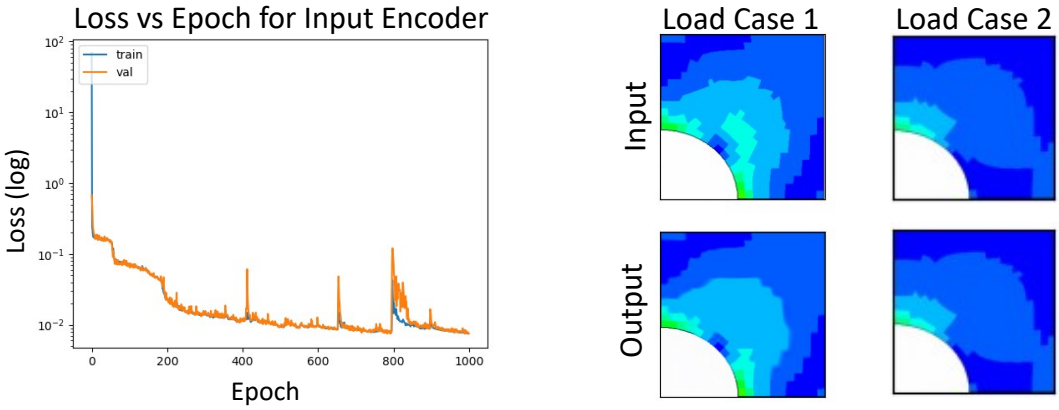


Figure 4.2: Regression and training results for the combined dataset.

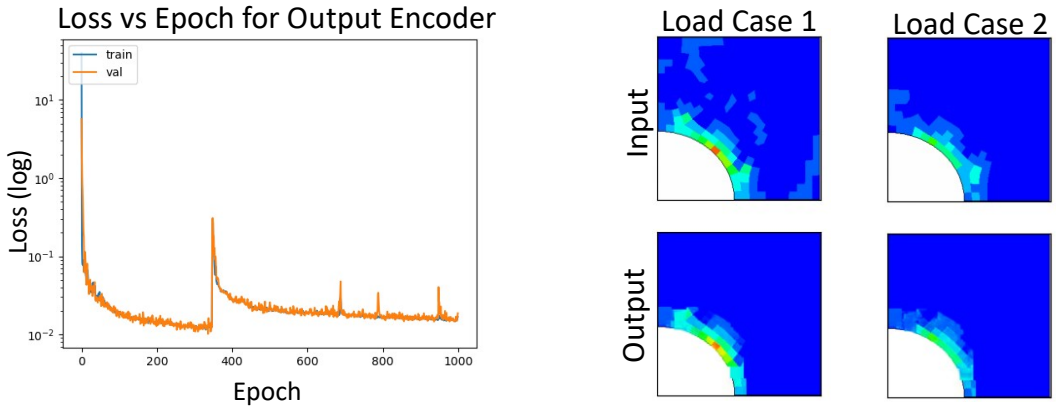
4.2 Phase II

Phase II is related to the design and development of a surrogate model to obtain error indicator images from stress images; in other terms, a model for image-to-image regression. The entire performance of the surrogate model depends on the training of input encoder-decoder (Input Auto Encoder) and output encoder-decoder (Output Auto Encoder) networks. Figure 4.3 shows training results for the Input Auto Encoder, while Figure 4.4 shows those for the Output Auto Encoder. We can see that the input images look exactly like the output image. Such kind of results are defined as the perfect training result for an auto encoder. It took 58.06 minutes and 56.27 minutes by input and output auto encoders respectively to complete the training on a dataset with 1000 images.



(a) (b)

Figure 4.3: Training results for input auto encoder: (a) Loss vs Epoch plot for both training and validation; (b) training results for the test dataset.



(a) (b)

Figure 4.4: Training results for output autoencoder: (a) loss vs epoch plot for both training and validation; (b) training result for the test dataset.

After verifying the results of both auto encoders, the next step is to connect these two networks using fully connected layers. This is required because we need to send the input encoded vectors to regression network to perform supervised learning with encoded vectors of the output dataset. This completes the training part of the surrogate model. Now the next steps is to run this trained surrogate model on the testing datasets. Figure 4.5 shows that the predicted image of error indicator matches exactly with the ground truth.

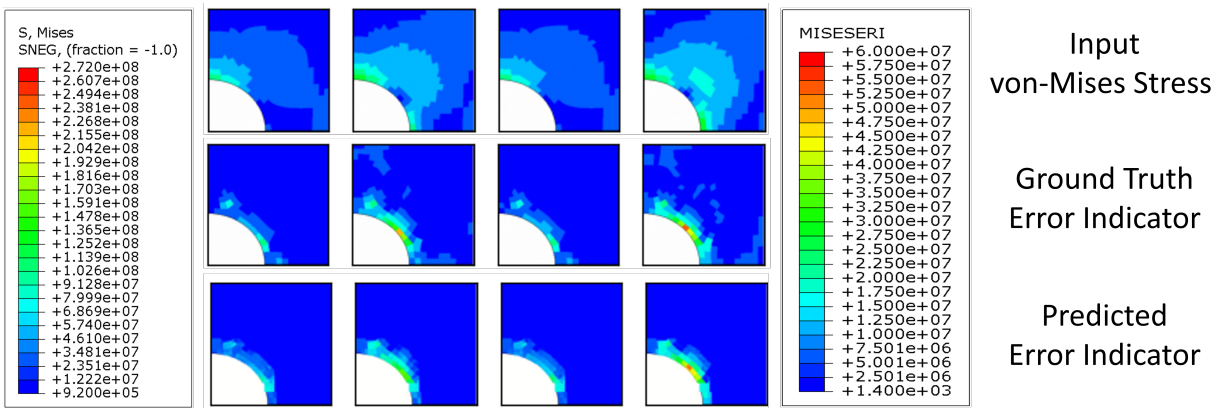


Figure 4.5: Performance of surrogate model for a single geometry.

Just as we did in phase I, the final surrogate model is also trained again to monitor its performance using the combined dataset. Here, the dataset contains 4000 images for panels with holes of different sizes and each data point represents a different load case. The surrogate model showed good accuracy in predicting error indicator images using given von Mises stress images. In Figure 4.6, we can see four different geometries with the first row of von Mises stress image as inputs, the second row as the corresponding error indicator image as the ground truth, and the third row predicted error indicator image. As we can see that the second and the third rows match exactly, we can conclude that the designed surrogate model is perfectly trained with good efficiency. To accelerate the training time, the batch size of 64 was increased to 128 images, and the number of the epoch is reduced from 1000 to 600. At the same time, we decreased the learning rate to 0.001 to maintain the accuracy of the results.

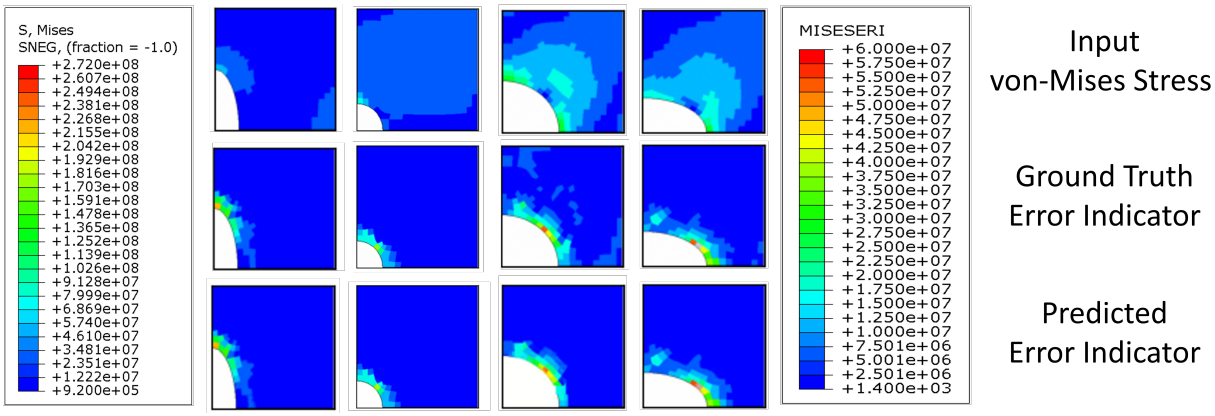


Figure 4.6: Performance of surrogate model on a global dataset that contains 4 different geometries.

4.3 Phase III

This sub-section shows the additional capabilities added to the surrogate model. With an increase in the dataset, it took around 15 hours to complete the training process of the surrogate model. Each of the three encoder-decoder networks took around 5 hours and the multi-output regression model took 18 minutes to train between the encoded state vectors of input and output.

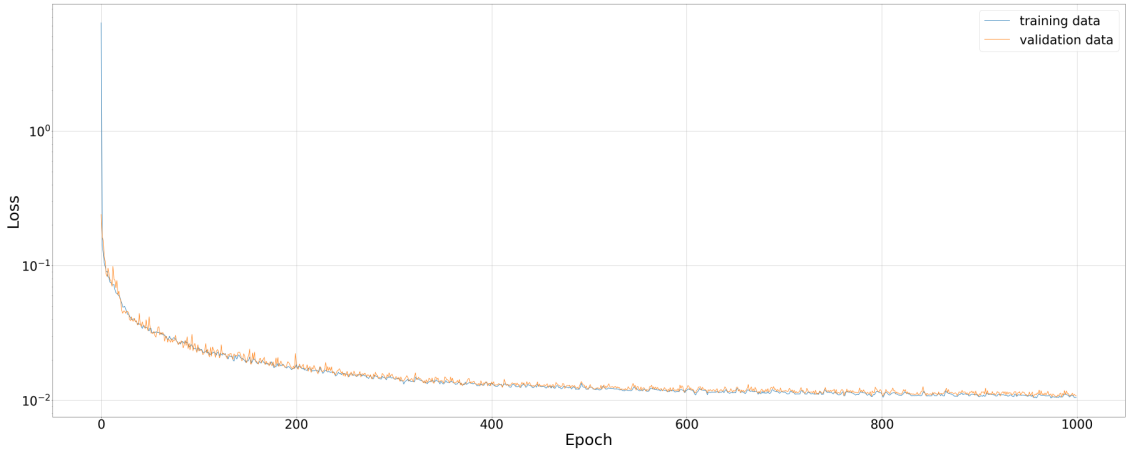


Figure 4.7: Training loss for input autoencoder.

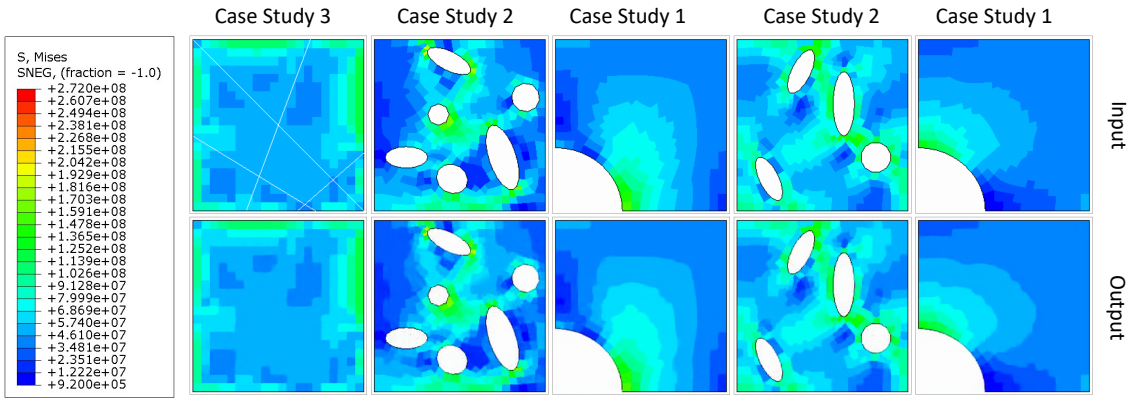


Figure 4.8: Results from input autoencoder on the training dataset.

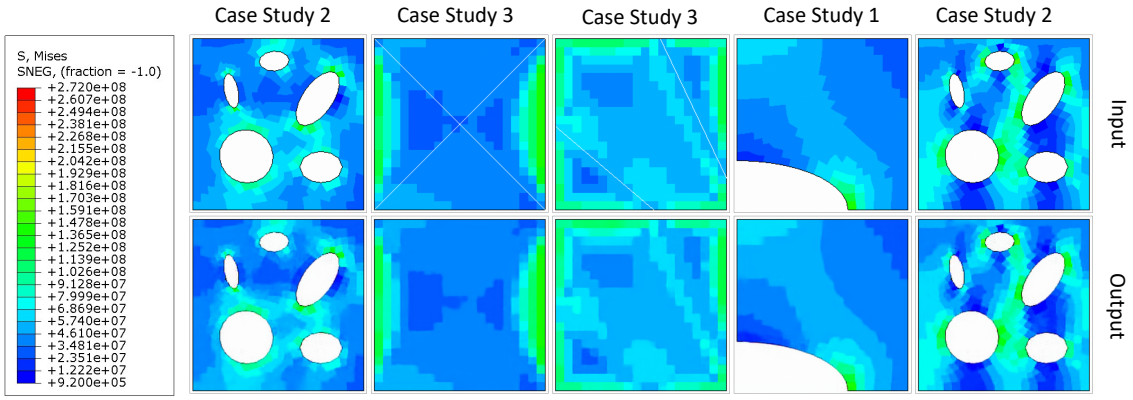


Figure 4.9: Results from input autoencoder on the testing dataset.

We can see that the encoder-decoder neural network architecture is able to reproduce the input images as the encoder-decoder's output, thus, we conclude that our auto encoder model is working for input stress images with good accuracy. We can see how well the Input Auto Encoder is trained from Figures, 4.8 and 4.9. The von Mises stress shows as the inputs in the first row match exactly like the output in the second row. These training results are developed using the generalized datasets of 3 case studies and presented for both training testing datasets. Figure 4.7 shows the trend of training and validation loss as it reaches to error loss percentage of order $10^{-}2$. Similarly Figures 4.10, 4.11, and 4.12 shows training and validation loss curve, training results for training and testing dataset respectively for

MISESERI output auto encoder and Figures 4.13, 4.14, and 4.15 for ENDENERI output auto encoder.

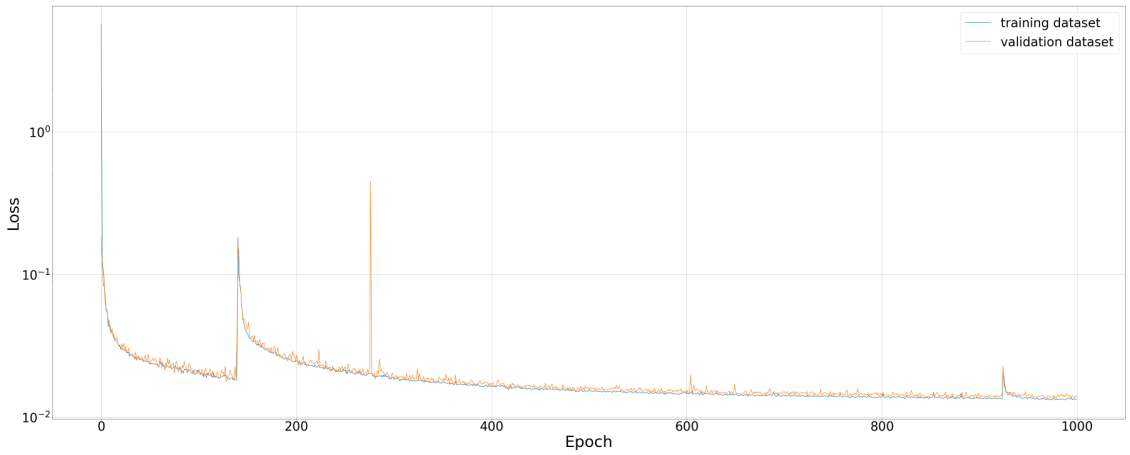


Figure 4.10: Training loss for MISESERI output autoencoder.

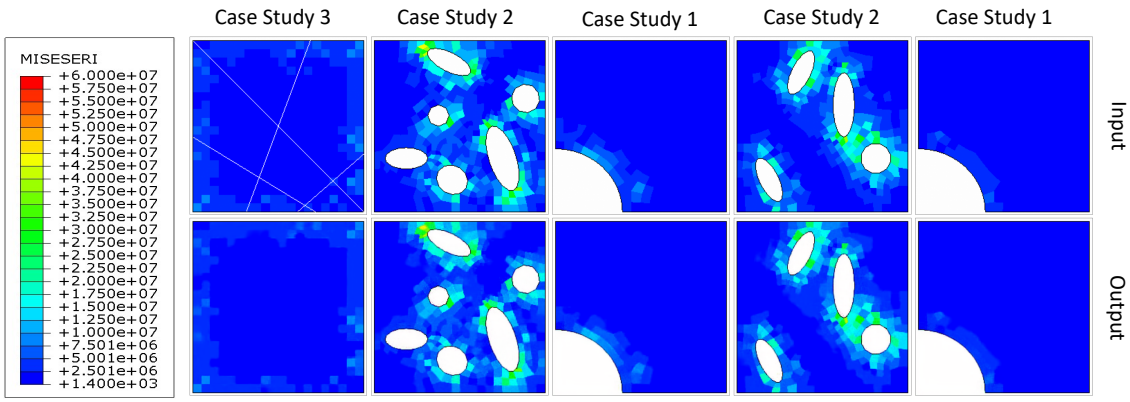


Figure 4.11: Results from MISESERI output autoencoder on the training dataset.

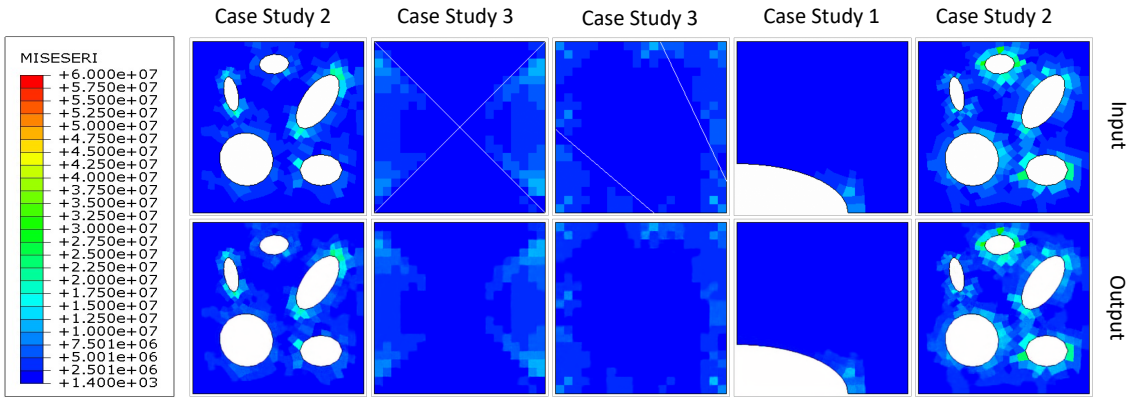


Figure 4.12: Results from ENDENERI output autoencoder on the testing dataset.

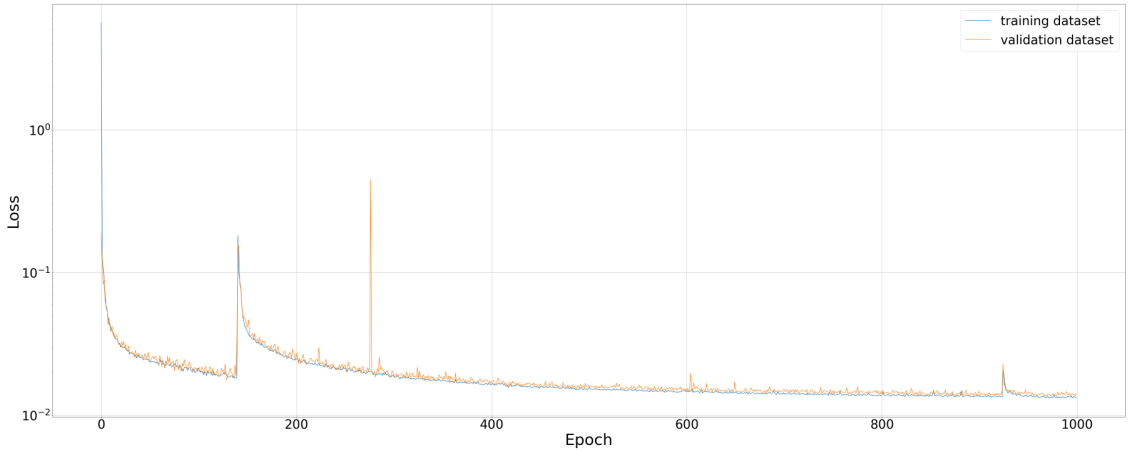


Figure 4.13: Training loss for ENDENERI output autoencoder.

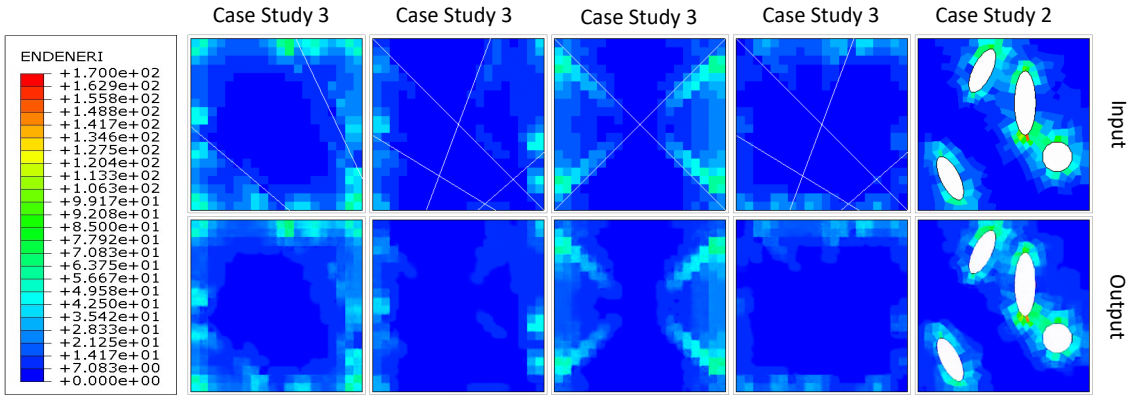


Figure 4.14: Results from ENDENERI output autoencoder on the training dataset.

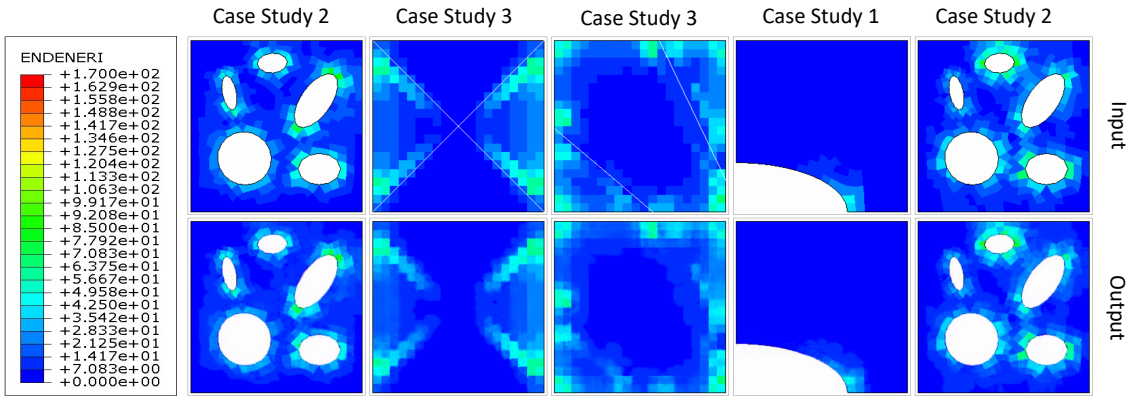


Figure 4.15: Results from ENDENERI output autoencoder on the testing dataset.

So far we have discussed the training performance on the individual encoder-decoder networks. But to see the prediction accuracy of the surrogate model, we combined the input encoder with the output decoder network using multi-output regression network. From the Figure, 4.16, we can see three rows, first row shows the von Mises stress as the input images, second row shows the ground truth images for MISESERI images and third shows the predicted MISESERI image developed using the surrogate model. Similarly, Figure 4.17 shows the results for element energy density error indicator (ENDENERI) from the final surrogate model.

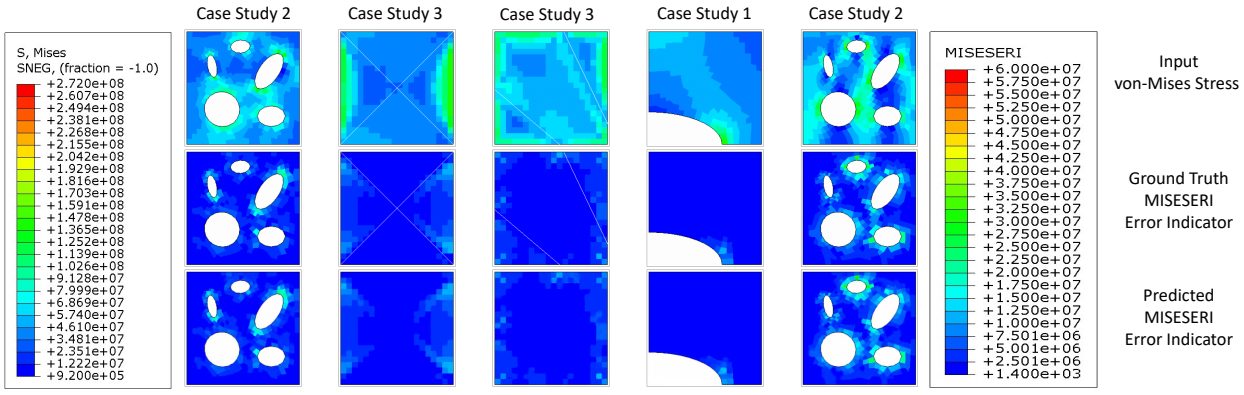


Figure 4.16: Final results from surrogate model predicting MIESERI error indicator using von Mises stress images.

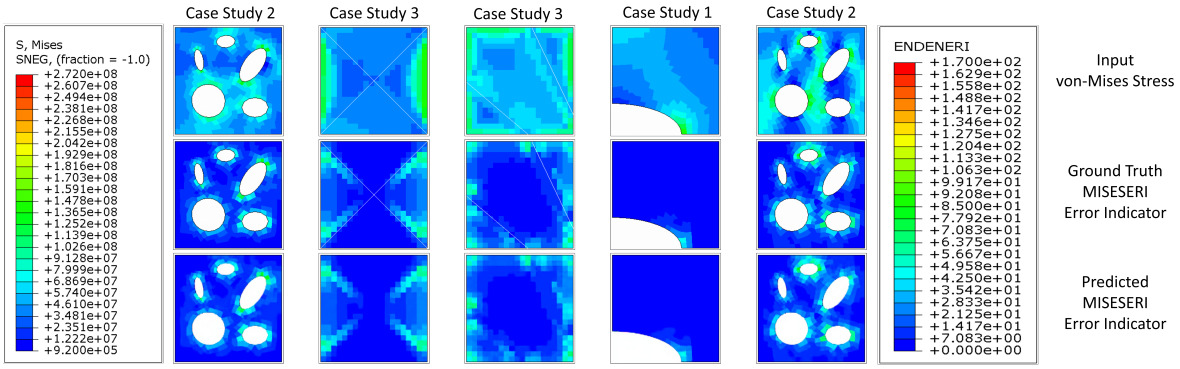


Figure 4.17: Final results from surrogate model predicting ENDENERI error indicator using von Mises stress images.

Even when the surrogate model was given with von Mises stress images with coarse mesh, the deep learning model was able to predict the appropriate error indicator as can be seen in Figure 4.18. By coarsening the mesh resolution by a factor of 1.5, the 256 $\times$ 256 image resolution is still able to capture the new mesh details perfectly and also the developed surrogate model is able to generate the error indicator results with the new stress contour information that resulted from this mesh coarsening. Another test was done with a completely new model geometry that incorporates all the three case studies to test the generalization of the surrogate model. It was found that the current version of the neural network architecture still needs some improvements for regenerating the input geometry and predicting the appropriate error indicator for a completely new dataset. There can be multiple steps that need to be taken for improving the efficiency of the surrogate model, for instance, data augmentation is a step to improve the overall generalization. A few more measures can be considered to enhance the training results of our surrogate model: (i) increasing the depth of CNN network architecture, (ii) improving the CNN training by optimizing the hyperparameters for both input and output autoencoders, and (iii) adding more hidden layers in the regression network used in the bottleneck region of surrogate model assembly woulid result in increasing the prediction accuracy.i

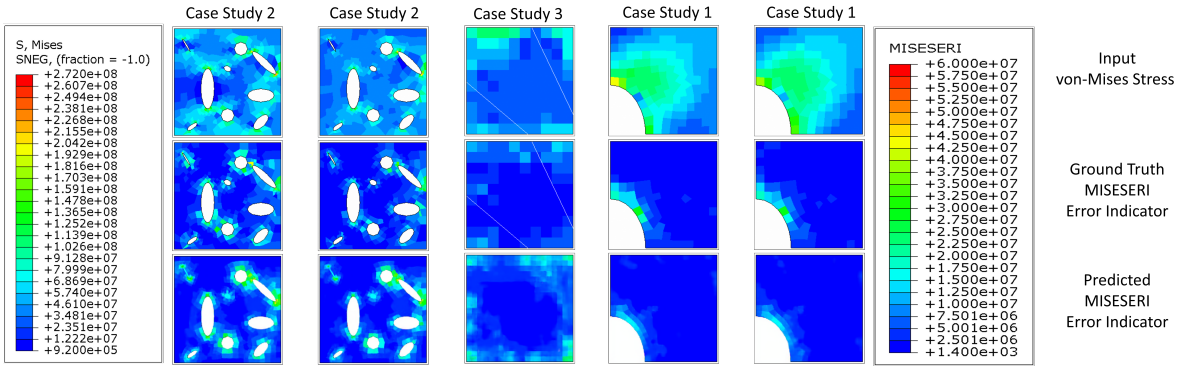


Figure 4.18: Surrogate model results with coarse mesh input von Mises stress images

Chapter 5

Conclusions

We have introduced an image processing based surrogate model for automatically predicting error for a given FEM solution using well trained neural network. This developed framework can be seen as a building block toward developing an adaptive re-meshing scheme, which would be fully automated, and with sufficient training, it can handle a broad range of FEM based models and geometries for structural analysis. The motivation is to develop a faster tool than what Abaqus/CAE provides for larger problems. With Deep CNNs, it is possible to perform re-meshing without any iterative step and predefined rules. This tool is developed using Deep CNN that takes von Mises stress images as input and von Mises error indicator (MISESERI) and energy density error indicator (ENDENERI) as outputs. Error indicators are output variables that provide an approximate measure of error in the requested base solution. They are computed on the basis of spatial gradient distribution and predicting the regions of smoothed or unsmoothed areas as low or high error locations. With Abaqus Python scripting capabilities it was very convenient to generate datasets as per the training and testing requirements for training the deep learning model using high-performance computers. While generating the data we used the quilt feature available in Abaqus/CAE for displaying the contours for the output variable. This allowed us to use single color per element helping us to hold the mesh discretization within the stress and error images. Our surrogate model has a joint architecture of encoder-decoder neural network with a fully-connected regression network. The encoder part of the architecture performs feature

extraction on both input and output images, resulting in optimizing training weights for both types of images. The fully connected regression model creates a mapping function between the encoded state of both input and output. The final stage is the decoder network which translates the encodings back to the image. To avoid mixing of training weights of both encoder and decoder networks with each other, two separate autoencoders were designed and trained.

During the Phase I, we evaluated our approach on a single dataset and found good accuracy in the results, the ground truth and predicted error indicator images look exactly the same. The entire training is done with a total of 1000 images of which 80% are used as training images, 10% as validation, and 10% as testing datasets. Next final training and testing of the surrogate model are done during the Phase II. Here, the global dataset is a collection of four different geometries each obtained by varying the hole sizes and shape. This global dataset consists of 2000 images each for stress and two types of error images (MISESERI and ENDENERI). Obtaining good results from these global datasets shows that the network holds generalization capabilities. Further to extend the capabilities, in addition to the quarter plate with a hole model, we developed two more case studies to test the current surrogate model. Phase III results showed that the current depth of the trained network is sufficient to work with a 256×256 image size having 3 channels with good accuracy. Using more datasets and training, this framework can be used to build an adaptive remeshing scheme using DNN. The present work is thus a good first step towards developing such an image-processing based, adaptive remeshing scheme.

Bibliography

[1] Martín Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S. Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Ian Goodfellow, Andrew Harp, Geoffrey Irving, Michael Isard, Yangqing Jia, Rafal Jozefowicz, Lukasz Kaiser, Manjunath Kudlur, Josh Levenberg, Dandelion Mané, Rajat Monga, Sherry Moore, Derek Murray, Chris Olah, Mike Schuster, Jonathon Shlens, Benoit Steiner, Ilya Sutskever, Kunal Talwar, Paul Tucker, Vincent Vanhoucke, Vijay Vasudevan, Fernanda Viégas, Oriol Vinyals, Pete Warden, Martin Wattenberg, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng. TensorFlow: Large-scale machine learning on heterogeneous systems, 2015. URL https://www.tensorflow.org/. Software available from tensorflow.org.

[2] Mohd Ahmed and Devender Singh. An adaptive parametric study on mesh refinement during adaptive finite element simulation of sheet forming operations. *Turkish Journal of Engineering and Environmental Sciences*, 32(3):163–175, 2008.

[3] G. Bebis and M. Georgiopoulos. Feed-forward neural networks. *IEEE Potentials*, 13 (4):27–31, 1994. doi: 10.1109/45.329294.

[4] Yoshua Bengio, Aaron Courville, and Pascal Vincent. Representation learning: A review and new perspectives. *IEEE transactions on pattern analysis and machine intelligence*, 35(8):1798–1828, 2013.

[5] Raouf Boutaba, Mohammad A Salahuddin, Noura Limam, Sara Ayoubi, Nashid Shahriar, Felipe Estrada-Solano, and Oscar M Caicedo. A comprehensive survey on

machine learning for networking: evolution, applications and research opportunities. *Journal of Internet Services and Applications*, 9(1):1–99, 2018.

[6] Guodong Chen and Krzysztof Fidkowski. Output-based error estimation and mesh adaptation using convolutional neural networks: Application to a scalar advection-diffusion problem. In *AIAA Scitech 2020 Forum*, page 1143, 2020.

[7] Heng-Tze Cheng, Levent Koc, Jeremiah Harmsen, Tal Shaked, Tushar Chandra, Hrishi Aradhye, Glen Anderson, Greg Corrado, Wei Chai, Mustafa Ispir, et al. Wide & deep learning for recommender systems. In *Proceedings of the 1st workshop on deep learning for recommender systems*, pages 7–10, 2016.

[8] François Chollet et al. Keras. https://keras.io, 2015.

[9] HH Dannelongue and PA Tanguy. Efficient data structures for adaptive remeshing with the fem. *Journal of Computational Physics*, 91(1):94–109, 1990.

[10] Jiuxiang Gu, Zhenhua Wang, Jason Kuen, Lianyang Ma, Amir Shahroudy, Bing Shuai, Ting Liu, Xingxing Wang, Gang Wang, Jianfei Cai, et al. Recent advances in convolutional neural networks. *Pattern Recognition*, 77:354–377, 2018.

[11] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016.

[12] Nal Kalchbrenner and Phil Blunsom. Recurrent continuous translation models. In *EMNLP*, 2013.

[13] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

[14] Patrick Knupp. Remarks on mesh quality. Technical report, Sandia National Lab.(SNL-NM), Albuquerque, NM (United States), 2007.

[15] Sotiris B Kotsiantis, Dimitris Kanellopoulos, and Panagiotis E Pintelas. Data preprocessing for supervised leaning. *International Journal of Computer Science*, 1(2):111–117, 2006.

[16] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. *Advances in neural information processing systems*, 25:1097–1105, 2012.

[17] David W Levy, Thomas Zickuhr, John Vassberg, Shreekant Agrawal, Richard A Wahls, Shahyar Pirzadeh, and Michael J Hemsch. Data summary from the first aiaa computational fluid dynamics drag prediction workshop. *Journal of Aircraft*, 40(5):875–882, 2003.

[18] Wing Kam Liu, Shaofan Li, and Harold Park. Eighty years of the finite element method: Birth, evolution, and future. *arXiv preprint arXiv:2107.04960*, 2021.

[19] Zeliang Liu, MA Bessa, and Wing Kam Liu. Self-consistent clustering analysis: an efficient multi-scale scheme for inelastic heterogeneous materials. *Computer Methods in Applied Mechanics and Engineering*, 306:319–341, 2016.

[20] W Lowrie, VS Lukin, and Uri Shumlak. A priori mesh quality metric error analysis applied to a high-order finite element method. *Journal of Computational Physics*, 230 (14):5564–5586, 2011.

[21] Andrew L Maas, Awni Y Hannun, Andrew Y Ng, et al. Rectifier nonlinearities improve neural network acoustic models. In *Proc. icml*, volume 30, page 3. Citeseer, 2013.

[22] Zhenguo Nie, Haoliang Jiang, and Levent Burak Kara. Stress field prediction in cantilevered structures using convolutional neural networks. *Journal of Computing and Information Science in Engineering*, 20(1):011002, 2020.

[23] R.R. Selmic and F.L. Lewis. Neural-network approximation of piecewise continuous functions: application to friction compensation. *IEEE Transactions on Neural Networks*, 13(3):745–751, 2002. doi: 10.1109/TNN.2002.1000141.

[24] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*, 2014.

[25] Michael Smith. *ABAQUS/Standard User's Manual, Version 6.9*. Dassault Systèmes Simulia Corp, United States, 2009.

[26] Daniel Strigl, Klaus Kofler, and Stefan Podlipnig. Performance and scalability of gpu-based convolutional neural networks. In *2010 18th Euromicro Conference on Parallel, Distributed and Network-based Processing*, pages 317–324, 2010. doi: 10.1109/PDP.2010.43.

[27] Ilya Sutskever, Oriol Vinyals, and Quoc V Le. Sequence to sequence learning with neural networks. In *Advances in neural information processing systems*, pages 3104–3112, 2014.

[28] Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, and Andrew Rabinovich. Going deeper with convolutions. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 1–9, 2015.

[29] Ali Yeilaghi Tamijani and Rakesh K Kapania. Buckling and static analysis of curvilinearly stiffened plates using mesh-free method. *AIAA journal*, 48(12):2739–2751, 2010.

[30] Matthew D Zeiler, M Ranzato, Rajat Monga, Min Mao, Kun Yang, Quoc Viet Le, Patrick Nguyen, Alan Senior, Vincent Vanhoucke, Jeffrey Dean, et al. On rectified linear units for speech processing. In *2013 IEEE International Conference on Acoustics, Speech and Signal Processing*, pages 3517–3521. IEEE, 2013.

[31] Min Zhang, Linpeng Li, Hai Wang, Yan Liu, Hongbo Qin, and Wei Zhao. Optimized compression for implementing convolutional neural networks on fpga. *Electronics*, 8(3), 2019. ISSN 2079-9292. URL https://www.mdpi.com/2079-9292/8/3/295.

[32] Wei Zhao and Rakesh K Kapania. Buckling analysis of unitized curvilinearly stiffened composite panels. *Composite Structures*, 135:365–382, 2016.