Copyright © 2018 Tech Science Press CMES, vol.117, no.3, pp.287-307, 2018

A Deep Learning-Based Computational Algorithm for Identifying Damage Load Condition: An Artificial Intelligence Inverse Problem Solution for Failure Analysis

Shaofei Ren[1,2] , Guorong Chen[2], Tiange Li[2], Qijun Chen[2] and Shaofan Li[2,*]

Abstract: In this work, we have developed a novel machine (deep) learning computational framework to determine and identify damage loading parameters (conditions) for structures and materials based on the permanent or residual plastic deformation distribution or damage state of the structure. We have shown that the developed machine learning algorithm can accurately and (practically) uniquely identify both prior static as well as impact loading conditions in an inverse manner, based on the residual plastic strain and plastic deformation as forensic signatures. The paper presents the detailed machine learning algorithm, data acquisition and learning processes, and validation/verification examples. This development may have significant impacts on forensic material analysis and structure failure analysis, and it provides a powerful tool for material and structure forensic diagnosis, determination, and identification of damage loading conditions in accidental failure events, such as car crashes and infrastructure or building structure collapses.

Keywords: Artificial intelligence (AI), deep learning, forensic materials engineering, plastic deformation, structural failure analysis.

1 Introduction

Most engineering materials, products and structures are designed, manufactured or constructed with an intention to function properly. However, they can fail, get damaged or may not operate or function as intended due to various reasons including material or design flaws, extreme loading, etc. It is important to identify the reasons of these failures or damage situations to improve the designs and detect any flaws in the materials or designs. One of the essential requirements of identifying the reasons of these failures is to know the loading conditions that lead to the failures.

Engineering products and structures are designed with specific intent and function. They may fail due to reasons including material shortcomings, construction flaws, extreme loading, or other conditions and behaviors exceeding their design parameters. To improve design and prevent failures, it is important to analyze actual failures, and identify the

[1] College of Shipbuilding Engineering, Harbin Engineering University, Harbin, China.
[2] Department of Civil and Environmental Engineering, University of California, Berkeley, CA, USA.
[*] Corresponding Author: Shaofan Li. Email: shaofan@berkeley.edu.

associated loading conditions.
Unfortunately, these loading conditions are not readily known while the forensic signatures such as the plastic strains or plastic deformations are easily measurable. For example, in car crashes, the impact loads on cars are not known, while the permanent deformations can be quantified after the fact. If the impact loads can be determined, it could potentially help insurance companies determine which party is responsible for the accident, and help car manufactures develop more realistic crash test scenarios. Both of these have high potential for concrete and significant economic impact.
Particularly for the situation of car crushes, it is very important to know the impact loads for two reasons. First, this will help greatly to determine which party is mainly responsible for the accident. Second, accidents are real crush tests. If the accident data can be added to the crush test data, it can help enhance car designs substantially.
What emerges from these considerations is an inverse problem of finding loading conditions from engineering responses. This represents an inverse of current engineering practice, in which the typical setup is to develop finite element models of structures, subject them to static and dynamic loading conditions, and then compute the resulting strains and residual displacements.
As a general methodology, we believed that machine learning (ML) and artificial intelligence (AI) techniques provide an effective solution for inverse problems. ML and AI encompass powerful tools for extracting complicated relationship between input and output sampling data, potentially through a training process, and then using the uncovered relationship to make predictions [Hastie, Tibshirani and Friedman (2009); Nasrabadi (2007)]. ML and AI have found a large number of successful applications in various fields beyond their birthplace in computer science[Sebastiani (2002); Bratko, Cormack, Filipič et al. (2006); Sajda (2006)]. Recent years have also witnessed a number of studies devoted to applying ML techniques to explore forensic materials engineering problems [Jones, Keatley, Goulermas et al. (2018); Mena (2016)]. In the context of this paper, the measurable engineering responses would be fed as input, with the loading conditions as output. The inverse nature of the problem does not hinder ML and AI's effectiveness in discovering complex mathematical relationships between input and output.
This paper develops a novel machine learning computational framework to determine and identify damage loading conditions for structures and materials based on their permanent or residual plastic deformation or damage state. Our work combines the current mature state of finite element models with the recent advances in machine learning methodologies. This approach advances the state of the art in forensic materials engineering, which seeks to examine material evidence and determine the original causes [Lei, Liu, Du et al. (2019); Zheng, Zheng and Zhang (2018); Zhou, Tang, Liu et al. (2018); Kirchdoerfer and Ortiz (2018)]. We believe the ML based approach can solve many previously intractable problems, with prior approaches incurring impractical computational costs due to the scale of the finite element models, the large degrees of freedom, and the complex and dynamic nature of the loading forces.
The rest of the paper begins with a detailed description of the machine learning algorithm

used. We then outline our process for gathering the required data to train the machine learning algorithms. This is followed by examples that demonstrate how we solve the inverse problem, including a cantilever beam of inelastic materials statically loaded at different locations, and the same beam loaded dynamically with impact loading. We seek to demonstrate with these examples that the machine learning algorithms can accurately identify both static loading and impact loading conditions based on observed residual plastic strain or deformation.

2 Methods

2.1 Deep neural network model

Recently, machine learning, especially deep neural networks, has become one of the most popular key words in every scientific or engineering field. Deep learning architectures, such as deep neural networks, deep belief networks and recurrent neural networks, have been applied to fields including computer vision, speech recognition, natural language processing, audio recognition, social network filtering, machine translation, bioinformatics, drug design and board game programs, where they have produced results comparable to and in some cases superior to human experts, e.g. [Ciresan, Meier and Schmidhuber (2012); Krizhevsky, Sutskever and Hinton (2012)].

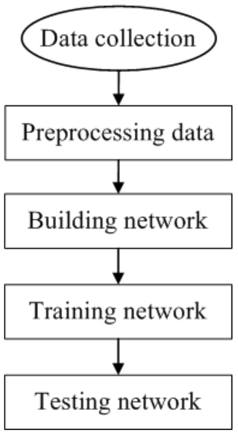


Figure 1: Flowchart of developing the machine learning neural network

Using machine learning methods, one could make relatively accurate predictions on some problems that were difficult to be solved before. In this work, we adopted a fundamental model of deep neural network (DNN), namely an artificial neural network. There are five stages in our model, as shown in Fig. 1 .
(1) Data collection: Collect data from software Abaqus.
(2) Data cleaning and processing with feature selection and feature engineering. During this stage, we applied dimension reduction, removed irrelevant data, and created new features

so the model could perform better.
The original data of static has 120 variables, which indicate 60 pairs of displace from x and y directions. The 60 pairs of data represent three parts of the cantilever beam which are the top, middle, and bottom parts. Due to the similarities, the top and bottom data was eliminated and we only keep the middle parts. Since the cantilever beam has a relative small displacement on x-direction, we only keep the y displacement to simplify the model. When doing the feature engineering, we tried to use our expertise and mathematical methods to create new relative features and established five new variables, including finding slope from plotting data of displacement x and displacement y, the summation of displacement y, the amplitude of displacement x, and the centroid distance. (First, we find a linear fit for the deformation change, namely, the approximate slope of the displacement x and displacement y from plotting data. Second, we generate the sum of displacement of y as another new feature. Third, to make the feature more obvious, we create a variable by using the top displacement x minus the bottom displacement x. Last but not least, we made centroid distance as a new feature. The equation is

$$\text{centroid distance} = \sqrt{(u_x - center)^2 + (u_y - center)^2)} \tag{1}$$

where ux is the displacement of x and uy is the displacement of y) For the dynamic data, we did a relative radical method. Like the static data we pick only 20 variables, instead of keep them we use the product of displacement x with displacement y, which lead all variables new. Then we applied summation of the variable and the centroid distance of the new dataset.
(3) Building network: Initialize bias, weight, number of layers and number of neurons in each layer.
(4) Application of deep neural network to obtain a specific mathematical model. It is noted that the basic mathematical model for the first stage of neural network is a simplified projection pursuit regression [Friedman and Stuetzle (1981)],

$$\hat{y_i} = \sum_{j=1}^{n} g_j(w_j^T x_i) \tag{2}$$

where g(x) is an activation function, w is a distributed weight, and x is an observation. During the second stage, the w is redistributed to optimize the loss or error used through back-propagation.
(5) Using a chosen test data set, which is different than the training sets, to validate the model and analyze errors.

2.2 Models and settings

As mentioned earlier, a deep neural network [LeCun, Bengio and Hinton (2015)] is used in this study to solve this inverse problem with procedures considered for the features. One sample fully-connected layer is shown in Fig. 2 [Castrounis (2016)]. In one certain neuron j, the employed mathematical model is described as

$$g_j(\beta_k^T X, \beta_0) = \sigma(\beta_0 X_0 + \beta_k^T X), \;\; \text{where } X_0 = 1 \tag{3}$$

One can clearly observe from Fig. 2 that, the input data flow(X) multiplied by the distributed weight(β_k) and then added a bias (β_0) as a parameter of an activation function ($\sigma(x)$), reflects the mathematical model in Eq. (3). Among all the options of activation functions, such as hyperbolic $tanh$, Sigmoid, or Relu. According to Ramachandran, "currently, the most successful and widely-used activation function is the Rectified Linear Unit (ReLU)." [Ramachandran, Zoph and Le (2018)] In our case, Rectified Linear Unit (ReLU) performs well as an activation function $\sigma(x)$ for both static and dynamic loading conditions. ReLU provides a nonlinear function $f(x) = max(0, x)$. Over-fitting is usually a concern that a model would perform too "well" on the training data set. In order to prevent over-fitting, a dropout was applied during the training process, "the key idea is to randomly drop units (along with their connections) from the neural network during training." [Srivastava, Hinton, Krizhevsky et al. (2014)].

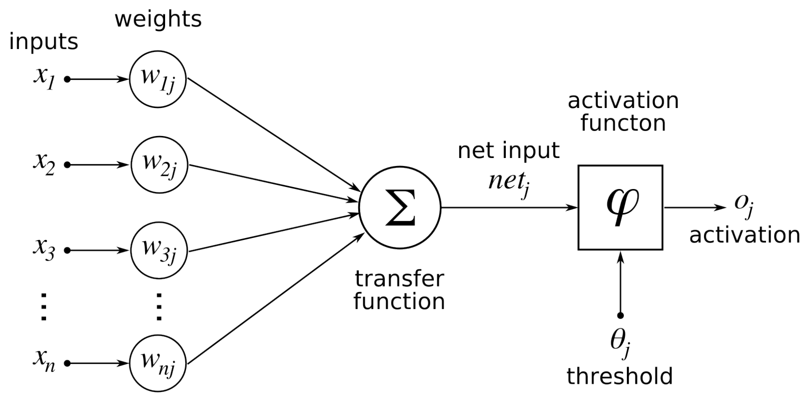


Figure 2: Illustration of neuron structure of the neural network

Furthermore, the used loss function is the Mean Square Error (MSE), which is defined as

$$MSE = \frac{1}{n}\sum_{i=1}^{n}(\hat{y}_i - y_i)^2 \tag{4}$$

MSE is commonly adopted as a loss function in statistical models. It provides an intuitive measurement of errors. The objective is to minimize MSE and make the model fit both the training data and validation data well. An optimizer will adjust between forward and backward propagation in order to achieve this objective. The optimization tool used herein to minimize MSE is the Adam optimizer [Kingma and Ba (2014)] with exponential descent in learning rate.

2.3 Implementation

TensorFlow is an open-source software library for dataflow programming across a range of tasks. It is a symbolic math library, and is also used for machine learning applications such as neural networks. In this project, we developed a machine learning computer code

by using TensorFlow.
There are four layers in the developed DNN code, with the learning rate of 0.0035, and the dropout of 0.05. We use the ReLu as the activation function, and we ran it in 18000 steps. In general, the number of hidden neurons should be between the size of the input layer and the size of the output layer. The input parameters of neural network is the plastic displacements or the permanent displacements along both the horizontal and vertical directions of all the nodes of the FEM mesh, except the boundary nodes, of the cantilever beam. The size of input layer is 60 or more after the numerical process, depending on the mesh of the model. The size of output layer is 3 or 8, depending on different numerical models. Thus, we choose from eight to sixty hidden layers in our machine learning code. After trying different numbers of hidden layers, we found that four hidden layers neural network provides good computation results for the FEM mesh size used and the numerical model. More hidden layers will lead to much more calculation time, but not much better results. Thus, we choose four hidden layers as the default structure for our machine learning test code. The number of hidden neurons should be less than twice the size of the input layer. Hence, we choose the first hidden layer 32 neurons, the second and third hidden layers have 64 neurons each. The final hidden layer has 8 neurons.

3 Data collection

3.1 Geometric and material properties

Table 1: Mechanical properties of AISI 4340 steel (33 HRc) (From [Guo and Yen (2004); Johnson and Cook (1985)]).

Density (kg/m^3)	7830
Young's modulus (GPa)	208
Poisson ratio	0.3
A(MPa)	792
B(MPa)	510
C	0.014
m	1.03
n	0.26
D_1	0.05
D_2	3.44
D_3	-2.12
D_4	0.002
D_5	0.61
$\dot{\epsilon}_0$(s^{-1})	1

A 2D cantilever beam was chosen for the study, which has a length of 5 meter and a width of

1 meter. The finite element model of the cantilever beam was developed with the ABAQUS software. Plane strain condition was assumed throughout this study, and CPE4R element was used, which is a 4-node bilinear plane strain quadrilateral element [Simulia (2011)]. In order to ensure the accuracy of the numerical calculation, different mesh sizes of the beam were chosen for the convergence analysis, and the final mesh size was chosen as 0.25 m. Accordingly, the number of nodes is 105, and the number of elements is 80.

We choose AISI 4340 steel (33 HRc) as the material of the cantilever beam, which is modeled by using the Johnson-Cook plasticity model ([Guo and Yen (2004); Johnson and Cook (1985)]). It is model as a thermo- elastoplastic solid, as expressed in the following equations,

$$\sigma = [A + B(\epsilon^p)^n][1 + C\ln(\frac{\dot{\epsilon}^p}{\dot{\epsilon}_0})][1 - (T^*)^m] \tag{5}$$

$$T^* = \frac{T - T_0}{T_m - T_0} \tag{6}$$

where σ is the flow stress, ϵ^p is the equivalent plastic strain, $\dot{\epsilon}^p$ is the strain rate, $\dot{\epsilon}_0$ is the reference strain rate, A, B, C, m, n are material constants, and T^* is the homologous temperature which is related to the absolute temperature T, the reference temperature T_0 and the melting temperature T_m.

The critical failure strain is defined as [Guo and Yen (2004); Johnson and Cook (1985)]:

$$\epsilon_f = [D_1 + D_2 e^{(D_3\sigma^*)}]\,[1 + D_4 \ln(\frac{\dot{\epsilon}^p}{\dot{\epsilon}_0})][1 + D_5 T^*] \tag{7}$$

where D_i are material constants, and σ^* is the dimensionless pressure-stress ratio.

Material parameters of AISI 4340 steel are listed in Tab. 1.

3.2 Boundary conditions and loads

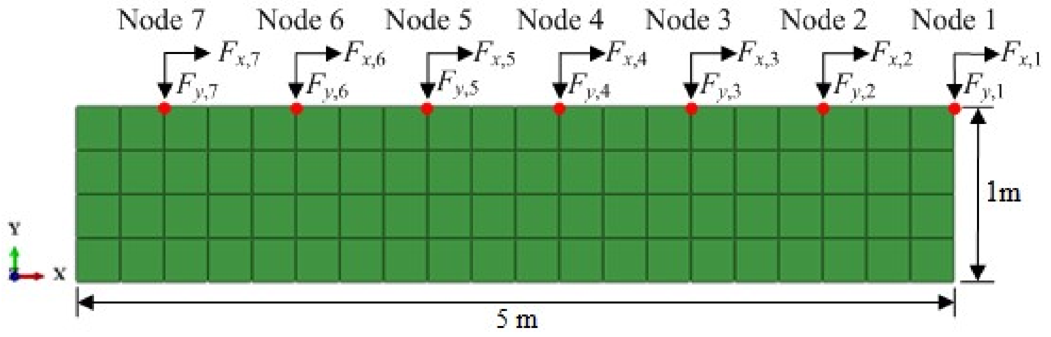


Figure 3: Finite element model of the cantilever beam and the loading positions

The finite element model of the cantilever beam is presented in Fig. 3. All degrees of freedom of the five nodes on the left at $x = 0$ m are rigidly fixed. As shown in this figure, seven numbered nodal points were chosen to apply loads and corresponding sets

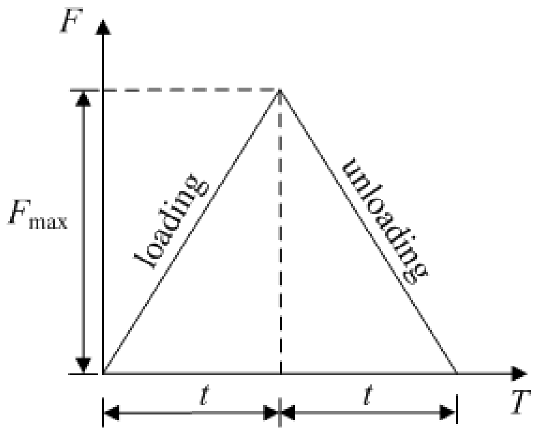


Figure 4: Dynamic loading time history

of simulation data were generated. In most cases when loads are applied to the three points closer to the support of the cantilever beam (nodes 5, 6 and 7 in Fig. 3), the residual displacement and plastic strain are all zero at all the nodes, which leads to the issue multiple-answer issue to DNN since zero residual displacement corresponds to three different locations. Thus, the loading at these points are excluded and only the loading points 1-4 are considered in the training, as shown in Fig. 3.
Both static and dynamic responses of the cantilever beam under concentrated loading forces are computed by using ABAQUS. It should be noted that the loading history of each concentrated force acting on the beam, which can cause plastic deformation of the beam, is featured with a bi-linear loading and unloading curve of bandwidth t and loading amplitude $F_{\max}$, as shown in Fig. 4.

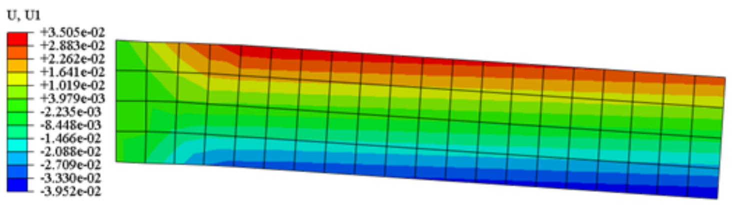
Four case studies have been conducted to test, validate, and verify the deep learning algorithm and the trained neural network:
(1) Prediction of static loads acting on the numbered nodes (also referred as training nodes in this paper). Static loads simultaneously imposed to one up to four numbered nodes shown in Fig. 3. Responses of the beam under static loads of different amplitudes were numerically predicted, and the database for the deep learning was developed. Then, a different deformation of the beam caused by loads applied to the numbered nodes was given by ABAQUS program, and the amplitude of the static load was predicted by the deep learning algorithm and compared with the exact solution.
(2) Prediction of static loads acting between the numbered nodes. In this case study, the database for the deep learning was also developed by applying static forces to the numbered nodes. Then, a deformation of the beam caused by a load acting on a node between the two adjacent numbered nodes (as shown in Fig. 3) was given by ABAQUS program. Amplitude and position of the static load were predicted by the deep learning algorithm.
(3) Prediction of the impact load acting on the numbered nodes. Impact loads imposed

Table 2: Loads for the static analysis

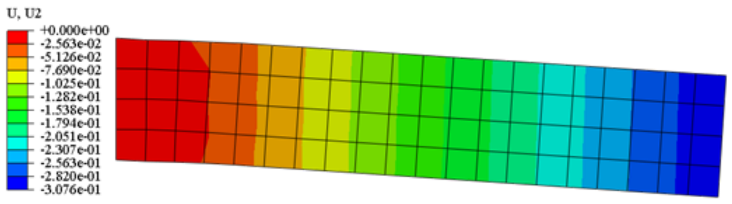
Node	F_{max} (N)	t
1	$5.0 \times 10^7 - 8.0 \times 10^7$	0.5
2	$6.0 \times 10^7 - 9.0 \times 10^7$	0.5
3	$8.0 \times 10^7 - 12.0 \times 10^7$	0.5
4	$8.0 \times 10^7 - 14.0 \times 10^7$	0.5

Table 3: Loads for the dynamic analysis

Node	F_{max} (N)	t (s)
1	$7.1 \times 10^7 - 7.3 \times 10^7$	0.01-0.03
2	$8.1 \times 10^7 - 9.3 \times 10^7$	0.01-0.03
3	$10.0 \times 10^7 - 11.8 \times 10^7$	0.01-0.03
4	$12.0 \times 10^7 - 14.8 \times 10^7$	0.01-0.03



(a)



(b)

Figure 5: Permanent displacement distribution: (a) Plastic displacement along horizontal direction, and (b) Plastic displacement along vertical direction

to one numbered node. Responses of the beam under impact load with different amplitudes and durations were numerically predicted. Then, a different deformation of the beam caused by an impact load acting on this numbered node was given, and both the amplitude and duration of the impact load were predicted.

(4) Prediction of the impact load acting between the numbered nodes. This case study is similar to the second case study. Position, amplitude and duration of the impact load were

all predicted by the deep learning algorithm.
The bi-linear pulse load–time history curve is assumed throughout this study as shown in Fig. 4. For the static analysis, the step time T equals 1, and durations of loading and unloading are 0.5. For the dynamic response, durations of loading and unloading are significantly reduced to orders of 0.01-0.03 s to simulate the impact loads. Meanwhile, in order to minimize the influence of inertial effects and get the final stable deformation of the beam, the step time T for the dynamic analysis is set as 50-100 s. For the static and dynamic analysis, amplitude, duration and step time for the first two case studies are listed in Tab. 2 and Tab. 3, respectively. For the multi-points condition, amplitude of the load is reduced. For static analysis as an example, amplitude of the load was between 2.0×10^7 - 7.0×10^7 N for two nodes cases, 1.0×10^7 - 5.0×10^7 N for three nodes cases, and 1.0×10^7- 4.0×10^7 N for four nodes cases.

3.3 Training data

When applying above magnitude force to the cantilever beam, it will have permanent plastic deformation. One of particular plastic displacement examples is as shown in Fig. 5, while the associated plastic residual strain distribution is shown in Fig. 6.
After obtaining the residual plastic displacements from all FEM nodes of the beam, we combine them together as the input of training data of DNN. The process is as shown in Fig. 7. The right table of Fig. 7 is the input of one set of raw training data of DNN. The output of this set of training data is the value and location of the outside force according to this plastic displacement distribution, for static modes, and the location, magnitude and duration according to this plastic displacement distribution, for dynamic models.

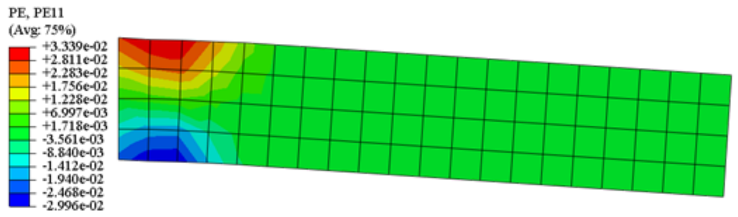
4 Results and discussions

4.1 Prediction of the static loads on training nodes

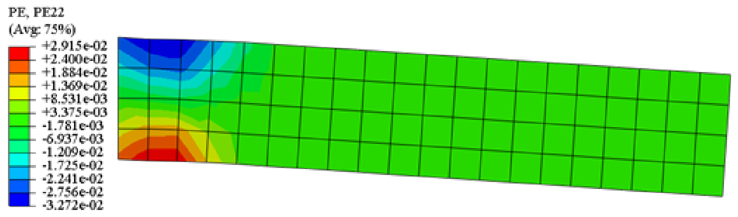
Table 4: The correct and predicted loads of the testing cases (10^7N)

Testing cases	**1**		**2**		**3**		**4**	
Combination	④		②④		①②④		①②③④	
	correct	predicted	correct	predicted	correct	predicted	correct	predicted
①x	0	-0.0353	0	-0.1221	2.1543	2.0785	1.2917	1.3145
①y	0	0.0214	0	-0.0524	-1.9067	-1.969	-1.5630	-1.5402
②x	0	-0.1143	4.5138	4.514	2.7296	2.8529	1.7999	1.8281
②y	0	-0.0335	-4.5320	-4.5488	-3.1083	-3.0752	-1.9380	-1.9136
③x	0	0.0294	0	-0.0004	0	0.0598	2.1204	2.1566
③y	0	-0.0469	0	0.0317	0	-0.0062	-2.7124	-2.8005
④x	9.4431	9.2816	6.1548	6.0149	3.4294	3.3453	2.8181	2.793
④y	-9.8902	-9.6926	-5.8801	-5.8765	-3.8756	-3.8447	-2.6192	-2.5966

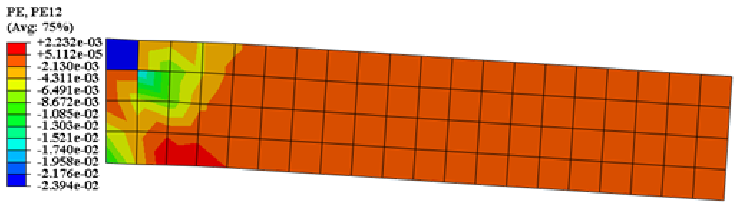
Generally speaking, in machine learning, the more the data are available, the more accurate a prediction may be. However, there is still no rule about how much data is enough. It



(a)



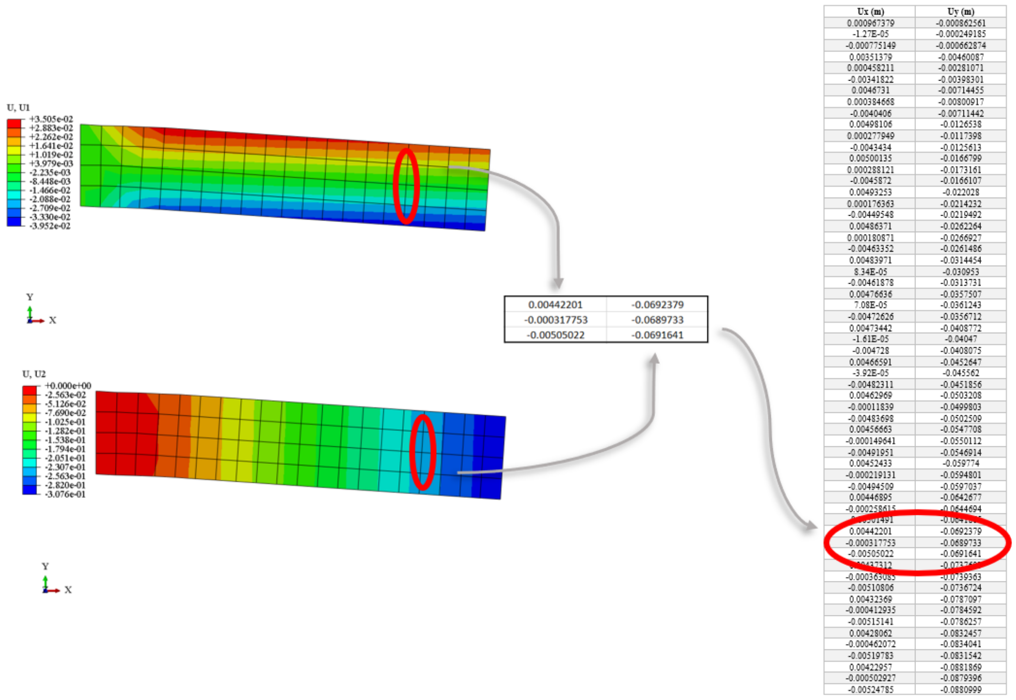
(b)



(c)

Figure 6: Permanent plastic strain distribution: (a) Plastic strain component ϵ^p_{11}, (b) Plastic strain component ϵ^p_{22}, and (c) Plastic strain component ϵ^p_{12}

depends on how complex of the problem and how complex of the learning algorithm are. So the rules we used to generate data is as follows,
1. Generating the preliminary database with a little samples.
2. Training the model to see how the performance of the model is and if the predicted accuracy hits the requirement.
3. If the results are not accurate enough, then generating more data to see if the performance increases.
4. If the performance increases, then repeat the 1 to 3 Steps until the result hits the required accuracy.
5. If the performance stays still or increases slowly, then modifying the learning model or learning parameters.
For our static problem, we have generated 290 sets of data (about 20 sets for each load case) from which we get a good training effect. The training effect is usually evaluated by the training loss which is a value representing the fitting of the model to the training data.



0.00442201	-0.0692379
-0.000317753	-0.0689733
-0.00505022	-0.0691641

Ux (m)	Uy (m)
0.000967379	-0.000862561
-1.27E-05	-0.000249185
-0.000775149	-0.000662874
0.00351379	-0.00460087
0.000458211	-0.00281071
-0.00341822	-0.00398301
0.0046731	-0.00714455
0.000384668	-0.00800917
-0.0040406	-0.00711442
0.00498106	-0.0126538
0.000277949	-0.0117398
-0.0043434	-0.0125613
0.00500135	-0.0166799
0.000288121	-0.0173161
-0.0045872	-0.0166107
0.00493253	-0.022028
0.000176363	-0.0214232
-0.00449548	-0.0219492
0.00486371	-0.0262264
0.000180871	-0.0266927
-0.00463352	-0.0261486
0.00483971	-0.0314454
8.34E-05	-0.030953
-0.00461878	-0.0313731
0.00476636	-0.0357507
7.08E-05	-0.0361243
-0.00472626	-0.0356712
0.00473442	-0.0408772
-1.61E-05	-0.04047
-0.004728	-0.0408075
0.00466591	-0.0452647
-3.92E-05	-0.045562
-0.00482311	-0.0451856
0.00462969	-0.0503208
-0.00011839	-0.0499803
-0.00483698	-0.0502509
0.00456663	-0.0547708
-0.000149641	-0.0550112
-0.00491951	-0.0546914
0.00452433	-0.059774
-0.000219131	-0.0594801
-0.00494509	-0.0597037
0.00446895	-0.0642677
-0.000258615	-0.0644694
[illegible].00501491	-0.064[illegible]
0.00442201	-0.0692379
-0.000317753	-0.0689733
-0.00505022	-0.0691641
[illegible]00437312	-0.073[illegible]
-0.000363085	-0.0739363
-0.00510806	-0.0736724
0.00432369	-0.0787097
-0.000412935	-0.0784592
-0.00515141	-0.0786257
0.00428062	-0.0832457
-0.000462072	-0.0834041
-0.00519783	-0.0831542
0.00422957	-0.0881869
-0.000502927	-0.0879396
-0.00524785	-0.0880999

Figure 7: Collect one set of training data input

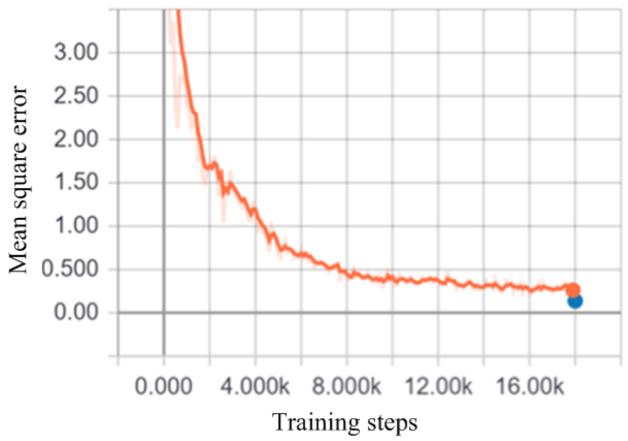


Figure 8: Training loss

In this study, the mean square error (MSE) between the outputs and the correct results was chosen as the training loss. There is an updated loss in each epoch which can show the fitting of the model. The record of the loss throughout the process of our case is shown as Fig. 8:

Table 5: Predicted errors of the testing cases (%)

Loading Cases	**1**	**2**	**3**	**4**
Combination	④	②④	①②④	①②③④
①x	null	null	3.5182	1.7654
①y	null	null	3.2667	1.4555
②x	null	0.0041	4.5163	1.5689
②y	null	0.3699	1.0641	1.2582
③x	null	null	null	1.7097
③y	null	null	null	3.246
④x	1.7104	2.2732	2.4529	0.8908
④y	1.9981	0.0602	0.797	0.8644

In Fig. 8, the loss gradually decreases and reaches the minimum value of 0.2611 after 18k steps. The minimum value and the smooth descend curve of the loss indicates that the model was trained steadily and has fitted the training data very well.
To test the performance of the model, four sets of testing data were generated. Each testing data represents one type of loading combination, as shown in Tab. 4. For this problem, we designed our output layers with eight neurons which represent the two direction loads in four valid loading points, as mentioned in Section 3.2. Therefore, inputting the testing data to our DNN model, we get the output as shown in Tab. 4. The errors of the prediction are listed in Tab. 5.
In Tab. 4 , due to the eight-neuron output layer, all the eight values of the output were non-zero, but the values on the real-loaded nodes were much larger than the values on the other nodes. For example, in Case 1, the real load is located in node ④. In the predicted result, the values of ④x and ④y is obviously much larger than the values on the other nodes. And the values of ④x and ④y is very close to the real values, with the errors of 1.71% and 1.998%. The output of the other three test cases have the same pattern. Predicted errors are all smaller than 5%, as shown in Tab. 5. Therefore, the trained DNN model can correctly predict the loading locations and the magnitude of the static loads which act on the training nodes.

4.2 Prediction of the static loads between training nodes

Table 6: Correct values of the testing data

Testing Cases	1	2	3	4	5	6	7	8
Interval	③∼④	③∼④	②∼③	②∼③	①∼②	①∼②	①∼②	①∼②
Location (m)	3.0	3.25	3.75	4.0	4.5	4.5	4.75	4.75
Load x (10^7N)	12	11.47	10	9.02	6	6.45	6.34	7.34
Load y (10^7N)	-12	-10.35	-10	-9.56	-6	-6.63	-6.76	-7.76

Table 7: Predicted results of the testing data

Testing Cases	1	2	3	4	5	6	7	8
Interval	③~④	③~④	②~③	②~③	①~②	①~②	①~②	①~②
Location (m)	2.922	3.037	3.82	4.131	4.723	4.543	4.835	4.618
Load x (10^7N)	12.214	11.163	9.327	8.356	5.675	6.389	5.886	7.167
Load y (10^7N)	-12.065	-10.808	-9.9	-9.383	-6.108	-6.707	-6.599	-7.877

Table 8: Predicted errors of the testing data (%)

Testing Cases	1	2	3	4	5	6	7	8
Interval	0	0	0	0	0	0	0	0
Location	2.604	6.554	1.88	3.281	4.949	0.96	1.785	2.769
Load x	1.784	2.68	6.729	7.366	5.418	0.814	7.159	2.361
Load y	0.545	4.424	1.002	1.85	1.792	1.164	2.377	1.505

In Section 4.1 we demonstrated the capability of the DNN model to predict the statics loads acting on the training nodes. In this section, following the data collection rules outlined in Section 4.1, we trained the DNN model with 133 sets of data that were obtained by static loads acting on the four valid training nodes individually. Then, we tested the model with 8 sets of deformation states caused by 8 different loads acting in the interval between the training nodes, to see if the DNN can make extended prediction. The testing data is shown in Tab. 6. We designed the output layer with three neurons which represent the location of theload and the magnitudes of the loads in x and y directions because the loads are all acting individually in this section. The predicted results are shown in Tab. 7 and Tab. 8.

According to the results, all the predicted locations were in the correct intervals. The correct interval is an important information which can indicate the location range of the load. Then one can further locate the load by subdividing or reducing the interval. For the prediction of the specific location and the magnitude in x and y directions, the maximum errors are 6.554%, 7.366% and 4.424% respectively. The average error of the output is 3.098%, which is less than 5%. The errors of the prediction are small and can meet a lots of engineering requirements. Therefore, the results show that the DNN model is able to predict the static loads in the interval between the training nodes.

4.3 Prediction of the impact loads on training nodes

Table 9: Predicted errors of the nodal impact loads(%)

Loading Cases	1	2	3	4
Location	3.321	0.94	3.175	0.474
Magnitude	0.289	0.293	1.641	2.338
Duration	12.321	5.032	0.682	3.535

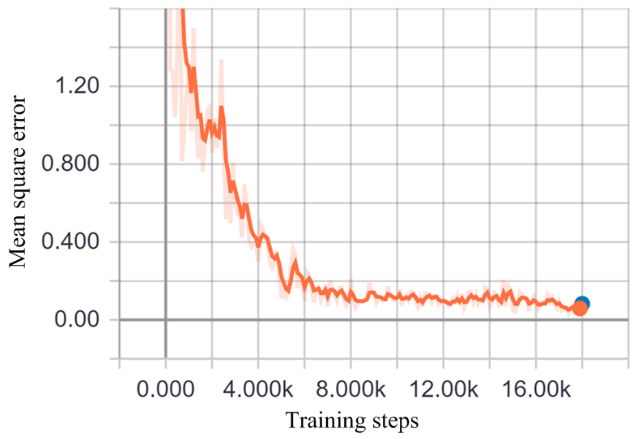


Figure 9: Training loss

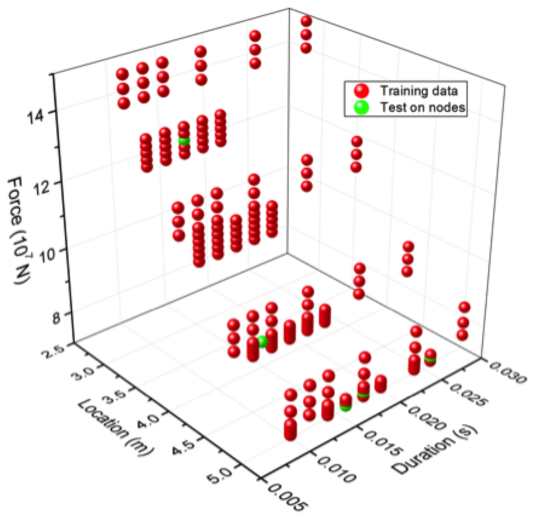


Figure 10: Training and testing data

Compared to the static problems, dynamic problems are more common in real situations but also more complicated. By following the data collection rules stated in Section 4.1, for the dynamic problem discussed here, about 40 sets of data for each load case would produce a good training effect. We finally trained the DNN model with 175 sets of deformation states caused by different single vertical impact load acting on the training points. Then five sets of test data were generated to test the performance of the trained model. All the training and testing data are shown in Fig. 10. The output layer has three neurons which represent the location, magnitude and the duration of the impact loads. The training loss throughout the training process is shown as Fig. 9. The minimum loss (MSE) reached 0.056 after 18k iteration steps which shows a good fitting of the model to

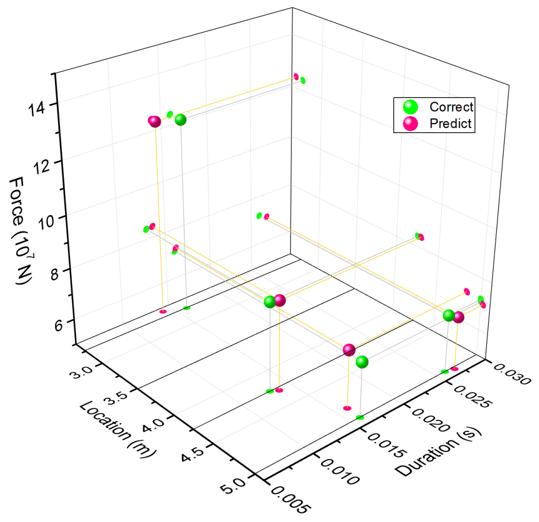


Figure 11: Predicted result of the loads on the nodes

the training data.The prediction of these impact loads are shown in Fig. 11. The predicted loads and the correct loads are very close to each other in the shown three dimension space. The maximum errors of the predicted location and the magnitude are 3.321% and 2.338%, respectively, as shown in Tab. 9. The maximum error of the duration is 12.321%. During the study, we found that the load duration is more difficult to be predicted than the other two parameters. The overall errors of these five testing cases are less than 5.4%. It shows that the DNN also works very well in the prediction of the impact loads which are located on the training nodes, especially for the location and the magnitude of the impact.

4.4 Prediction of the impact loads between training nodes

Table 10: Predicted errors of the loads in intervals(%)

Predicting Cases	1	2	3
Interval	③∼④	②∼③	①∼②
Location	3.058	9.718	3.418
Magnitude	2.047	6.859	1.195
Duration	7.213	8.516	1.233

For the prediction of the impact loads acting within different intervals, we generated three sets of testing deformation which were caused by three different impact loads acting in each interval individually. Then, we used the DNN model in the Section 4.3 which had been trained with the 175 sets of deformation by single impact loads acting on training nodes. The values of the training data and the test loads are shown in Fig. 12. The output

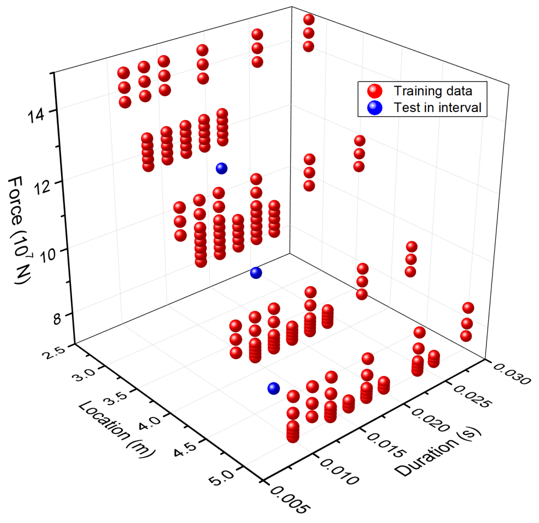


Figure 12: Training and testing data

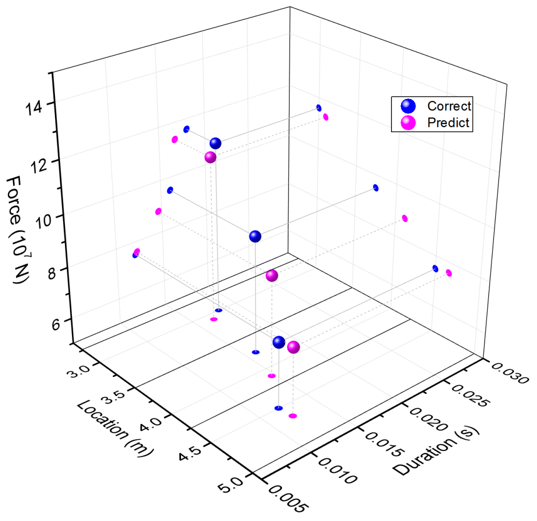


Figure 13: Predicted result of the loads in the intervals

of the DNN model is shown in Fig. 13, and the error of the output is listed in Tab. 10. The maximum error of the location, magnitude and duration are 9.718%, 6.869% and 7.213%, respectively. Compared to the Section 4.3, the overall errors are increased but still less than 10%. And the predicted locations are all in the correct intervals. It is concluded that the DNN can predict the loads in the interval between the training nodes, too.

The results show that the DNN model is capable to do interpolation between the training data by itself. However, the interpolation accuracy still depends on the density of the training data. The higher the density of the training data, the higher the predicted accuracy will be.

Thus, while training the DNN model, we can choose some typical loading cases as training data depending on the accuracy demand, which would save a lot of time in training phrase. To further reduce the predicted error, another approach is developed as explained in the Section 4.5.

4.5 Improving accuracy of prediction on the load location

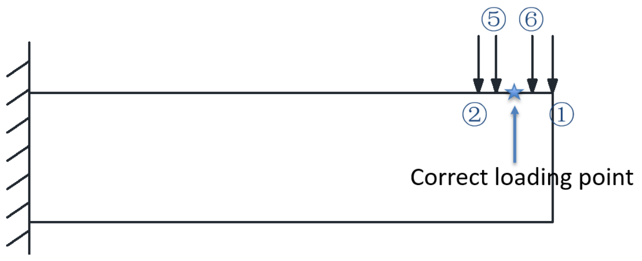


Figure 14: Refine interval

Table 11: Correct values of the predicted impact loads

Predicting Cases	1	2
Interval	⑤~⑥	⑤~⑥
Location (m)	4.5	4.5
Magnitude(10^7N)	8.2	8.4
Duration (s)	0.018	0.014

Table 12: Predicted value of the impact loads

Predicting Cases	1	2	1	2
	Before refine		After refine	
Interval	⑥~①	⑥~①	⑤~⑥	⑤~⑥
Location(m)	4.906	4.66	4.464	4.444
Magnitude(10^7N)	7.332	7.739	8.444	8.435
Duration(s)	0.0191	0.0146	0.0171	0.0145

According to the above results, it is shown that the DNN model can tell the correct interval in which the loads acts. Therefore, after the first prediction, we can concentrate our attention to the predicted interval. To further locate the true value of the load, we can subdivide the interval into several smaller intervals, and then add more training data within this area. Here, we took the Testing Case 3 in the Section 4.4 as an example. The DNN model had predicted that the load was located in interval ①~②. So we can subdivide the interval ①~② into three smaller intervals. The location coordinate of the smaller

Table 13: Predicted errors of the nodal impact loads(%)

Predicting Cases	1	2	1	2
	Before refine		After refine	
Interval	⑥∼①	⑥∼①	⑤∼⑥	⑤∼⑥
Location	9.02	3.552	0.798	1.238
Magnitude	10.583	7.866	2.978	0.421
Duration	5.945	4.236	4.864	3.725

intervals are 4.25∼4.375 m, 4.375∼4.625 m and 4.625∼5 m, as shown in Fig. 14. Eighteen sets of new training data were generated on each of the new numbered points ⑤ and ⑥, respectively. Then, we retrained the DNN model with the data on Node ①,②,⑤ and ⑥. Different from the previous model which was trained with all data, the retrained model would only predict the loads in interval ①∼② and achieve a much higher accuracy in the prediction. Two testing data were generated, as shown in Tab. 11. The predicted results and the errors before and after the retraining are shown in Tab. 12 and Tab. 13. The predicted errors of all three parameters are reduced greatly by refining the interval, from 9.02%, 10.583% and 5.945% to 0.798%, 2.978% and 4.864%, respectively. Also, the prediction of the smaller intervals reached 100% accuracy. Therefore, by refining the interval and retraining the DNN model step by step, it is possible to finally reach the required accuracy. Based on the convergence tendency, it is also shown that finer mesh will lead to correct finer intervals and high accuracy results. Thus, if we could have large amounts of data with forces applied on a large variety of different locations, which is equivalent to dividing the cantilever beam to large numbers of sections, the predicted results will still be in the correct interval. The accuracy will achieve almost 100% if we have unlimited amount of data.

5 Conclusions

In this study, we applied deep (machine) learning techniques to solve an inverse engineering problem of identification of the location, amplitude, and duration of impact forces on a structure, both static and dynamic loads, based on the permanent residual deformation as well as the permanent strain distributions. For static problems, the developed machine learning algorithm can predict both the location as well as the amplitude of the force with high accuracy. The prediction on the location of the applied load is not necessarily from the training data, after studying the training data, the machine learning algorithm can automatically use interpolation to find the applied load location that is not in the train data. This is to say that we only need to train the neural network with a limited number of loading sets, it can then predict any loading locations on the boundary of the cantilever beam. This is a remarkable success for an inverse problem solution that was impossible to realize because of the non-unique solutions for inverse problems. Based on this study, we have come to a conclusion that by using the forensic signatures such as permanent deformation and residual plastic

strain distributions one can uniquely (practically) determine the applied load conditions inversely with high accuracy for most engineering purposes.
For dynamic loading problems, the current version of our machine learning algorithms can also predict both the location and amplitude of applied loads with high accuracy, whereas the accuracy of the prediction on the duration of the loads is not high, even though it still have good accuracy i.e. the error is within 5% to 10%. These results are very much promising for both static and dynamic inverse problems. Results herein demonstrate that the ML or AI based approaches can solve many previously intractable problems. We shall soon report our machine learning algorithm for inverse failure analysis of three-dimensional structures with more complex geometries and loading conditions, including crashworthiness analysis and seismic or other extreme hazardous event induced structure failure forensic analysis.

References

Bratko, A.; Cormack, G. V.; Filipič, B.; Lynam, T. R.; Zupan, B. (2006): Spam filtering using statistical data compression models. *Journal of Machine Learning Research*, vol. 7, no. Dec, pp. 2673-2698.

Castrounis, A. (2016): Artificial intelligence, deep learning, and neural networks explained. https://www.kdnuggets.com/2016/10/artificial-intelligence-deep-learningneural-networks-explained.html.

Ciresan, D.; Meier, U.; Schmidhuber, J. (2012): Multi-column deep neural networks for image classification. In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, Providence, RI,* pp. 3642-3649.

Friedman, J. H.; Stuetzle, W. (1981): Projection pursuit regression. *Journal of the American statistical Association*, vol. 76, no. 376, pp. 817-823.

Guo, Y.; Yen, D. W. (2004): A fem study on mechanisms of discontinuous chip formation in hard machining. *Journal of Materials Processing Technology*, vol. 155, pp. 1350-1356.

Hastie, T.; Tibshirani, R.; Friedman, J. (2009): Unsupervised learning. *Elements of Statistical Learning*, pp. 485-585.

Johnson, G. R.; Cook, W. H. (1985): Fracture characteristics of three metals subjected to various strains, strain rates, temperatures and pressures. *Engineering Fracture Mechanics*, vol. 21, no. 1, pp. 31-48.

Jones, A.; Keatley, A.; Goulermas, J.; Scott, T.; Turner, P.; Awbery, R.; Stapleton, M. (2018): Machine learning techniques to repurpose uranium ore concentrate (uoc) industrial records and their application to nuclear forensic investigation. *Applied Geochemistry*, vol. 91, pp. 221-227.

Kingma, D. P.; Ba, J. (2014): Adam: A method for stochastic optimization. Preprint arXiv:1412.6980.

Kirchdoerfer, T.; Ortiz, M. (2018): Data-driven computing in dynamics. *International Journal for Numerical Methods in Engineering*, vol. 113, no. 11, pp. 1697-1710.

Krizhevsky, A.; Sutskever, I.; Hinton, G. E. (2012): Imagenet classification with deep convolutional neural networks. *Advances in Neural Information Processing Systems*, pp. 1097-1105.

LeCun, Y.; Bengio, Y.; Hinton, G. (2015): Deep learning. *Nature*, vol. 521, pp. 436-444.

Lei, X.; Liu, C.; Du, Z.; Zhang, W.; Guo, X. (2019): Machine learning-driven real-time topology optimization under moving morphable component-based framework. *Journal of Applied Mechanics*, vol. 81, 011004.

Mena, J. (2016): *Machine Learning Forensics for Law Enforcement, Security, and Intelligence*. Auerbach Publications, Boca Raton, Florida, USA.

Nasrabadi, N. M. (2007): Pattern recognition and machine learning. *Journal of Electronic Imaging*, vol. 16, no. 4, 049901.

Ramachandran, P.; Zoph, B.; Le, Q. V. (2018): Searching for activation functions. In: *Proceedings of the International Conference on Learning Representations (ICLR)*, Workshop track.

Sajda, P. (2006): Machine learning for detection and diagnosis of disease. *Annual Review of Biomedical Engineering*, vol. 8, pp. 537-565.

Sebastiani, F. (2002): Machine learning in automated text categorization. *ACM Computing Surveys (CSUR)*, vol. 34, no. 1, pp. 1-47.

Simulia, D. (2011): Abaqus 6.11 analysis user's manual. *Abaqus*, vol. 6, pp. 22-25.

Srivastava, N.; Hinton, G.; Krizhevsky, A.; Sutskever, I.; Salakhutdinov, R. (2014): Dropout: A simple way to prevent neural networks from overfitting. *Journal of Machine Learning Research*, vol. 15, no. 1, pp. 1929-1958.

Zheng, X.; Zheng, P.; Zhang, R.-Z. (2018): Machine learning material properties from the periodic table using convolutional neural networks. *Chemical Science*, vol. 9, no. 44, pp. 8426-8432.

Zhou, Q.; Tang, P.; Liu, S.; Pan, J.; Yan, Q. (2018): Learning atoms for materials discovery. *Proceedings of the National Academy of Sciences*, vol. 115, no. 28, pp. E6411-E6417.