Available online at www.sciencedirect.com
ScienceDirect
Comput. Methods Appl. Mech. Engrg. 372 (2020) 113401
Computer methods in applied mechanics and engineering
www.elsevier.com/locate/cma

Deep learned finite elements

Jaeho Jung[a,b], Kyungho Yoon[b], Phill-Seung Lee[b,*]

[a] Korea Atomic Energy Research Institute, 989 Daedeok-daero, Yuseong-gu, Daejeon 34057, Republic of Korea
[b] Department of Mechanical Engineering, Korea Advanced Institute of Science and Technology, 291 Daehak-ro, Yuseong-gu, Daejeon 34141, Republic of Korea

Received 12 May 2020; received in revised form 21 August 2020; accepted 26 August 2020
Available online xxxx

Abstract

In this paper, we propose a method that employs deep learning, an artificial intelligence technique, to generate stiffness matrices of finite elements. The proposed method is used to develop 4- and 8-node 2D solid finite elements. The deep learned finite elements practically pass the patch tests and the zero energy mode tests. Through various numerical examples, the performance of the developed elements is investigated and compared with those of existing elements. Computation efficiency is also studied. It was confirmed that the deep learned finite elements can potentially outperform existing finite elements. The proposed method can be applied to generate various types of finite elements in the future.

Keywords: Finite element; Solid element; Stiffness matrix; Artificial intelligence; Deep learning; Neural network

1. Introduction

Finite element method (FEM) is widely employed not only in structural analysis but also in almost all fields of engineering such as analysis of electromagnetic fields, flows, heat transfer, fluid–structure interactions [1–8]. In particular, FEM is the most powerful tool for structural analysis. FEM is inevitably used in various stages, from product design to manufacturing. However, the challenge of improving FEM is still being addressed.

An artificial neural network (ANN) is a brain-inspired system consisting of connected artificial neurons and is used to perform tasks based on data without specific rules. After the pioneering works of McCulloch and Pitts [9] and Rosenblatt [10], Hinton et al. [11,12] developed advanced algorithms for training networks that enabled the use of deep learning. Deep learning was used in AlexNet [13], which won the ImageNet Large Scale Visual Recognition Challenge (ILSVRC 2012) owing to its outstanding image recognition ability, and its application has gained momentum. Alpha Go [14] and Alpha Go Zero [15] use deep learning and have drawn worldwide attention for defeating top-ranked professional Go players. Deep learning is increasingly being applied in various fields such as automotive industry, medicine, finance and law.

Deep learning is also being studied for application in the field of numerical analysis, and several attempts have been made to use deep learning to solve partial differential equations [16–18]. Furthermore, deep learning has actively been adopted for computational fluid dynamics [19–24]. Several studies have related deep learning to the

* Corresponding author.
E-mail address: phillseung@kaist.edu (P.S. Lee).

https://doi.org/10.1016/j.cma.2020.113401
0045-7825/

FEM. Takeuchi and Kosugi [25] showed that FEM formulations can be expressed as a neural network. Deep learning has been applied to construct surrogate models [26,27] and constitutive models for material nonlinear finite element analysis [28,29]. Oishi and Yagawa [30] utilized deep learning to increase the accuracy of numerical integration when calculating the stiffness matrix of finite elements.

In this paper, we show how deep learning can be used to generate a stiffness matrix of finite elements. We construct a neural network to generate the strain–displacement matrix, which is a key component of the stiffness matrix, corresponding to the geometry and material properties given as input data. For efficient learning, the geometric information is normalized to reduce the amount of training data. Strain values corresponding to a given displacement are obtained from a reference data model discretized with a fine mesh. Using the obtained data, we train a neural network that can generate strain–displacement matrices at Gauss points. Preprocessing is performed to generate the input of the trained network, and post-processing is performed to generate the stiffness matrix from the output. A correction procedure is also applied to sufficiently represent rigid body motions of the finite elements.

The proposed method is used to develop 8-node and 4-node quadrilateral plane stress finite elements, which we call "deep learned finite elements." Basic numerical tests, including the patch and zero-energy mode tests, are carried out. The performance of the developed deep learned finite elements is demonstrated through various numerical problems. This study shows that finite elements can be generated by using deep learning and that deep learned finite elements can outperform existing finite elements.

The rest of this paper is organized as follows. Section 2 briefly reviews the standard isoparametric finite element procedure. Section 3 presents the method for generating 8-node quadrilateral finite elements by deep learning, including data generation, network configuration and training, and the construction of the stiffness matrix. Section 4 presents the method for generating 4-node quadrilateral finite elements. Section 5 reports the basic test results of the deep learned finite elements, and Sections 6 and 7 discuss the performance of the obtained finite elements through various numerical examples. Finally, the conclusions are presented in Section 8.

2. Isoparametric finite element procedure

The procedure to generate deep learned finite elements is based on the formulation of isoparametric finite elements. In this section, we briefly review the isoparametric finite element procedure for a q-node 2D quadrilateral solid element [1].

The geometry of the q-node element is interpolated by

$$\mathbf{x} = \sum_{i=1}^{q} h_i(r,\, s)\mathbf{x}_i \quad \text{with} \quad \mathbf{x} = \begin{bmatrix} x & y \end{bmatrix}^{\mathrm{T}} \quad \text{and} \quad \mathbf{x}_i = \begin{bmatrix} x_i & y_i \end{bmatrix}^{\mathrm{T}}, \tag{1}$$

where $\mathbf{x}_i$ is the position vector of node i in the global Cartesian coordinate system as shown in Fig. 1(a) and $h_i(r,\, s)$ are the shape functions defined in the natural coordinate system as shown in Fig. 1(b).

The corresponding displacement interpolation is given by

$$\mathbf{u} = \sum_{i=1}^{q} h_i(r,\, s)\mathbf{u}_i \quad \text{with} \quad \mathbf{u} = \begin{bmatrix} u & v \end{bmatrix}^{\mathrm{T}} \quad \text{and} \quad \mathbf{u}_i = \begin{bmatrix} u_i & v_i \end{bmatrix}^{\mathrm{T}}, \tag{2}$$

in which $\mathbf{u}_i$ is the displacement vector of node i.

The derivatives of the displacement with respect to the global coordinates are calculated using the Jacobian matrix $\mathbf{J}$:

$$\begin{bmatrix} \dfrac{\partial u}{\partial x} \\ \dfrac{\partial u}{\partial y} \end{bmatrix} = \mathbf{J}^{-1} \begin{bmatrix} \dfrac{\partial u}{\partial r} \\ \dfrac{\partial u}{\partial s} \end{bmatrix}, \quad \begin{bmatrix} \dfrac{\partial v}{\partial x} \\ \dfrac{\partial v}{\partial y} \end{bmatrix} = \mathbf{J}^{-1} \begin{bmatrix} \dfrac{\partial v}{\partial r} \\ \dfrac{\partial v}{\partial s} \end{bmatrix} \quad \text{with} \quad \mathbf{J} = \begin{bmatrix} \dfrac{\partial x}{\partial r} & \dfrac{\partial y}{\partial r} \\ \dfrac{\partial x}{\partial s} & \dfrac{\partial y}{\partial s} \end{bmatrix}. \tag{3}$$

The derivatives in Eq. (3) are used to obtain the strain–displacement matrix $\mathbf{B}$:

$$\boldsymbol{\varepsilon} = \mathbf{B}(r, s)\mathbf{u}, \tag{4}$$

where $\boldsymbol{\varepsilon}$ is the strain vector and $\mathbf{u}$ is the nodal displacement vector

$$\begin{aligned} \boldsymbol{\varepsilon} &= \begin{bmatrix} \varepsilon_{xx} & \varepsilon_{yy} & \gamma_{xy} \end{bmatrix}^{\mathrm{T}} \quad \text{with} \quad \gamma_{xy} = 2\varepsilon_{xy}, \\ \mathbf{u} &= \begin{bmatrix} u_1 & u_2 & \cdots & u_q & v_1 & v_2 & \cdots & v_q \end{bmatrix}^{\mathrm{T}}. \end{aligned} \tag{5}$$

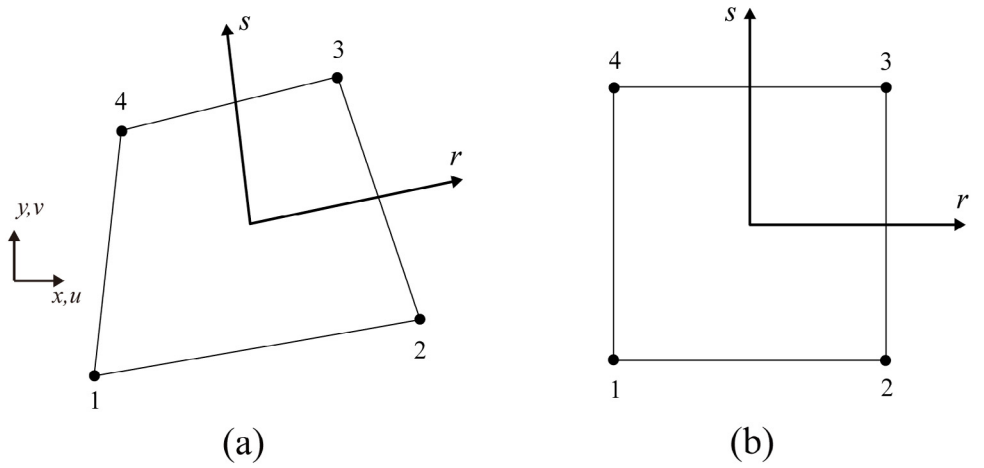

Fig. 1. 4-node quadrilateral element in the (a) global Cartesian coordinate system and (b) natural coordinate system.

Note that the matrix $\mathbf{B}$ has the dimensions of $3 \times 2q$ because the strain and nodal displacement vectors contain 3 and $2q$ components, respectively.

The stiffness matrix ($\mathbf{K}$) of the 2D solid element with the thickness t is given by

$$\mathbf{K} = t \int_{-1}^{1} \int_{-1}^{1} \mathbf{B}^{\mathrm{T}}(r,s)\mathbf{C}\mathbf{B}(r,s) \det \mathbf{J}(r,s) dr ds, \tag{6}$$

in which $\mathbf{C}$ is the material law matrix.

The stiffness matrix in Eq. (6) is numerically calculated using the $p \times p$ Gaussian quadrature

$$\mathbf{K} = t \sum_{i=1}^{p} \sum_{j=1}^{p} {}^{(i,j)}w^{(i,j)} \mathbf{B}^{\mathrm{T}} \mathbf{C}^{(i,j)} \mathbf{B}^{(i,j)} J, \tag{7}$$

where ${}^{(i,j)}w$ denotes the weight factor at the Gauss point (r_i, s_j), ${}^{(i,j)}J = \det \mathbf{J}(r_i, s_j)$, and ${}^{(i,j)}\mathbf{B} = \mathbf{B}(r_i, s_j)$.

3. Deep learned finite elements

In this section, we introduce the procedure to generate the strain–displacement matrix of an 8-node quadrilateral finite element *via* deep learning and to obtain its stiffness matrix. The methodology for constructing the neural network model is presented in detail, including data generation, network configuration and training. Finally, we present how to obtain the stiffness matrix using the trained neural network.

3.1. Data generation

In order to construct a neural network that generates the strain–displacement matrix of an arbitrary finite element, a large amount of strain data corresponding to randomly given geometries, displacements, and material properties are required. Since processing all these data is extremely difficult, a key point in this study was to appropriately reduce the amount of data for efficient network training.

In this study, the geometry of 8-node finite elements is limited to a quadrilateral whose mid-side node (nodes 5–8 in Fig. 2) are placed at the center of the adjacent corner nodes (nodes 1–4 in Fig. 2). In other words, the shape of the element is determined by the locations of its four corner nodes. We also use normalized geometry as a representative of all the similar shapes. Here, the normalized geometry refers to a quadrilateral where the two nodes at either end of the maximum length side are located at (0, 0) and (1, 0) in 2D Cartesian coordinates.

The nth normalized random geometry is generated as follows. Let the position vector of node i corresponding to the nth geometry be denoted as $\mathbf{x}_i^{(n)}$. Node 1 ($\mathbf{x}_1^{(n)}$) and node 2 ($\mathbf{x}_2^{(n)}$) of the nth geometry are fixed at the coordinates of (0, 0) and (1, 0), respectively, as shown in Fig. 2. Then, the coordinates of the other two corner nodes ($\mathbf{x}_3^{(n)}$ and $\mathbf{x}_4^{(n)}$) are randomly placed where the distance between node 1 and node 2 should be the maximum length edge.

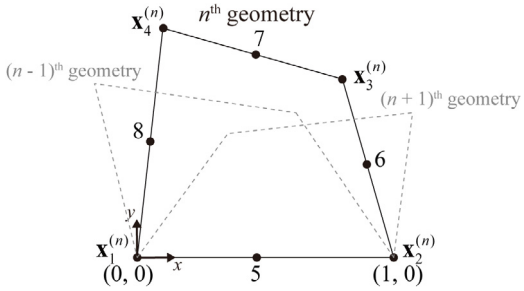

Fig. 2. Random generation of the nth normalized element geometry.

In this way, normalized element geometries can be generated. Here, we excluded severely distorted geometries such as quadrilaterals with an interior angle of less than 10° or greater than 170° and ratios between the maximum and minimum side lengths of greater than 10. Young's modulus ($E = 2.0 \times 10^{11}$) was adopted, and Poisson's ratio ($\upsilon^{(n)}$) was randomly applied with a uniform distribution in the range of 0–0.499999999.

To derive the relation between the nodal displacements ($\mathbf{u}_i^{(n)}$) and the corresponding reference strains in an element with the nth geometry and $\upsilon^{(n)}$ (hereafter called element n), nodal displacements are generated randomly with a uniform distribution in the range of -0.25 to 0.25, as shown in Fig. 3(a). The reference strain data are obtained from a reference data model having the same geometry as element n with a uniform $N \times N$ mesh, see Fig. 3(b). In this study, the reference data model was discretized using standard 4-node quadrilateral elements (Q4) [31].

The reference data model has $(N+1)^2$ nodes and $4 \times N$ outer nodes. We then map the outer displacements of the element n into those of the reference data model. To do so, the displacement $\hat{\mathbf{u}}_j^{(n)}$ at each outer node of the model is obtained from the displacement interpolation of the element n. The displacement vector at outer node j is given by

$$\hat{\mathbf{u}}_j^{(n)} = \sum_{i=1}^{8} h_i(r_j, s_j)\mathbf{u}_i^{(n)}, \tag{8}$$

in which (r_j, s_j) represents the natural coordinates of the element n corresponding to outer node j of the reference data model, and h_i is ith shape function of standard 8-node quadrilateral element [31]. Note that $N = 50$ was used in this study (see Appendix B).

After the displacements of all outer nodes are mapped, the outer nodal displacements are applied to the reference data model as prescribed displacements. The nodal displacement vector of the reference data model is divided into the inner displacements ($\hat{\mathbf{u}}_{\mathrm{I}}^{(n)}$) and outer displacements ($\hat{\mathbf{u}}_{\mathrm{O}}^{(n)}$), and the equilibrium equation is as follows:

$$\hat{\mathbf{K}}^{(n)} \begin{bmatrix} \hat{\mathbf{u}}_{\mathrm{I}}^{(n)} \\ \hat{\mathbf{u}}_{\mathrm{O}}^{(n)} \end{bmatrix} = \begin{bmatrix} \mathbf{0} \\ \hat{\mathbf{R}}_{\mathrm{O}} \end{bmatrix} \quad \text{and} \quad \hat{\mathbf{K}}^{(n)} = \begin{bmatrix} \hat{\mathbf{K}}_{\mathrm{II}} & \hat{\mathbf{K}}_{\mathrm{IO}} \\ \hat{\mathbf{K}}_{\mathrm{OI}} & \hat{\mathbf{K}}_{\mathrm{OO}} \end{bmatrix}, \tag{9}$$

where $\hat{\mathbf{K}}^{(n)}$ is the stiffness matrix of the reference data model and $\hat{\mathbf{R}}_{\mathrm{O}}$ is the reaction force vector.

Then, the inner nodal displacements are calculated using

$$\hat{\mathbf{u}}_{\mathrm{I}}^{(n)} = -\left(\hat{\mathbf{K}}_{\mathrm{II}}\right)^{-1} \hat{\mathbf{K}}_{\mathrm{IO}} \hat{\mathbf{u}}_{\mathrm{O}}^{(n)}. \tag{10}$$

Note that the inner nodal displacements are obtained regardless of Young's modulus, but depend on Poisson's ratio. For this reason, Young's modulus was not randomly applied (see Appendix A).

Using the displacements calculated in Eq. (10), the strain field of the reference data model is obtained. Then, strain values are extracted from the reference data model at the points corresponding to the 3×3 Gauss points in element n. The strain vector corresponding to Gauss point (i, j) is defined by ${}^{(i,j)}\hat{\boldsymbol{\varepsilon}}^{(n)} = \left[{}^{(i,j)}\hat{\varepsilon}_{xx}^{(n)} \quad {}^{(i,j)}\hat{\varepsilon}_{yy}^{(n)} \quad {}^{(i,j)}\hat{\gamma}_{xy}^{(n)}\right]^{\mathrm{T}}$.

Poisson's ratio, the nodal coordinates of the normalized geometry, the nodal displacements, and the strain values extracted from the reference data model are made into one training data. The nth training data ($\mathbf{D}^{(n)}$) is configured as

$$\mathbf{D}^{(n)} = \begin{bmatrix} \upsilon^{(n)} & \mathbf{D}_x^{(n)} & \mathbf{D}_u^{(n)} & \mathbf{D}_\varepsilon^{(n)} \end{bmatrix} \tag{11}$$

with

$$\mathbf{D}_x^{(n)} = \begin{bmatrix} \mathbf{x}_3^{(n)\mathrm{T}} & \mathbf{x}_4^{(n)\mathrm{T}} \end{bmatrix}, \quad \mathbf{D}_u^{(n)} = \begin{bmatrix} \mathbf{u}_1^{(n)\mathrm{T}} & \cdots & \mathbf{u}_8^{(n)\mathrm{T}} \end{bmatrix} \quad \text{and} \quad \mathbf{D}_\varepsilon^{(n)} = \begin{bmatrix} {}^{(1,1)}\hat{\boldsymbol{\varepsilon}}^{(n)\mathrm{T}} & \cdots & {}^{(3,3)}\hat{\boldsymbol{\varepsilon}}^{(n)\mathrm{T}} \end{bmatrix},$$

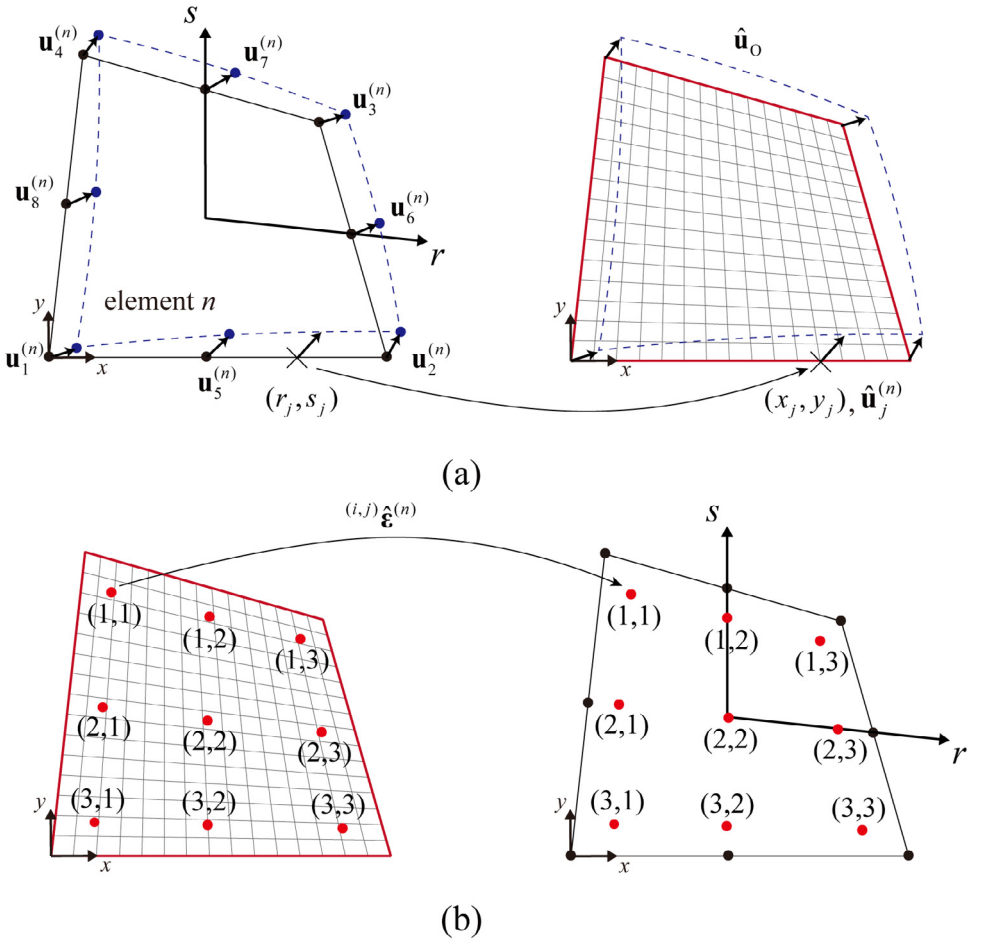

Fig. 3. Schematics of the reference strain data generation procedure. (a) Random displacements generation and mapping of the outer displacement to the reference data model. (b) Strain extraction from the reference data model. The red dots represent the location of Gauss points where strain values are extracted. . (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)

where $\upsilon^{(n)}$, $\mathbf{D}_x^{(n)}$, $\mathbf{D}_u^{(n)}$, and $\mathbf{D}_\varepsilon^{(n)}$ denote Poisson's ratio (1 value), nodal coordinates (2 × 2 values), nodal displacements (8 × 2 values), and strain values (3 × 3 × 3 values), respectively. In total, the nth training data contain 48 values.

3.2. Network configuration and training

The output of the network is ${}^{(i,j)}_{\text{output}}\mathbf{B}^{(n)}$, the nth normalized strain–displacement matrix (3 × 16) at Gauss point (i, j). The row and column of the matrix correspond to 3 strains and 16 nodal displacements (eight for u and eight for v), respectively.

Poisson's ratio ($\upsilon^{(n)}$) and the nodal coordinates ($\mathbf{D}_x^{(n)}$) of the training data are the inputs of the neural network, while the nodal displacements ($\mathbf{D}_u^{(n)}$) and strain values ($\mathbf{D}_\varepsilon^{(n)}$) of the training data are used for the following cost function $C(\boldsymbol{\theta})$:

$$C(\boldsymbol{\theta}) = \frac{1}{27M}\sum_{n=1}^{M}\sum_{i=1}^{3}\sum_{j=1}^{3}\sum_{k=1}^{3}{}^{(i,j)}w\left|\frac{\sum_{l=1}^{16}\left({}^{(i,j)}_{\text{output}}\mathrm{b}_{kl}^{(n)}(\boldsymbol{\theta})\,u_l^{(n)}\right)-{}^{(i,j)}\hat{\varepsilon}_k^{(n)}}{{}^{(i,j)}\hat{\varepsilon}_k^{(n)}}\right|, \tag{12}$$

where $\boldsymbol{\theta}$ denote the network weights, M is the number of training data, ${}^{(i,j)}_{\text{output}}\mathrm{b}^{(n)}_{kl}(\boldsymbol{\theta})$ is the component at the kth row and lth column of ${}^{(i,j)}_{\text{output}}\mathbf{B}^{(n)}(\boldsymbol{\theta})$, and ${}^{(i,j)}w$ denotes the weight factor corresponding to Gauss point (i, j). In Eq. (12), ${}^{(i,j)}\hat{\varepsilon}^{(n)}_1 = {}^{(i,j)}\hat{\varepsilon}^{(n)}_{xx}$, ${}^{(i,j)}\hat{\varepsilon}^{(n)}_2 = {}^{(i,j)}\hat{\varepsilon}^{(n)}_{yy}$, and ${}^{(i,j)}\hat{\varepsilon}^{(n)}_3 = {}^{(i,j)}\hat{\gamma}^{(n)}_{xy}$.

It is physically essential that finite elements should produce zero-strain energy under rigid body translations and rotations. Accordingly, the strain–displacement matrix generated from the network should satisfy the following conditions:

$$ {}^{(i,j)}_{\text{output}}\mathbf{B}^{(n)}(\boldsymbol{\theta})\Delta\mathbf{u} = \mathbf{0}, \tag{13} $$

in which $\Delta\mathbf{u}$ is the displacement vector corresponding to rigid body translations.

The three strain components (ε_{xx}, ε_{yy} and γ_{xy}) should be zero for the x- and y-directional rigid body translations at all 3×3 Gauss points, which yields the following equations:

$$ \sum_{l=1}^{8} {}^{(i,j)}_{\text{output}}\mathrm{b}^{(n)}_{kl}(\boldsymbol{\theta}) = 0 \quad \text{and} \quad \sum_{l=9}^{16} {}^{(i,j)}_{\text{output}}\mathrm{b}^{(n)}_{kl}(\boldsymbol{\theta}) = 0 \quad \text{for} \quad i,\ j,\ k = 1,\ 2,\ 3. \tag{14} $$

Note that the 1st to 8th columns correspond to the x-directional displacements (u) and the 9th to 16th columns correspond to the y-directional displacements (v) as shown in Eq. (5).

To enforce the matrix ${}^{(i,j)}_{\text{output}}\mathbf{B}^{(n)}(\boldsymbol{\theta})$ complying the constraints in Eq. (14), we generate an intermediate strain–displacement matrix ${}^{(i,j)}\breve{\mathbf{B}}^{(n)}(\boldsymbol{\theta})$ in the network. The intermediate matrix contains only the first 7 columns for each x- and y-directional displacement. That is, the 8th and 16th columns of ${}^{(i,j)}_{\text{output}}\mathbf{B}^{(n)}(\boldsymbol{\theta})$ are excluded and thus the intermediate matrix has the dimensions of 3×14. Then, the excluded columns are calculated according to Eq. (14) to produce the 3×16 matrix ${}^{(i,j)}_{\text{output}}\mathbf{B}^{(n)}(\boldsymbol{\theta})$ in the network:

$$ {}^{(i,j)}_{\text{output}}\mathrm{b}^{(n)}_{kl}(\boldsymbol{\theta}) = \begin{cases} {}^{(i,j)}\breve{\mathrm{b}}^{(n)}_{kl}(\boldsymbol{\theta}), & l = 1, 2, \ldots, 7 \\ -\sum_{l=1}^{7} {}^{(i,j)}\breve{\mathrm{b}}^{(n)}_{kl}(\boldsymbol{\theta}), & l = 8 \\ {}^{(i,j)}\breve{\mathrm{b}}^{(n)}_{k(l-1)}(\boldsymbol{\theta}), & l = 9, 10, \ldots, 15 \\ -\sum_{l=8}^{14} {}^{(i,j)}\breve{\mathrm{b}}^{(n)}_{kl}(\boldsymbol{\theta}), & l = 16 \end{cases} \quad \text{for} \quad i,\ j,\ k = 1,\ 2,\ 3, \tag{15} $$

where ${}^{(i,j)}\breve{\mathrm{b}}^{(n)}_{kl}(\boldsymbol{\theta})$ are the components of the intermediate matrix ${}^{(i,j)}\breve{\mathbf{B}}^{(n)}(\boldsymbol{\theta})$.

A network was constructed as shown in Fig. 4. A fully connected neural network of 6 layers was employed because its structure can be used as a universal approximator [32,33], and batch normalization was applied to each layer. An exponential linear unit was used as an activation function at each layer before the output. The network width was 378 ($= 3 \times 3 \times 3 \times 14$), which was the same as the number of all components of ${}^{(i,j)}\breve{\mathbf{B}}^{(n)}(\boldsymbol{\theta})$ (3×14) generated at 3×3 Gauss points. After the 378 outputs were reshaped, ${}^{(i,j)}\breve{\mathbf{B}}^{(n)}(\boldsymbol{\theta})$ was generated at each Gauss point. Finally, ${}^{(i,j)}_{\text{output}}\mathbf{B}^{(n)}(\boldsymbol{\theta})$ was obtained using Eq. (15).

The network was trained with a total of 300,000 data (M = 300,000). To test the network, 50,000 data were generated of which 30,000 data were randomly selected for testing. When the strain value ${}^{(i,j)}\hat{\varepsilon}^{(n)}_k$ is close to 0, the cost function $C(\boldsymbol{\theta})$ in Eq. (12) increases sensitively even though the network generates a strain close enough to the input strains ($\mathbf{D}^{(n)}_{\varepsilon}$). Therefore, training data containing an absolute value of less than 0.005 in the components of $\mathbf{D}^{(n)}_{\varepsilon}$ were excluded.

When such training data were excluded in this way, the generated element largely failed in the shearing patch test because data corresponding to pure shearing were not included in training data. Therefore, additional training data for pure shearing were generated and used for network training. We generated 3000 x-directional shearing data by applying $\mathbf{u}^{(n)}_i = \begin{bmatrix} y^{(n)}_i & 0 \end{bmatrix}^{\mathrm{T}}$, and 3000 y-directional shearing data by applying $\mathbf{u}^{(n)}_i = \begin{bmatrix} 0 & x^{(n)}_i \end{bmatrix}^{\mathrm{T}}$. In addition, 1200 test data (600 shearing data for each direction) were generated. The zero-strain components generated from the additional data were replaced with 0.5% of the maximum strain component in each data because the cost function did not converge well when the zero values were trained.

The network was implemented using TensorFlow [34], Adam optimization [35] was adopted as the optimizer, and Xavier initializer [36] was applied to initialize the weights of the network. We performed training for 30,000 epochs, and a batch size of 50,000 was used. The learning rate converged linearly from 0.01 to 0 as the epoch

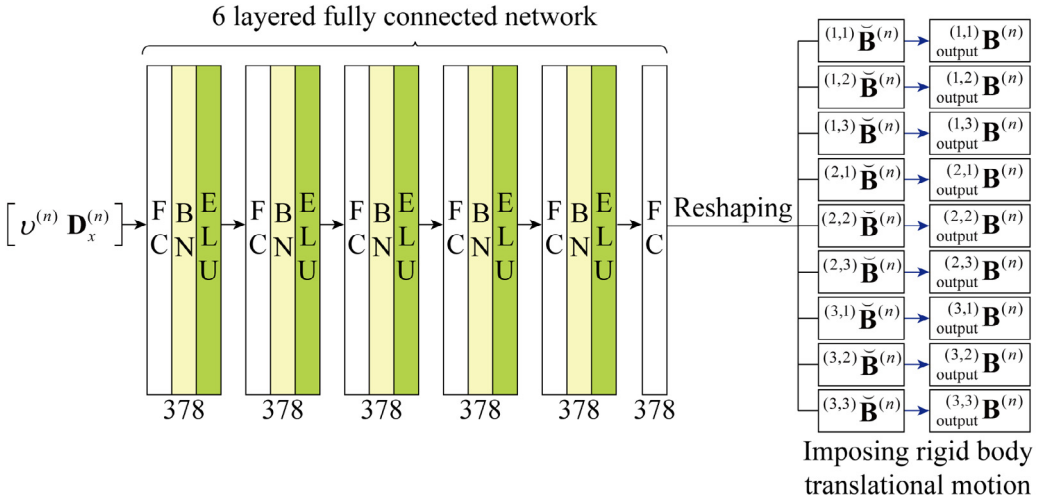

Fig. 4. Network configuration for deep learned finite elements. (FC: fully connected layer, BN: batch normalization, ELU: exponential linear unit).

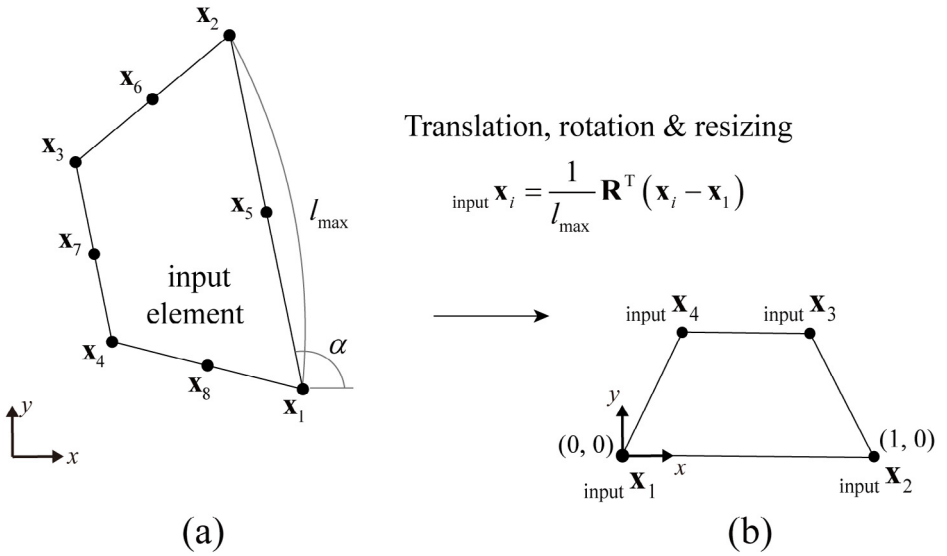

Fig. 5. Pre-processing procedure to obtain the network input. (a) Original element geometry. (b) Normalized element geometry.

progressed. In the early stage of training, the learning rate of the network weights was set to a large value to broadly search $C(\boldsymbol{\theta})$. Then, the learning rate was gradually decreased to zero, and thereby $C(\boldsymbol{\theta})$ precisely reached the minimum value. As a result of training, the average error for the training data was 1.24% and the average error for the test data was 1.67%.

3.3. Construction of the stiffness matrix

Normalized geometries were employed for the efficient training of the network. In order to apply the trained network to elements with arbitrary geometries, pre-processing for the input of the trained network is necessary. In addition, post-processing of the network output is required to generate the stiffness matrix.

3.3.1. Pre-processing of the network input

Geometry normalization is performed for an element with an arbitrary geometry. The element connectivity is assigned so that the side length between node 1 and node 2 is the longest. Then, as shown in Fig. 5, the nodal

coordinates of the element ($\mathbf{x}_i$) are translated, rotated, and resized to obtain the input normalized nodal coordinates ($_{\text{input}}\mathbf{x}_i$) where node 1 and node 2 are positioned at (0, 0) and (1, 0), respectively.

The normalized nodal coordinates are obtained by

$$_{\text{input}}\mathbf{x}_i = \frac{1}{l_{\max}}\mathbf{R}^{\mathrm{T}}(\mathbf{x}_i - \mathbf{x}_1) \quad \text{for} \quad i = 1, 2, 3, 4 \tag{16}$$

$$\text{with } \mathbf{R} = \begin{bmatrix} \cos\alpha & -\sin\alpha \\ \sin\alpha & \cos\alpha \end{bmatrix},$$

in which α is the angle between the longest side and the x-axis (see Fig. 5(a)), and $l_{\max}$ is the longest side length. The coordinates of $_{\text{input}}\mathbf{x}_3$ and $_{\text{input}}\mathbf{x}_4$ obtained from this process and Poisson's ratio υ are used as input data for the trained network.

3.3.2. Post-processing of the network output

The trained network outputs the strain–displacement matrices for the element ($^{(i,j)}_{\text{output}}\mathbf{B}$) of normalized geometry at every Gauss point. The post-processing for the network output is performed to calculate the strain–displacement matrix of the element ($^{(i,j)}_{\text{DL8}}\mathbf{B}$) of the original geometry, as shown in Fig. 6.

The strain–displacement matrix $^{(i,j)}_{\text{DL8}}\mathbf{B}$ is obtained using the following equation

$$^{(i,j)}_{\text{DL8}}\mathbf{B} = \frac{1}{l_{\max}}\mathbf{T}^{(i,j)}_{\text{output}}\mathbf{B}\mathbf{Q}^{\mathrm{T}}, \tag{17}$$

with

$$\mathbf{T} = \begin{bmatrix} \cos^2\alpha & \sin^2\alpha & -\sin\alpha\cos\alpha \\ \sin^2\alpha & \cos^2\alpha & \sin\alpha\cos\alpha \\ 2\sin\alpha\cos\alpha & -2\sin\alpha\cos\alpha & \cos^2\alpha - \sin^2\alpha \end{bmatrix},$$

$$\mathbf{Q} = (q_{kl}) \in \mathbb{R}^{16\times 16} \quad \text{with} \quad q_{kl} = \delta_{kl}\cos\alpha - \delta_{k(l-8)}\sin\alpha + \delta_{(k-8)l}\sin\alpha,$$

in which $\mathbf{T}$ is the strain transformation matrix [37], $\mathbf{Q}$ is the displacement rotation matrix, and δ_{kl} represents the Kronecker delta. Note that $\mathbf{Q}$ in Eq. (17) is given according to the order of the components of $\mathbf{u}$ in Eq. (5).

Since $^{(i,j)}_{\text{DL8}}\mathbf{B}$ is the strain–displacement matrix at Gauss point (i, j) approximated by the network, it is very hard to make the element pass the patch tests exactly. Therefore, our goal is for the element to pass the patch tests as close as possible. To do so, we correct the matrix using the well-known B-bar method [38]. The corrected strain–displacement matrix $^{(i,j)}_{\text{DL8}}\bar{\mathbf{B}}$ is obtained as

$$^{(i,j)}_{\text{DL8}}\bar{\mathbf{B}} = {}^{(i,j)}_{\text{DL8}}\mathbf{B} + {}_{\text{DL8}}\mathbf{B}' \quad \text{with} \quad {}_{\text{DL8}}\mathbf{B}' = \frac{t}{V}\sum_{i=1}^{3}\sum_{j=1}^{3} {}^{(i,j)}w\,(^{(i,j)}_{\text{Q8}}\mathbf{B} - {}^{(i,j)}_{\text{DL8}}\mathbf{B})^{(i,j)}J, \tag{18}$$

where V is the element volume and $^{(i,j)}_{\text{Q8}}\mathbf{B}$ is the strain–displacement matrix of the standard 8-node quadrilateral element at Gauss point (i, j) [31].

Using the corrected strain–displacement matrix $^{(i,j)}_{\text{DL8}}\bar{\mathbf{B}}$, the stiffness matrix of the element is finally calculated by

$${}_{\text{DL8}}\mathbf{K} = t\sum_{i=1}^{3}\sum_{j=1}^{3} {}^{(i,j)}w\,{}^{(i,j)}_{\text{DL8}}\bar{\mathbf{B}}^{\mathrm{T}}\mathbf{C}\,{}^{(i,j)}_{\text{DL8}}\bar{\mathbf{B}}^{(i,j)}J. \tag{19}$$

In the deep learned finite elements developed in this study, a strain–displacement matrix is only generated at Gauss points. Therefore, strain and stress values are calculated at Gauss points. The strain and stress fields in the element are obtained by extrapolating the strain and stress values at the Gauss points.

4. 4-node deep learned quadrilateral finite elements

This section describes the procedure to generate the strain–displacement matrix of a 4-node quadrilateral element and to obtain its stiffness matrix. The 4-node element is degenerated from the 8-node deep learned quadrilateral element obtained in Section 3.

The corner and mid-side nodes of the 8-node element are considered as compatible and incompatible nodes, respectively, as shown in Fig. 7. The strain–displacement matrix for the 8-node deep learned quadrilateral element,

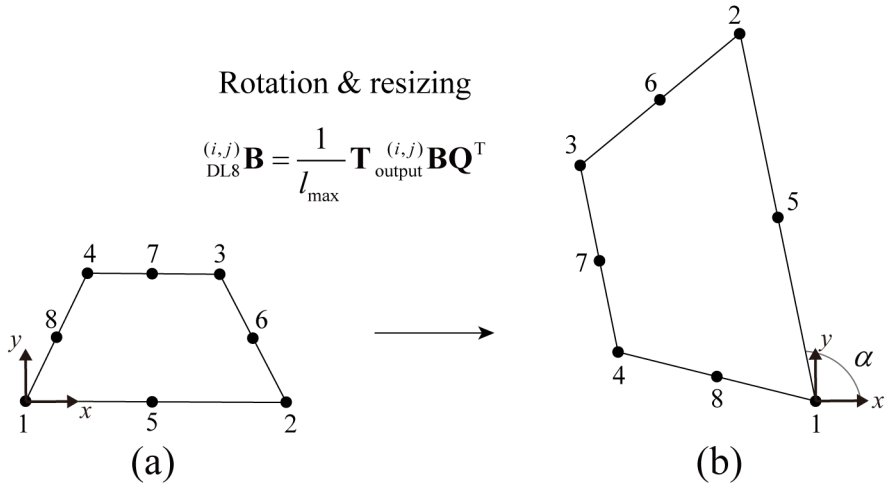

Fig. 6. Post-processing procedure for the network output to obtain strain–displacement matrices for the original element geometry. (a) Normalized element geometry. (b) Original element geometry.

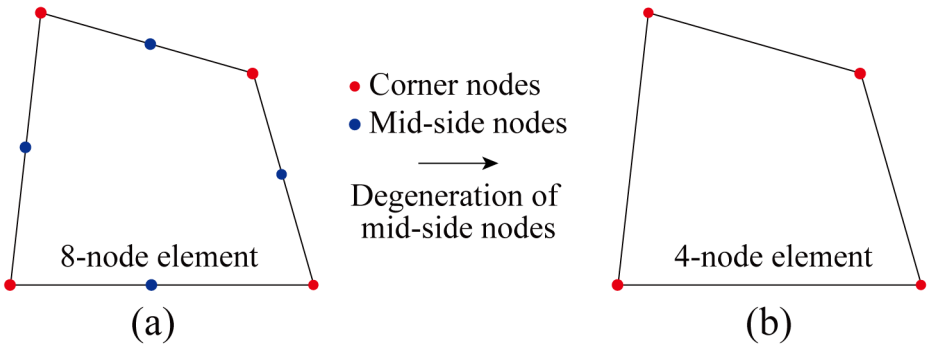

Fig. 7. Degeneration process for the 4-node deep learned quadrilateral element. (a) 8-node deep learned finite element generated from the trained network. (b) 4-node deep learned element obtained through the degeneration of mid-side nodes.

$^{(i,j)}_{\mathrm{DL8}}\mathbf{B}$ in Eq. (17), is divided into two parts corresponding to the displacements at compatible and incompatible nodes as follows:

$$^{(i,j)}_{\mathrm{DL8}}\mathbf{B}\mathbf{u} = \begin{bmatrix} {}^{(i,j)}\mathbf{B}_{\mathrm{C}} & {}^{(i,j)}\mathbf{B}_{\mathrm{I}} \end{bmatrix} \begin{bmatrix} \mathbf{u}_{\mathrm{C}} \\ \mathbf{u}_{\mathrm{I}} \end{bmatrix}, \tag{20}$$

where $^{(i,j)}\mathbf{B}_{\mathrm{C}}$ and $^{(i,j)}\mathbf{B}_{\mathrm{I}}$ are the strain–displacement matrices corresponding to displacements at compatible and incompatible nodes, respectively, and $\mathbf{u}_{\mathrm{C}}$ and $\mathbf{u}_{\mathrm{I}}$ are displacement vectors corresponding to compatible and incompatible nodes, respectively.

To make the element pass the patch test more closely, the strain–displacement matrices are corrected as follows

$$^{(i,j)}\bar{\mathbf{B}}_{\mathrm{C}} = {}^{(i,j)}\mathbf{B}_{\mathrm{C}} + \mathbf{B}'_{\mathrm{C}} \quad \text{and} \quad {}^{(i,j)}\bar{\mathbf{B}}_{\mathrm{I}} = {}^{(i,j)}\mathbf{B}_{\mathrm{I}} + \mathbf{B}'_{\mathrm{I}}, \tag{21}$$

with

$$\mathbf{B}'_{\mathrm{C}} = \frac{t}{V}\sum_{i=1}^{3}\sum_{j=1}^{3} {}^{(i,j)}w \left({}^{(i,j)}_{\mathrm{Q4}}\mathbf{B} - {}^{(i,j)}\mathbf{B}_{\mathrm{C}} \right) {}^{(i,j)}J,$$

$$\mathbf{B}'_{\mathrm{I}} = -\frac{t}{V}\sum_{i=1}^{3}\sum_{j=1}^{3} {}^{(i,j)}w\,{}^{(i,j)}\mathbf{B}_{\mathrm{I}}\,{}^{(i,j)}J,$$

Table 1
Eigenvalues corresponding to the 1st–6th modes for the various geometries (in Fig. 8) of DL8 and DL4 elements. (The 1st, 2nd and 3rd modes correspond to rigid body motions.)

Mode	Geometry 1		Geometry 2		Geometry 3		Geometry 4	
	DL8	DL4	DL8	DL4	DL8	DL4	DL8	DL4
1	5.88E−14	4.82E−13	3.32E−13	1.83E−13	1.31E−13	8.31E−13	2.45E−13	7.93E−13
2	3.83E−13	9.43E−13	4.84E−13	7.84E−12	6.68E−13	1.25E−12	7.50E−13	2.86E−12
3	3.36E−03	1.31E−03	2.24E−04	4.87E−03	1.04E−04	5.01E−03	1.44E−03	2.33E−03
4	4.16E+02	1.15E+03	4.33E+02	4.36E+02	1.96E+02	4.69E+02	1.80E+01	7.41E+01
5	4.19E+02	1.15E+03	4.71E+02	5.25E+02	3.87E+02	5.02E+02	1.04E+02	2.73E+02
6	5.07E+02	4.97E+03	4.98E+02	1.14E+03	4.92E+02	1.18E+03	1.23E+02	2.12E+03

in which ${}^{(i,j)}\bar{\mathbf{B}}_{\mathrm{C}}$ and ${}^{(i,j)}\bar{\mathbf{B}}_{\mathrm{I}}$ are the corrected strain–displacement matrices and ${}^{(i,j)}_{\mathrm{Q4}}\mathbf{B}$ is the strain–displacement matrix of the standard 4-node element.

Using the corrected ${}^{(i,j)}\bar{\mathbf{B}}_{\mathrm{C}}$ and ${}^{(i,j)}\bar{\mathbf{B}}_{\mathrm{I}}$ in Eq. (21), the stiffness matrix is calculated as

$$\mathbf{K}=\begin{bmatrix}\mathbf{K}_{\mathrm{CC}} & \mathbf{K}_{\mathrm{CI}}\\ \mathbf{K}_{\mathrm{IC}} & \mathbf{K}_{\mathrm{II}}\end{bmatrix}=t\sum_{i=1}^{3}\sum_{j=1}^{3}{}^{(i,j)}w^{(i,j)}_{\mathrm{DL4}}\mathbf{B}^{\mathrm{T}}\,\mathbf{C}\,{}^{(i,j)}_{\mathrm{DL4}}\mathbf{B}^{(i,j)}J \tag{22}$$

with

$${}^{(i,j)}_{\mathrm{DL4}}\mathbf{B}=\left[{}^{(i,j)}\bar{\mathbf{B}}_{\mathrm{C}}\quad {}^{(i,j)}\bar{\mathbf{B}}_{\mathrm{I}}\right].$$

The submatrices related to incompatible displacements in Eq. (22) are eliminated using the static condensation procedure. Finally, the stiffness matrix of the deep learned 4-node finite element is obtained

$${}_{\mathrm{DL4}}\mathbf{K}=\mathbf{K}_{\mathrm{CC}}-\mathbf{K}_{\mathrm{CI}}(\mathbf{K}_{\mathrm{II}})^{-1}\mathbf{K}_{\mathrm{IC}}. \tag{23}$$

5. Basic numerical tests

In this section, zero energy mode and patch tests are performed for the deep learned 8-node (DL8) and 4-node (DL4) finite elements.

5.1. Zero energy mode tests

In the zero energy mode test, the zero eigenvalues of the stiffness matrix of a single unsupported element are counted. Undistorted (in Fig. 8(a)) and distorted (in Fig. 8(b), (c) and (d)) element geometries are considered with unit thickness. Young's modulus $E = 1.5 \times 10^3$ and Poisson's ratio $\upsilon = 0.3$ are given.

Table 1 presents the eigenvalues calculated up to the sixth strain energy modes. The first three eigenvalues correspond to the three rigid body modes (two translation and one rotation modes) for all geometry cases. The eigenvalue of mode 3 (rotation mode) is larger than those of mode 1 and 2 (translation modes). However, the three eigenvalues are sufficiently smaller than those of the deformation modes (modes 4, 5 and 6) and thus the practical use of the DL8 and DL4 elements is available.

5.2. Patch tests

Three patch tests are performed with the mesh geometry in Fig. 9(a) for x- and y-directional stretching and shearing [1]. The loading and displacement boundary conditions are shown in Fig. 9(b)–(d). The patch of elements is subjected to the minimum number of constraints to prevent rigid body motions and nodal point forces on the boundary corresponding to constant stress states are applied. If the constant stress fields are calculated, the patch tests are passed [39,40].

The deep learned finite elements (DL8 and DL4) practically pass the patch tests. The results of the patch test are shown in Fig. 10. Table 2 presents the minimum and maximum stress values across Gauss points. In other words, the deep learned finite elements can represent constant strain fields with practically sufficient accuracy.

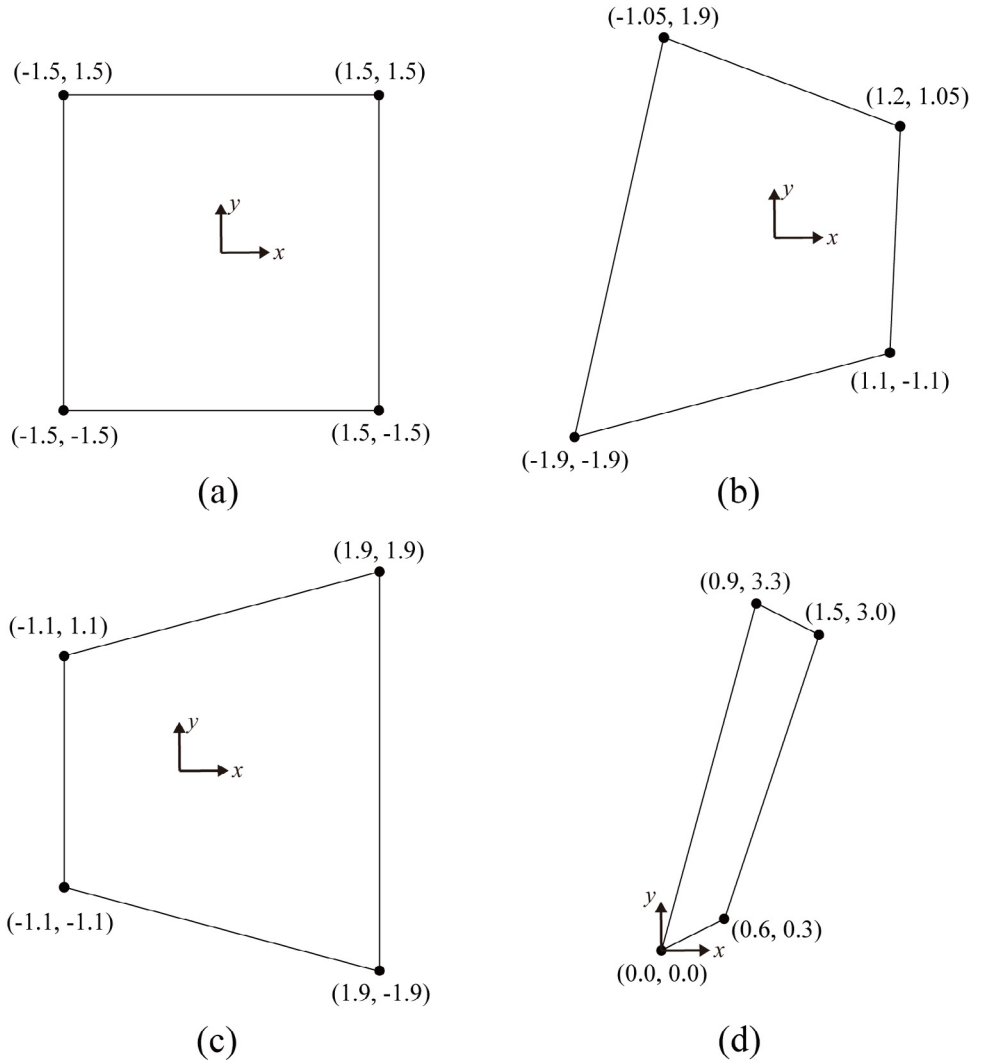

Fig. 8. Element geometries used for the zero energy mode test: (a) Geometry 1, (b) Geometry 2, (c) Geometry 3, and (d) Geometry 4.

Table 2
Minimum and maximum stress values across Gauss points in the patch tests (minimum value/maximum value).

		σ_{xx}	σ_{yy}	σ_{xy}
x-directional stretch in Fig. 9(b)	DL8 element	0.9950/1.0055	−0.0043/0.0037	−0.0055/0.0040
	DL4 element	0.9810/1.0169	−0.0074/0.0051	−0.0140/0.0134
	Ref. values	1.0	0.0	0.0
y-directional stretch in Fig. 9(c)	DL8 element	−0.0055/0.0050	0.9932/1.0065	−0.0051/0.0057
	DL4 element	−0.0057/0.0069	0.9895/1.0144	−0.0188/0.0264
	Ref. values	0.0	1.0	0.0
Shearing in Fig. 9(d)	DL8 element	−0.0059/0.0062	−0.0044/0.0059	0.9918/1.0052
	DL4 element	−0.0097/0.0073	−0.0050/0.0033	0.9884/1.0139
	Ref. values	0.0	0.0	1.0

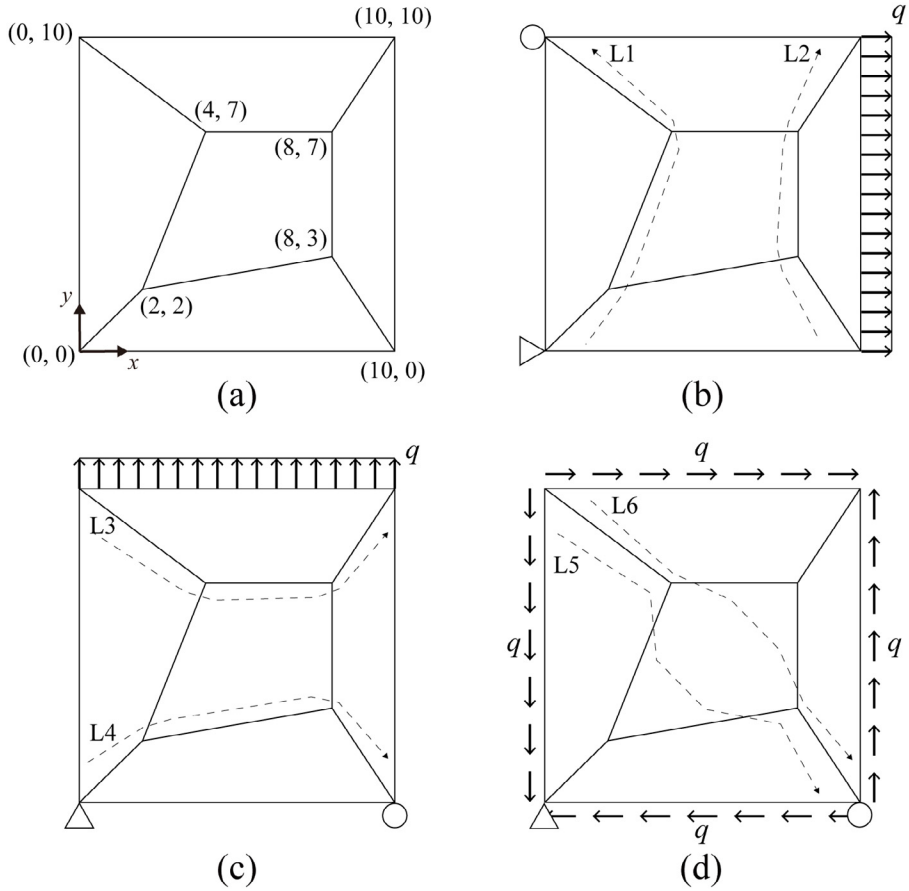

Fig. 9. Mesh geometry used for the patch tests is shown in (a) ($q = 1.0$, thickness $= 1.0$, $E = 3.0 \times 10^7$, $\nu = 0.3$). Loading and boundary conditions and the lines through element Gauss points for stress evaluation are shown in (b), (c) and (d).

6. Numerical examples

In this section, the performance of the proposed elements (DL8 and DL4) is investigated through various numerical problems: Cook's skew beam problem, taper beam problem, block problem, cantilever beam problem, and wrench problem.

The obtained results are compared with those of various existing elements as follows:

- Q4: Standard 4-node quadrilateral element [31]
- QM6: 4-node quadrilateral element with incompatible modes [41]
- Q8: Standard 8-node quadrilateral element [31]
- Q9: Standard 9-node quadrilateral element [31]
- P182: 4-node quadrilateral element in ANSYS [42]
- P183: 8-node quadrilateral element in ANSYS [42]

Therefore, 4 quadratic elements (Q8, Q9, P183 and DL8) and 4 linear elements (Q4, QM6, P182 and DL4) are considered. In general, 9-node elements have higher accuracy than 8-node elements, but 8-node elements have less degrees of freedom.

To investigate the predictive capability of the elements in detail, convergence studies are performed. The solution convergence is measured using the relative strain energy error given as

$$E_e = \left| \frac{E_{\text{ref}} - E_h}{E_{\text{ref}}} \right|, \tag{24}$$

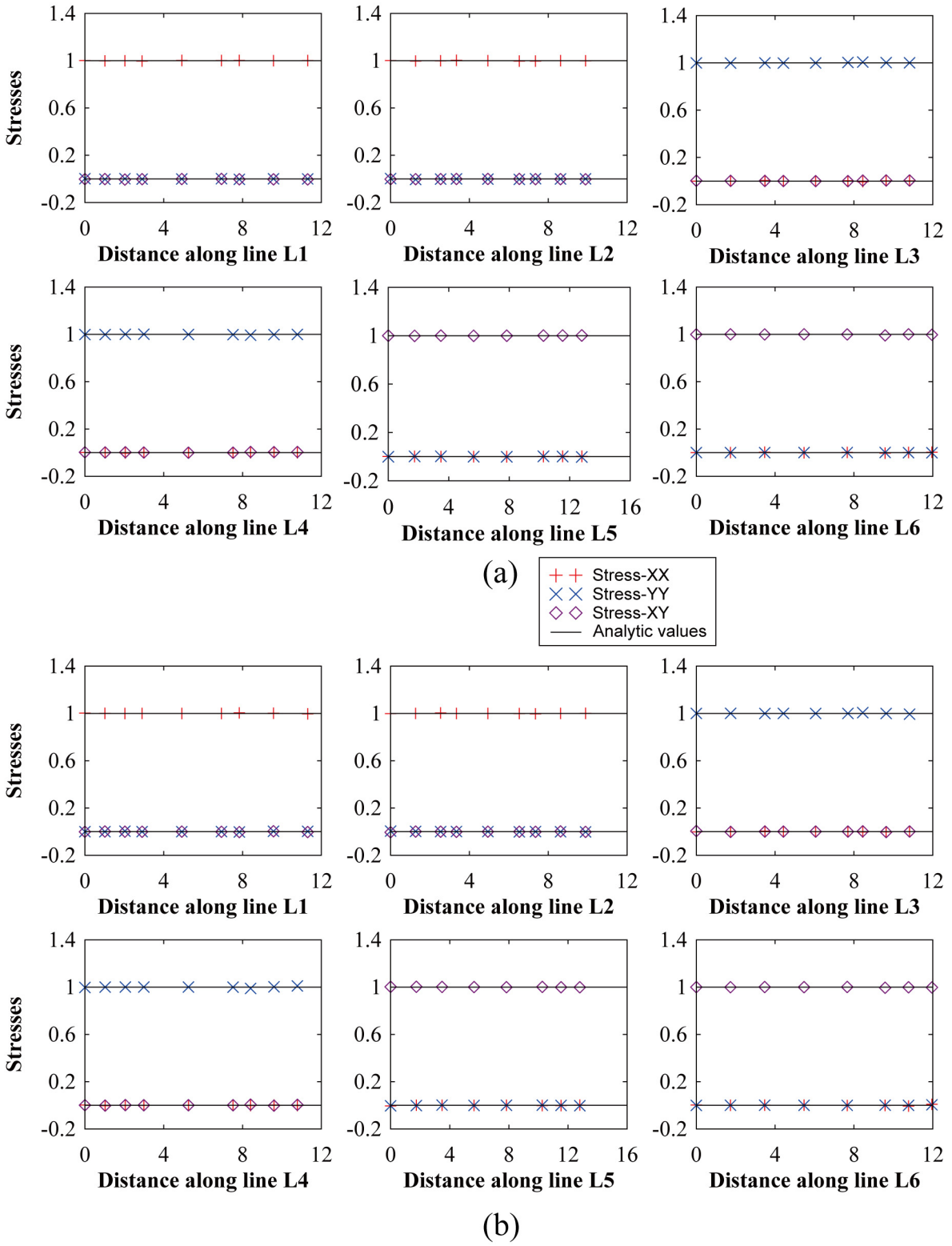

Fig. 10. Stresses along lines L1–L6 for the patch tests: (a) DL8 element. (b) DL4 element.

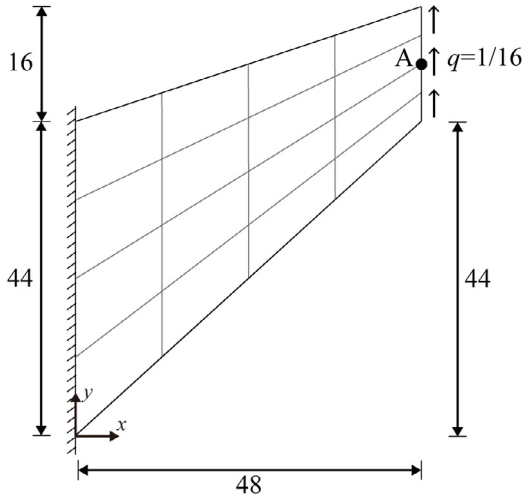

Fig. 11. Cook's skew beam problem description ($E = 1.0$, $\upsilon = 1/3$, thickness $= 1.0$).

Table 3
Normalized deflections at point A in the Cook's skew beam problem (reference solution: 23.9662).

Mesh	Quadratic elements				Linear elements			
	Q8	Q9	P183	DL8	Q4	QM6	P182	DL4
2 × 2	0.9479	0.9717	0.9668	0.9868	0.4942	0.8783	0.8783	0.8756
4 × 4	0.9892	0.9947	0.9900	0.9992	0.7635	0.9604	0.9604	0.9606
8 × 8	0.9966	0.9983	0.9967	0.9995	0.9213	0.9884	0.9884	0.9883
16 × 16	0.9987	0.9993	0.9989	0.9996	0.9776	0.9965	0.9965	0.9964
32 × 32	0.9995	0.9998	0.9996	0.9997	0.9938	0.9989	0.9989	0.9988

in which E_{ref} and E_h denote the strain energy stored in the entire structure obtained from the reference and finite element solutions, respectively [43–45].

The optimal convergence of the relative strain energy error is estimated as $E_e \cong ch^{2k}$ where c is a constant, $k = 1$ and 2 for linear and quadratic elements, respectively, and h is the element size [1]. We use $h = 1/N$ for the linear elements and $1/2N$ for the quadratic elements.

6.1. Cook's skew beam problem

The skew beam problem proposed by Cook [46] is considered. The problem description on the geometry and boundary conditions is illustrated in Fig. 11. The skew beam is subjected to a distributed shearing force of magnitude $q = 1/16$ (force per length) at the right end, and the left end is clamped. The plane stress condition is employed with Young's modulus $E = 1.0$ and Poisson's ratio $\nu = 1/3$. The skew beam is discretized by using $N \times N$ element meshes ($N = 2$, 4, 8, 16, and 32).

Table 3 shows the tip deflections (v_A) at point A (shown in Fig. 11) normalized by the reference solution. The reference solution is obtained using a 100 × 100 mesh of Q9 elements. Fig. 12 presents the convergence curves of the linear (Q4, QM6, P182 and DL4) and quadratic (Q8, Q9, P183 and DL8) elements. The DL8 element outperforms the other quadratic elements considered while the three linear elements (QM6, P182 and DL4) show almost the same convergence behavior. Fig. 13 displays the distributions of the shear stress (τ_{xy}) obtained using the DL8 element when $N = 2$, 4, and 8. The reference stress distribution shown in Fig. 13(d) is given by a 64 × 64 mesh of Q9 elements. As N increases, the stress solution of the DL8 element converges well to the reference solution.

6.2. Tapered beam problem

The tapered beam problem described in Fig. 14 is solved. The beam is subjected to the uniformly distributed load of $q = 1$ along the top side and the left side is fully clamped. The plane stress condition is considered with

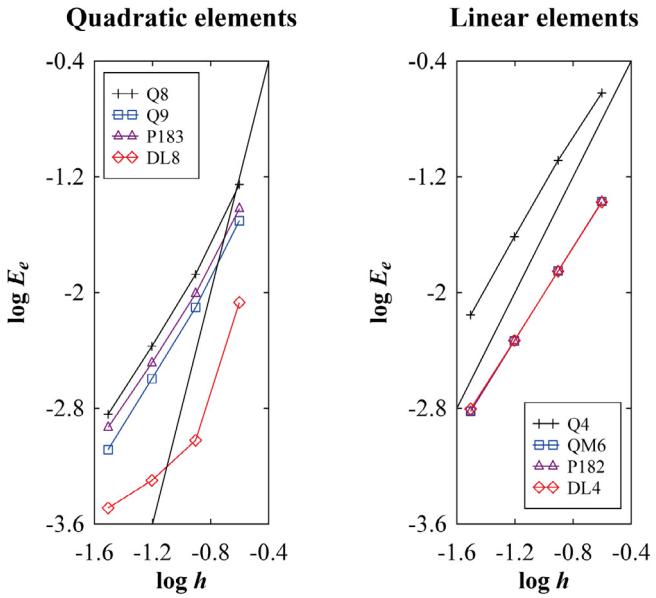

Fig. 12. Convergence curves in the Cook's skew beam problem: The bold lines represent the optimal convergence rates.

Young's modulus $E = 3.0 \times 10^2$ and Poisson's ratio $\upsilon = 0.3$. The taper beam is discretized by using $N \times 3N$ element meshes ($N = 2$, 4, 8, and 16).

Fig. 15 displays the convergence curves for the linear and quadratic elements considered. The reference solution is obtained using a 64×64 mesh of Q9 elements. The DL8 element shows an improved convergence behavior compared to the other quadratic elements.

6.3. Block problem

The block problem described in Fig. 16 is investigated. The geometry and boundary conditions are shown in Fig. 16. A uniformly distributed load of $q = 1$ is applied with 45° tilted direction on the right half of the top side. The bottom side of the structure is fully clamped. The plane stress condition with Young's modulus $E = 3.0 \times 10^7$ and Poisson's ratio $\upsilon = 0.3$ is considered.

The regular and distorted meshes are employed as shown in Fig. 17(a) and (b). The distorted meshes are obtained by randomly repositioning the interior nodes of the corresponding regular meshes. The nodal coordinates of the distorted meshes (x' and y') are determined by

$$x' = x + \beta_x h \quad \text{and} \quad y' = y + \beta_y h, \tag{25}$$

in which x and y are respectively the x- and y-nodal coordinates of the regular meshes, and β_x and β_y are uniformly generated random numbers ranging from -0.35 to 0.35.

Fig. 18 presents the convergence curves for the linear and quadratic elements. The reference solution is obtained using a 64×64 regular mesh of Q9 elements. The DL8 element shows the superior convergence behavior even in distorted meshes.

6.4. Cantilever beam problem

We consider the cantilever beam problem as shown in Fig. 19. The beam is clamped at the left end and subjected to a uniformly distributed load of $q = 1$ at the free tip. The plane stress condition is considered with Young's modulus $E = 1.0 \times 10^7$ and Poisson's ratio $\upsilon = 0.3$. Three different mesh patterns of meshes shown in Fig. 19(a)–(c) are adopted.

Table 4 shows the vertical displacements at point A normalized by the reference solution. The reference solution is obtained using a 10×60 regular mesh of Q9 elements. The Q8 element shows the performance deterioration in

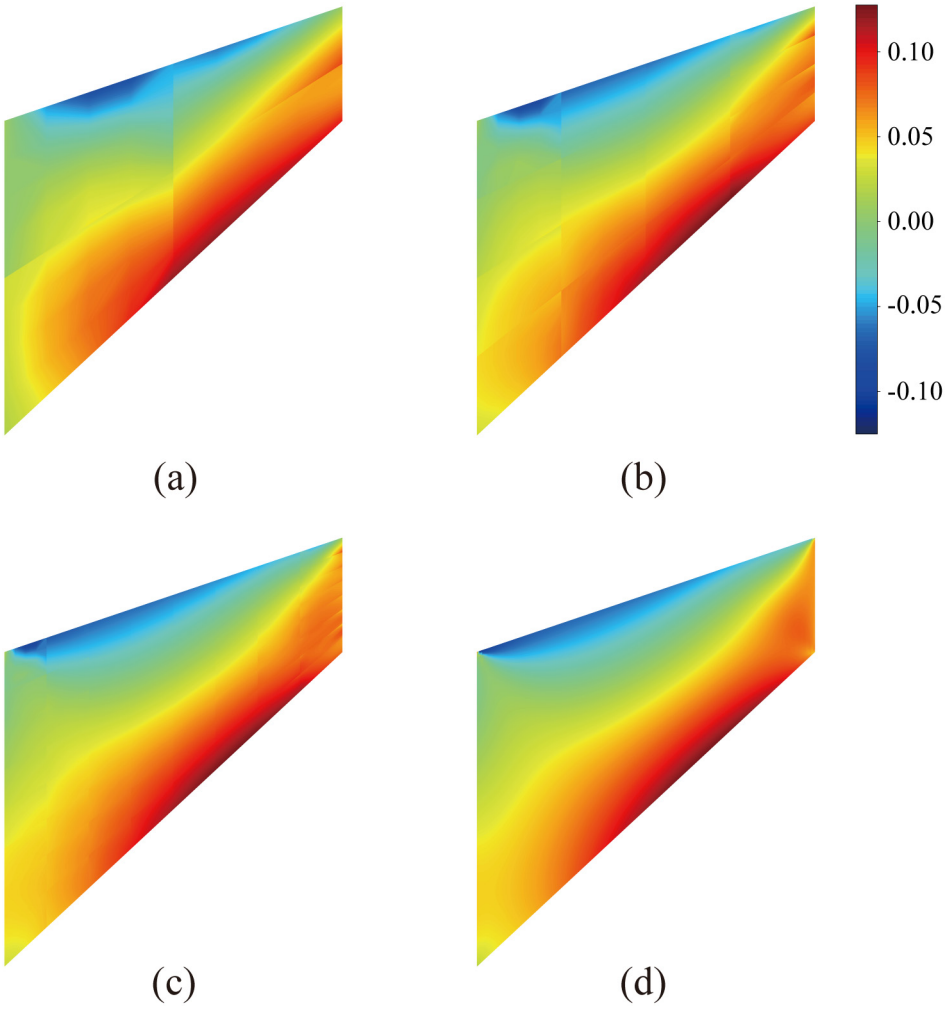

Fig. 13. Stress distributions (τ_{xy}) calculated in Cook's skew beam problem using the DL8 element: (a) $N = 2$, (b) $N = 4$, and (c) $N = 8$. (d) Reference solution obtained using the Q9 element with $N = 64$.

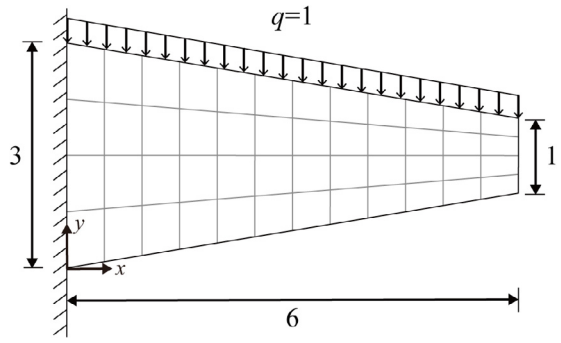

Fig. 14. Tapered beam problem ($E = 3.0 \times 10^2$, $\upsilon = 0.3$, thickness $= 0.1$).

the trapezoidal mesh pattern. The performance of the DL8 element is comparable to that of the Q9 element despite having fewer DOFs.

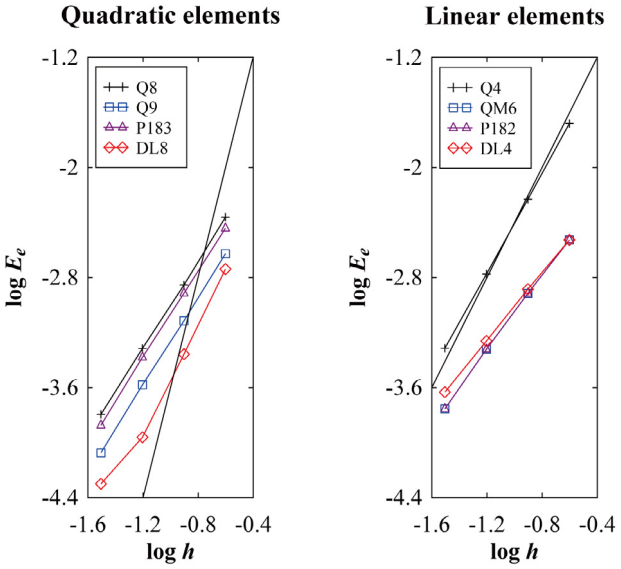

Fig. 15. Convergence curves in the tapered beam problem: The bold lines represent the optimal convergence rates.

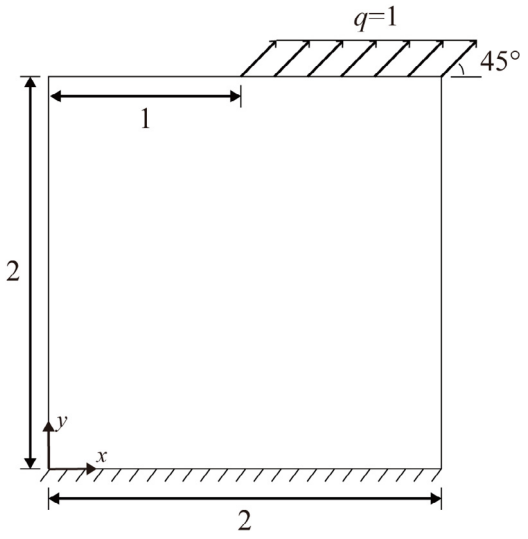

Fig. 16. Block problem ($E = 3.0 \times 10^7$, $\upsilon = 0.3$, thickness $= 1.0$).

6.5. Wrench problem

We consider the wrench problem described in Fig. 20. The geometry and boundary conditions are shown in Fig. 20. A uniformly distributed load of $q = 10^6$ is applied along the line AB. The plane stress condition is considered using Young's modulus $E = 2.0 \times 10^{11}$ and Poisson's ratio $\upsilon = 0.3$. Three different meshes are considered, as shown in Fig. 20(a)–(c). The reference solution is obtained using Q9 elements and the mesh in Fig. 20(d).

Fig. 21 displays the convergence curves for the quadratic and linear elements. Fig. 22 shows errors in the vertical displacement ($E_d = \left|(v - v_{ref})/v_{ref}\right|$) along the line AB shown in Fig. 20. The results show that the DL8 and DL4 elements perform very well.

7. Computational efficiency

The computation cost of the proposed deep learned finite elements (DL8 and DL4) is assessed through the Cook's skew beam problem described in Section 6.1. The computation times taken from obtaining the stiffness matrices to

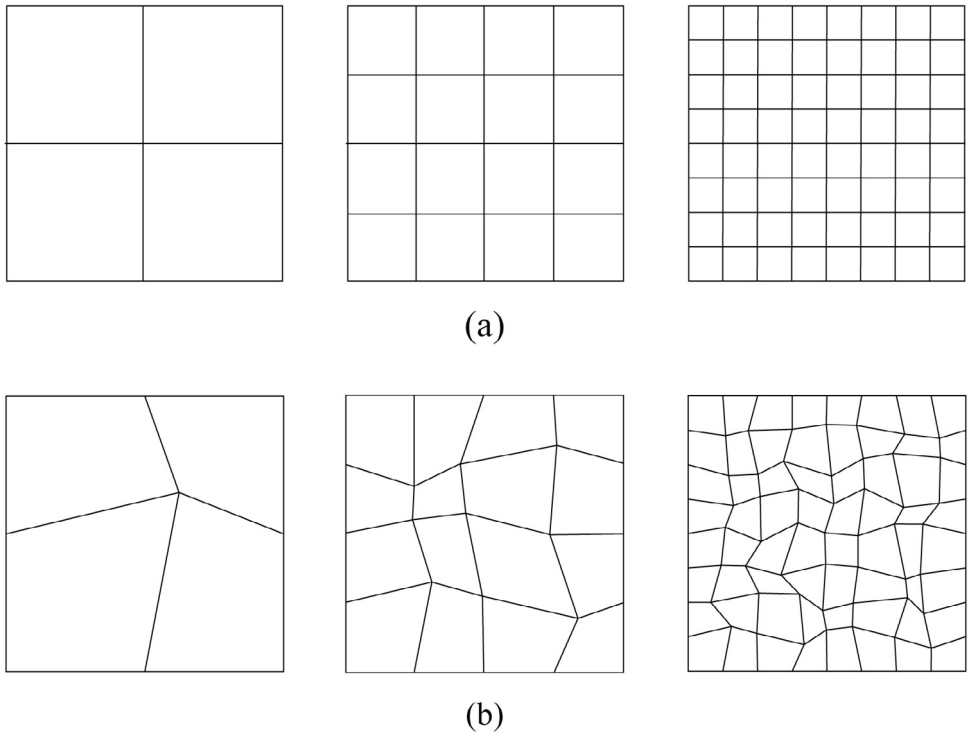

Fig. 17. Meshes used for the block problem: (a) Regular meshes used with $N = 2$, 4 and 8. (b) Distorted meshes used with $N = 2$, 4 and 8.

Table 4
Normalized vertical displacements at point A in the cantilever beam problem (reference solution: -3.4694×10^{-3}).

Mesh	Quadratic elements				Linear elements			
	Q8	Q9	P183	DL8	Q4	QM6	P182	DL4
Rectangular	0.9861	0.9935	0.9877	0.9862	0.3796	0.9937	0.9937	1.0115
Trapezoidal	0.8984	0.9877	0.9704	0.9940	0.1351	0.2064	0.2064	0.2080
Parallelogram	0.9888	0.9898	0.9997	0.9866	0.1492	0.7932	0.7932	0.7888

solving the linear equations are measured. All calculations were performed using a quad-core workstation (Intel(R) Core (TM) i7-2600 CPU @ 3.80 GHz, 12 GB memory, Microsoft Windows 10 64 bit) under Python environment. The linear equations were solved using a direct solver in NumPy library [47].

Fig. 23 displays the relations between the computation time versus solution accuracy relative errors in the energy norm in Eq. (24). Regular meshes with $N = 16$, 32, and 64 are used for the assessment. In Cook's skew beam problem, the DL8 element outperforms in the aspect of computational efficiency among the tested elements. That is, the DL8 element uses less computation time compared to other quadratic elements when obtaining similar solution accuracy. Also, this element produces more accurate solutions compared to other quadratic elements for similar computation time used. However, the computational efficiency of the DL4 element is not as good as that of the QM6 element.

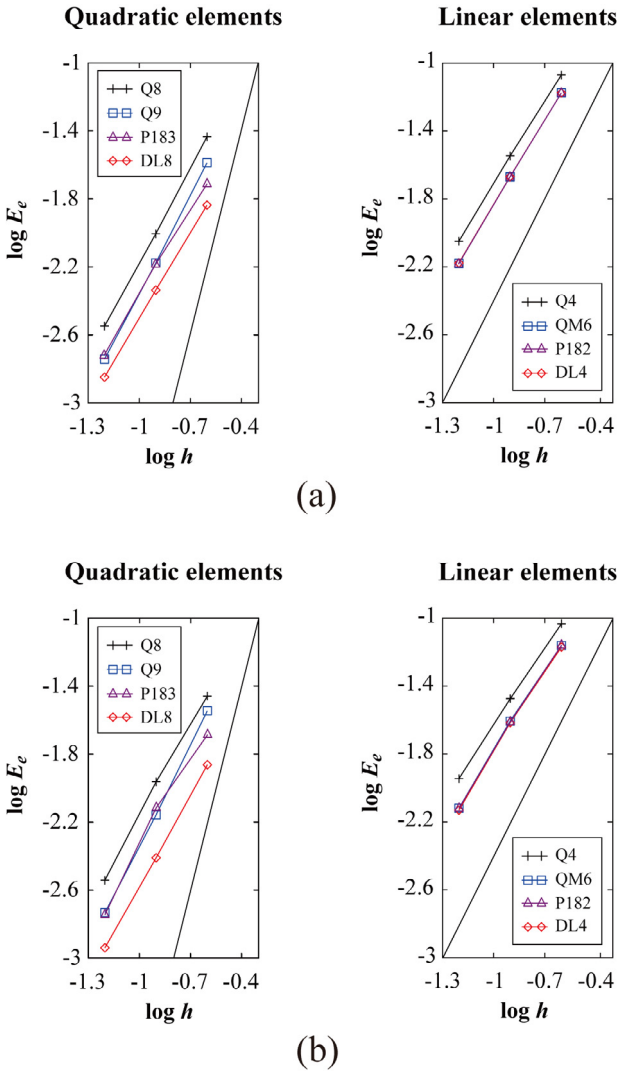

Fig. 18. Convergence curves in the block problem in (a) regular meshes and (b) distorted meshes: The bold lines represent the optimal convergence rates.

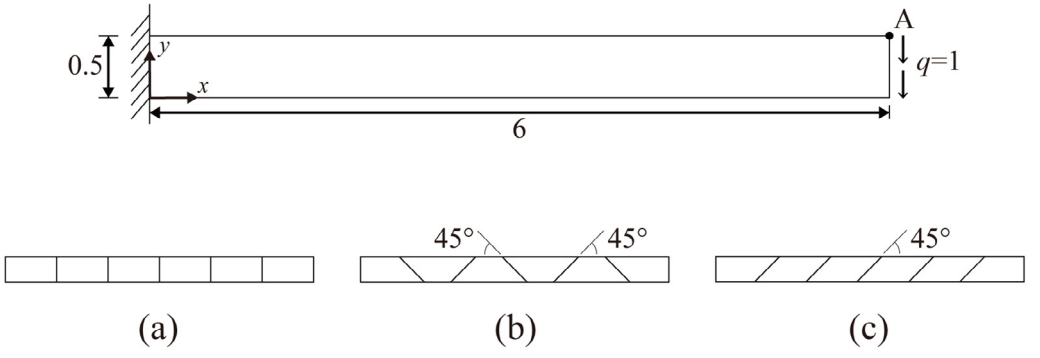

Fig. 19. Cantilever beam problem ($E = 1.0 \times 10^7$, $\upsilon = 0.3$, thickness $= 0.1$): (a) Regular mesh. (b) Trapezoidal mesh. (c) Parallelogram mesh.

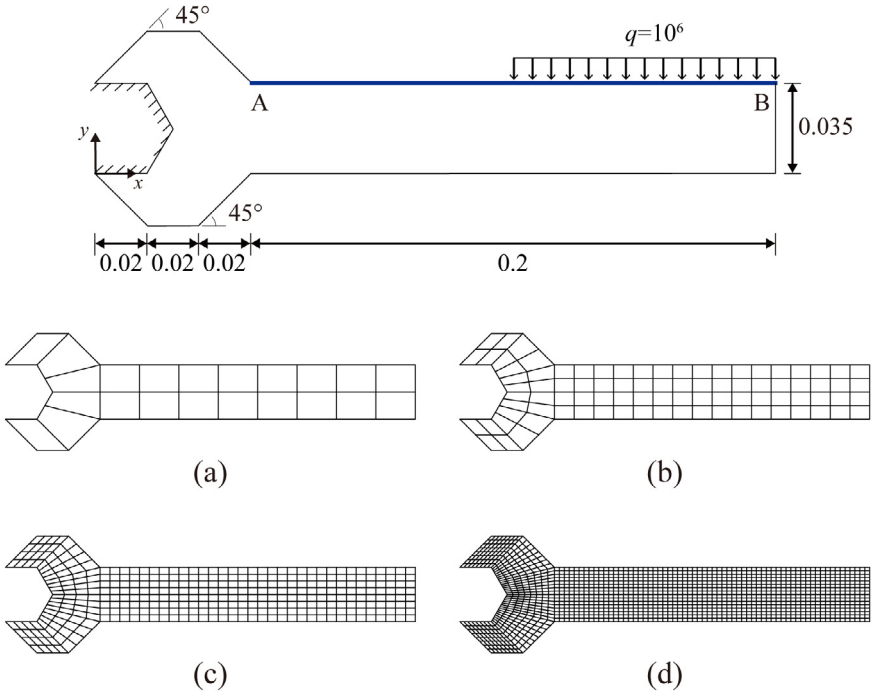

Fig. 20. Wrench problem ($E = 2.0 \times 10^{11}$, $\upsilon = 0.3$, thickness $= 0.01$): (a) Coarse mesh ($N = 2$). (b) Medium mesh ($N = 4$). (c) Fine mesh ($N = 8$). (d) Mesh used for the reference solution.

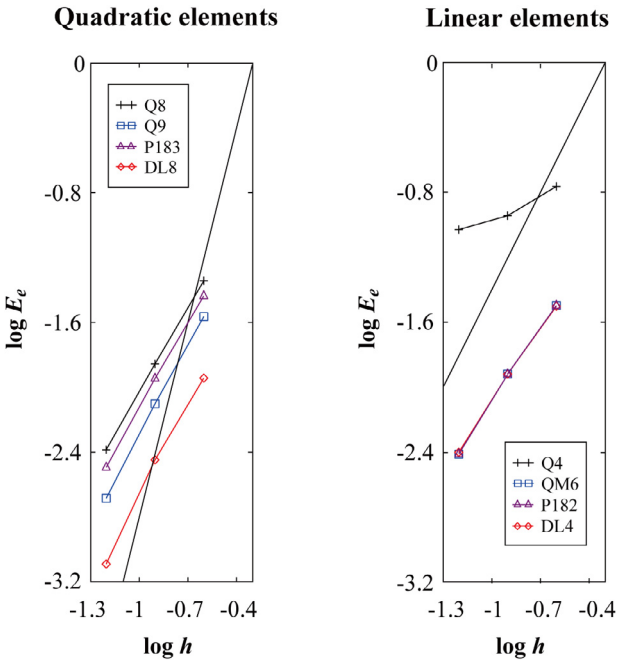

Fig. 21. Convergence curves in the wrench problem: The bold lines represent the optimal convergence rates.

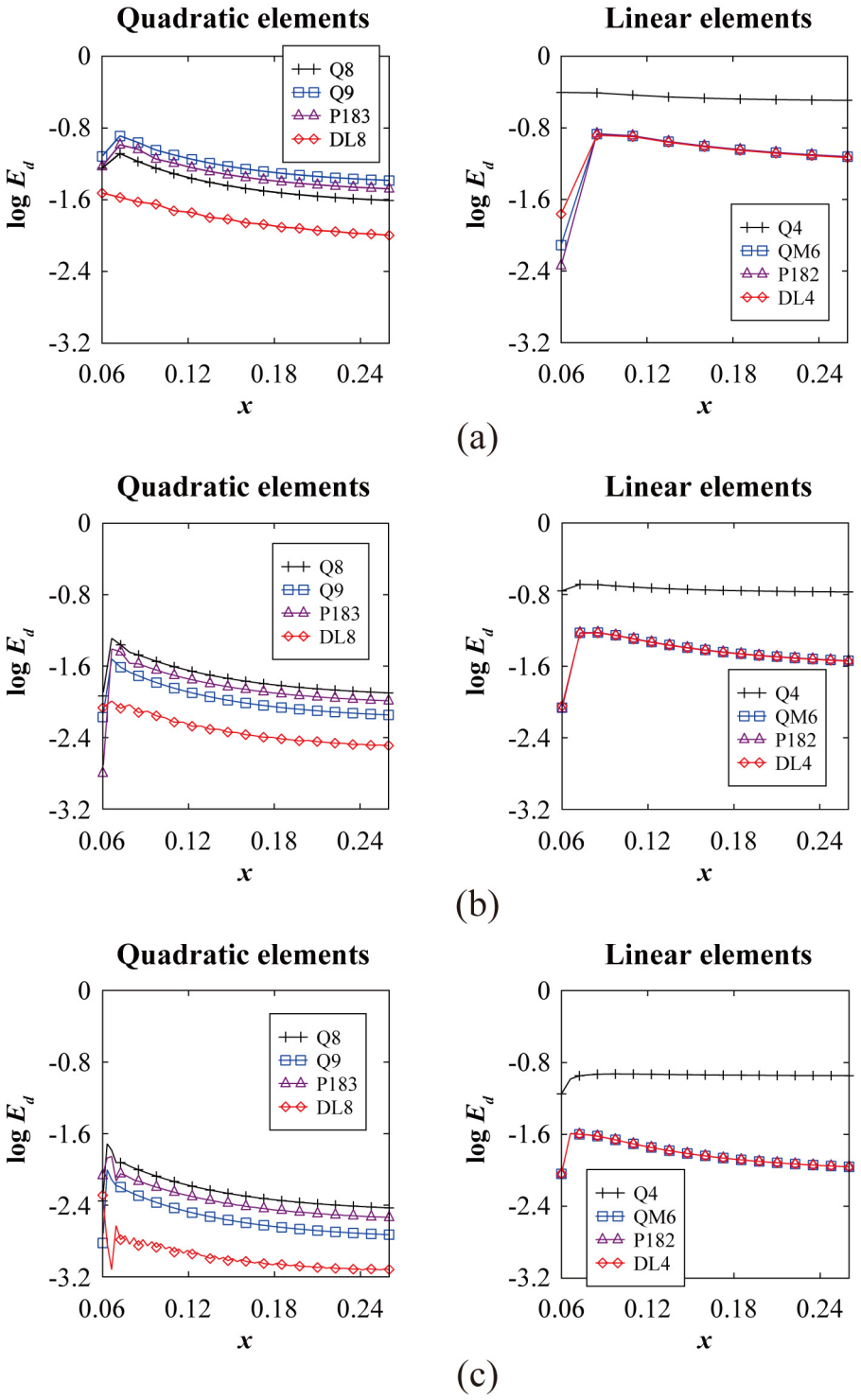

Fig. 22. Errors in the vertical displacement along the line AB in the wrench problem: (a) Coarse mesh. (b) Medium mesh. (c) Fine mesh.

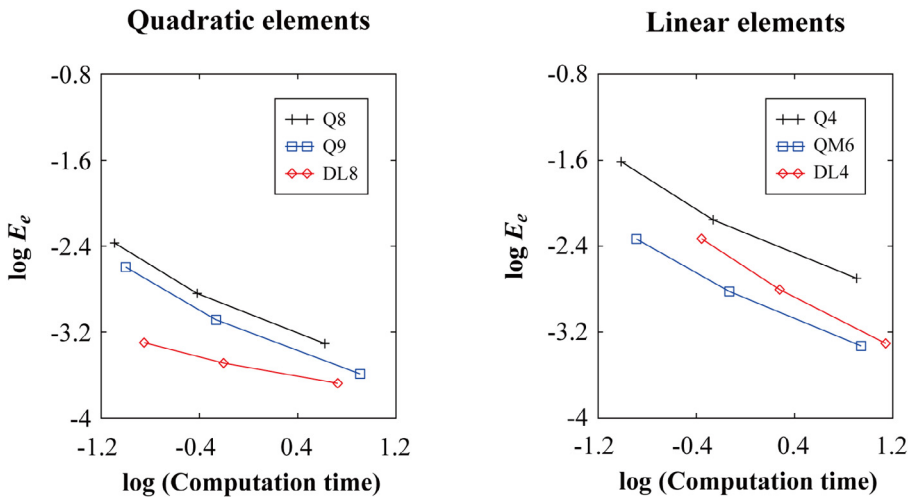

Fig. 23. Computational efficiency curves in the Cook's skew beam problem. The computation times are measured in seconds.

8. Concluding remarks

In this paper, we proposed a methodology to generate stiffness matrices of finite elements using deep learning. The deep learned 8- and 4-node quadrilateral elements were developed. This study presented various new concepts and processes: normalized element geometry, reference data model for the training data, pre-processing for the input, and post-processing for the output. We also proposed a way to make finite elements better represent rigid body motions and constant strain fields. The performance of the DL8 and DL4 elements was evaluated through various numerical examples. In particular, the DL8 element showed promising ability in both accuracy and computational efficiency.

This study has great implications by showing that artificial intelligence can be used for finite element development. The proposed method is not limited to the 2D quadrilateral solid elements applied in this study. It can be extended to various finite elements, including 3D solid, beam, and shell finite elements with different shapes and orders [48–54]. Of course, applying the method proposed in this paper to nonlinear analysis is also very valuable. The computation time of the neural network is proportional to the number of weights used [30]. Further improvement of the computational efficiency can be achieved by reducing the number of weights and optimizing the network structure.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Acknowledgments

This work was supported by the National Research Foundation of Korea (NRF) grant funded by the Korean government (MSIT) (No. NRF-2018R1A2B3005328). This work was also supported by the "Human Resources Program in Energy Technology" of the Korea Institute of Energy Technology Evaluation and Planning (KETEP), granted financial resource from the Ministry of Trade, Industry & Energy, Republic of Korea (No. 20184030202000). This research was the result of a study on the "HPC Support" Project, supported by the Ministry of Science and ICT (MSIT) and National IT Industry Promotion Agency (NIPA). We also thank Dr. Yonggyun Yu at Korea Atomic Energy Research Institute (KAERI) for his valuable comments.

Table A.1
Averaged errors of the trained neural networks according to the mesh density of the reference data model (N).

N	Training data error (%)	Test data error (%)
10	1.57	2.68
30	1.55	1.98
50	1.24	1.67

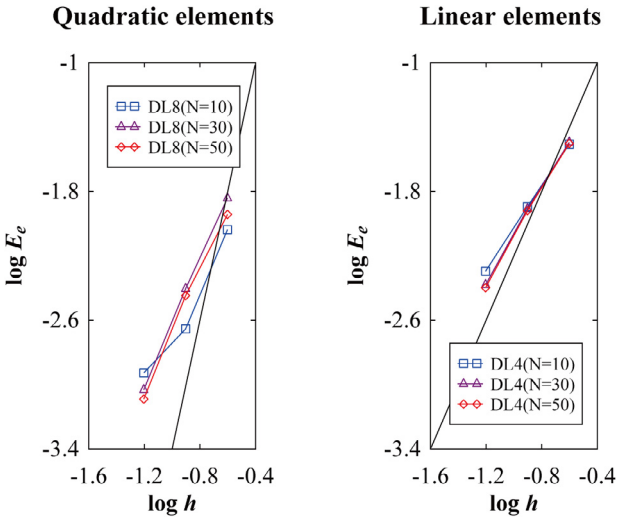

Fig. A.1. Convergence curves according to the mesh density of the reference data model (N) in the wrench problem The bold lines represent the optimal convergence rates.

Appendix A. Effect of material properties on the internal displacements

We here investigate the effect of two material properties (i.e., Young's modulus and Poisson's ratio) on the internal displacements $\hat{\mathbf{u}}_{\mathrm{I}}^{(n)}$ calculated by the prescribed outer displacements $\hat{\mathbf{u}}_{\mathrm{O}}^{(n)}$. The internal displacement vector $\hat{\mathbf{u}}_{\mathrm{I}}^{(n)}$ is calculated using the matrices $\hat{\mathbf{K}}_{\mathrm{II}}$ and $\hat{\mathbf{K}}_{\mathrm{IO}}$ in Eq. (10).

The material law matrix $\mathbf{C}^{(n)}$ is a function of Young's modulus (E) and Poisson's ratio (υ) represented by

$$\mathbf{C}^{(n)}(E,\upsilon) = E\bar{\mathbf{C}}^{(n)}(\upsilon), \tag{A.1}$$

in which $\bar{\mathbf{C}}^{(n)}$ is a function of Poisson's ratio.

Then, the following equation can be derived

$$\hat{\mathbf{K}}_{\mathrm{II}} = E\bar{\mathbf{K}}_{\mathrm{II}} \quad \text{and} \quad \hat{\mathbf{K}}_{\mathrm{IO}} = E\bar{\mathbf{K}}_{\mathrm{IO}}, \tag{A.2}$$

in which $\bar{\mathbf{K}}_{\mathrm{II}}$ and $\bar{\mathbf{K}}_{\mathrm{IO}}$ are matrices independent of Young's modulus.

Substituting Eq. (A.2) into Eq. (10), the internal displacement vector $\hat{\mathbf{u}}_{I}^{(n)}$ is obtained regardless of Young's modulus (E):

$$\hat{\mathbf{u}}_{\mathrm{I}}^{(n)} = -\left(\bar{\mathbf{K}}_{\mathrm{II}}\right)^{-1}\bar{\mathbf{K}}_{\mathrm{IO}}\hat{\mathbf{u}}_{\mathrm{O}}^{(n)}. \tag{A.3}$$

Appendix B. Mesh density of the reference data model

The deep learned finite elements are based on the reference data model; thus, their performance depends on the mesh density of the reference data model. Here, we study the dependency considering three reference data models with various mesh densities $N = 10$, 30, and 50. The three neural networks corresponding to the mesh densities were obtained via the procedure described in Sections 3.1 and 3.2.

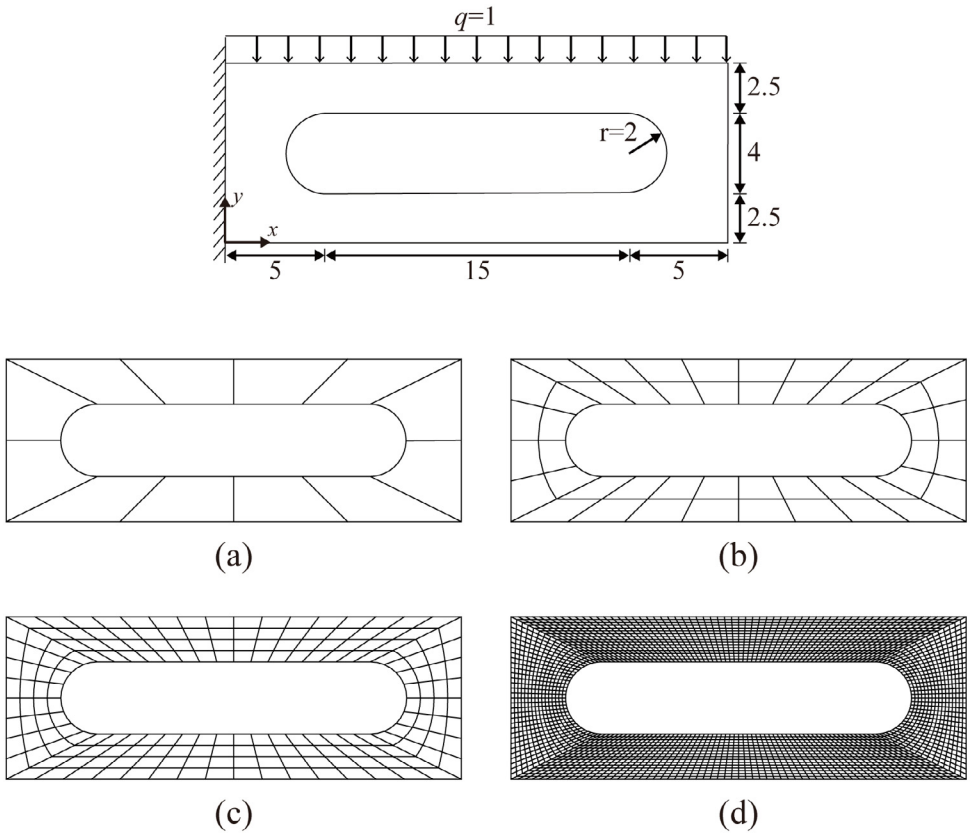

Fig. A.2. Tool zig problem ($E = 2.0 \times 10^{11}$, $\upsilon = 0.3$, thickness $= 2.0$): (a) Coarse mesh ($N = 2$). (b) Medium mesh ($N = 4$). (c) Fine mesh ($N = 8$). (d) Mesh used for the reference solution.

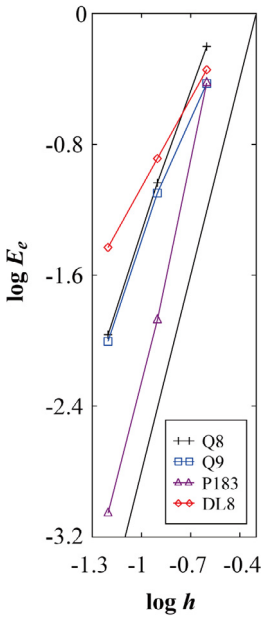

Fig. A.3. Convergence curves in the tool zig problem: The bold line represents the optimal convergence rates.

Table A.1 represents the training and test data errors of the trained neural networks. The use of the fine mesh reference data model (i.e., $N = 50$) leads to less error compared to that of the coarse mesh reference data model ($N = 10$ and 30). Fig. A.1 shows the convergence curves of the DL8 and DL4 elements generated from the three reference data models in the wrench problem illustrated in Fig. 20.

Appendix C. Convergence behavior of the DL8 element in a curved geometry model

As mentioned in Section 3.1, the geometry of the DL8 element is limited to a quadrilateral whose mid-side nodes are placed at the center of the adjacent corner nodes and thus curved geometries were not trained for the element. Nevertheless, it is highly interesting to investigate the convergence behavior of the DL8 element when modeling a curved geometry.

Herein, we consider the tool zig problem described in Fig. A.2. The geometry and boundary conditions are shown in Fig. A.2. A uniformly distributed load of $q = 1$ is applied along the top side. The plane stress condition is considered using Young's modulus $E = 2.0 \times 10^{11}$ and Poisson's ratio $\upsilon = 0.3$. Three different meshes are considered, as shown in Fig. A.2(a)–(c). The reference solution is obtained using the Q9 elements and the mesh in Fig. A.2(d).

Fig. A.3 shows the convergence curves for the quadratic elements. As expected, the DL8 element does not exhibit a good convergence behavior, compared to other quadratic elements.

References

[1] K.J. Bathe, Finite Element Procedures, Prentice Hall, Upper Saddle River, NJ, 2006.
[2] J.M. Jin, The Finite Element Method in Electromagnetics, John Wiley & Sons, Hoboken, NJ, 2015.
[3] P.M. Gresho, R.L. Sani, Incompressible Flow and the Finite Element Method. Volume 1: Advection-Diffusion and Isothermal Laminar Flow, John Wiley & Sons, Hoboken, NJ, 1998.
[4] P. Lesaint, P.A. Raviart, On a Finite Element Method for Solving the Neutron Transport Equation, Academic Press, Cambridge, MA, 1974.
[5] J. Donea, S. Giuliani, J.P. Halleux, An arbitrary Lagrangian-Eulerian finite element method for transient dynamic fluid–structure interactions, Comput. Methods Appl. Mech. Engrg. 33 (1982) 689–723.
[6] J.L. Volakis, A. Chatterjee, L.C. Kempel, Finite Element Method Electromagnetics: Antennas, Microwave Circuits, and Scattering Applications, John Wiley & Sons, Hoboken, NJ, 1998.
[7] J.N. Reddy, D.K. Gartling, The Finite Element Method in Heat Transfer and Fluid Dynamics, CRC press, 2010.
[8] V. Girault, P.A. Raviart, Finite Element Approximation of the Navier–Stokes Equations, in: Lecture Notes in Mathematics, Berlin Springer Verlag, 1979.
[9] W.S. McCulloch, W. Pitts, A logical calculus of the ideas immanent in nervous activity, Bull. Math. Biophys. 5 (1943) 115–133.
[10] F. Rosenblatt, The perceptron: a probabilistic model for information storage and organization in the brain, Psychol. Rev. 65 (1958) 386.
[11] G.E. Hinton, S. Osindero, Y.W. Teh, A fast learning algorithm for deep belief nets, Neural Comput. 18 (2006) 1527–1554.
[12] D.E. Rumelhart, G.E. Hinton, R.J. Williams, Learning representations by back-propagating errors, Nature 323 (1986) 533–536.
[13] A. Krizhevsky, I. Sutskever, G.E. Hinton, Imagenet classification with deep convolutional neural networks, in: Advances in Neural Information Processing Systems, 2012, pp. 1097–1105.
[14] D. Silver, A. Huang, C.J. Maddison, A. Guez, L. Sifre, G. Van Den Driessche, J. Schrittwieser, I. Antonoglou, V. Panneershelvam, M. Lanctot, S. Dieleman, D. Grewe, J. Nham, N. Kalchbrenner, I. Sutskever, T. Lillicrap, M. Leach, K. Kavukcuoglu, T. Graepel, D. Hassabis, Mastering the game of Go with deep neural networks and tree search, Nature 529 (2016) 484.
[15] D. Silver, J. Schrittwieser, K. Simonyan, I. Antonoglou, A. Huang, A. Guez, T. Hubert, L. Baker, M. Lai, A. Bolton, Y. Chen, T. Lillicrap, F. Hui, L. Sifre, G. van den Driessche, T. Graepel, D. Hassabis, Mastering the game of Go without human knowledge, Nature 550 (2017) 354.
[16] J. Sirignano, K. Spiliopoulos, DGM: a deep learning algorithm for solving partial differential equations, J. Comput. Phys. 375 (2018) 1339–1364.
[17] M. Raissi, Deep hidden physics models: deep learning of nonlinear partial differential equations, J. Mach. Learn. Res. 19 (2018) 932–955.
[18] M. Raissi, P. Perdikaris, G.E. Karniadakis, Physics-informed neural networks: a deep learning framework for solving forward and inverse problems involving nonlinear partial differential equations, J. Comput. Phys. 378 (2019) 686–707.
[19] J. Tompson, K. Schlachter, P. Sprechmann, K. Perlin, Accelerating Eulerian fluid simulation with convolutional networks, in: Proceedings of the 34th International Conference on Machine Learning, Vol. 70, 2017, pp. 3424–3433.
[20] X. Guo, W. Li, F. Iorio, Convolutional neural networks for steady flow approximation, in: Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, 2016, pp. 481–490.
[21] O. Hennigh, Lat-net: compressing lattice Boltzmann flow simulations using deep neural networks, 2017, arXiv Preprint arXiv:1705.09036.

[22] J. Ling, A. Kurzawski, J. Templeton, Reynolds averaged turbulence modelling using deep neural networks with embedded invariance, J. Fluid Mech. 807 (2016) 155–166.
[23] Z.J. Zhang, K. Duraisamy, Machine learning methods for data-driven turbulence modeling, in: 22nd AIAA Computational Fluid Dynamics Conference, 2015, p. 2460.
[24] A.D. Beck, D.G. Flad, C.-D. Munz, Deep neural networks for data-driven turbulence models, 2018, arXiv Preprint arXiv:1806.04482.
[25] J. Takeuchi, Y. Kosugi, Neural network representation of finite element method, Neural Netw. 7 (1994) 389–395.
[26] L. Liang, M. Liu, C. Martin, W. Sun, A deep learning approach to estimate stress distribution: a fast and accurate surrogate of finite-element analysis, J. R. Soc. Interface 15 (2018) 20170844.
[27] A. Chamekh, H.B.H. Salah, R. Hambli, Inverse technique identification of material parameters using finite element and neural network computation, Int. J. Adv. Manuf. Technol. 44 (2009) 173.
[28] Y.M. Hashash, S. Jung, J. Ghaboussi, Numerical implementation of a neural network based material model in finite element analysis, Internat. J. Numer. Methods Engrg. 59 (2004) 989–1005.
[29] D. Chen, D.I. Levin, S. Sueda, W. Matusik, Data-driven finite elements for geometry and material design, ACM Trans. Graph. 34 (2015) 74.
[30] A. Oishi, G. Yagawa, Computational mechanics enhanced by deep learning, Comput. Methods Appl. Mech. Engrg. 327 (2017) 327–351.
[31] O.C. Zienkiewicz, R.L. Taylor, The Finite Element Method: The Basis, Butterworth-Heinemann, Oxford, UK, 2000.
[32] K. Hornik, M. Stinchcombe, H. White, Multilayer feedforward networks are universal approximators, Neural Netw. 2 (1989) 359–366.
[33] N. Le Roux, Y. Bengio, Deep belief networks are compact universal approximators, Neural Comput. 22 (2010) 2192–2207.
[34] M. Abadi, P. Barham, J. Chen, Z. Chen, A. Davis, J. Dean, M. Devin, S. Ghemawat, G. Irving, M. Isard, M. Kudlur, J. Levenberg, R. Monga, S. Moore, D.G. Murray, B. Steiner, P. Tucker, V. Vasudevan, P. Warden, M. Wicke, Y. Yu, X. Zheng, TensorFlow: a system for large-scale machine learning, in: 12th USENIX Symposium on Operating Systems Design and Implementation, 2016, pp. 265–283.
[35] D.P. Kingma, J. Ba, Adam: a method for stochastic optimization, 2014, arXiv Preprint arXiv:1412.6980.
[36] X. Glorot, Y. Bengio, Understanding the difficulty of training deep feedforward neural networks, in: Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics, 2010, pp. 249–256.
[37] D. Roylance, Transformation of Stresses and Strains, in: Lecture Notes for Mechanics of Materials, 2001.
[38] O.C. Zienkiewicz, R.L. Taylor, The Finite Element Method: Basic Formulation and Linear Problems, McGraw-Hill, New York, NY, 1989.
[39] Y. Ko, P.S. Lee, K.J. Bathe, The MITC4+ shell element and its performance, Comput. Struct. 169 (2016) 57–68.
[40] C. Lee, P.S. Lee, The strain-smoothed MITC3+ shell element, Comput. Struct. 223 (2019) 106096.
[41] R.L. Taylor, P.J. Beresford, E.L. Wilson, A non-conforming element for stress analysis, Internat. J. Numer. Methods Engrg. 10 (1976) 1211–1219.
[42] P.C. Kohnke, ANSYS Theory Reference: Release 5.5, ANSYS, Inc., 1998.
[43] K.J. Bathe, P.S. Lee, Measuring the convergence behavior of shell analysis schemes, Comput. Struct. 89 (2011) 285–301.
[44] P.S. Lee, K.J. Bathe, The quadratic MITC plate and MITC shell elements in plate bending, Adv. Eng. Softw. 41 (2010) 712–728.
[45] Y. Ko, Y. Lee, P.S. Lee, K.J. Bathe, Performance of the MITC3+ and MITC4+ shell elements in widely-used benchmark problems, Comput. Struct. 193 (2017) 187–206.
[46] R.D. Cook, D.S. Malkus, M.E. Plesha, R.J. Witt, Concepts and Applications of Finite Element Analysis, John Wiley & Sons, Hoboken, NJ, 2007.
[47] S.V. Walt, S.C. Colbert, G. Varoquaux, The NumPy array: a structure for efficient numerical computation, Comput. Sci. Eng. 13 (2011) 22–30.
[48] K. Yoon, P.S. Lee, Nonlinear performance of continuum mechanics based beam elements focusing on large twisting behaviors, Comput. Methods Appl. Mech. Engrg. 281 (2014) 106–130.
[49] Y. Lee, P.S. Lee, K.J. Bathe, The MITC3+ shell finite element and its performance, Comput. Struct. 138 (2014) 12–23.
[50] Y. Ko, P.S. Lee, K.J. Bathe, A new 4-node MITC element for analysis of two-dimensional solids and its formulation in a shell element, Comput. Struct. 192 (2017) 34–49.
[51] S. Kim, P.S. Lee, New enriched 3D solid finite elements: 8-node hexahedral, 6-node prismatic, and 5-node pyramidal elements, Comput. Struct. 216 (2019) 40–63.
[52] H.M. Jeon, Y. Lee, P.S. Lee, K.J. Bathe, The MITC3+ shell element in geometric nonlinear analysis, Comput. Struct. 146 (2015) 91–104.
[53] D.N. Kim, K.J. Bathe, A triangular six-node shell element, Comput. Struct. 87 (2009) 1451–1460.
[54] M.L. Bucalem, K.J. Bathe, Higher-order MITC general shell elements, Internat. J. Numer. Methods Engrg. 36 (1993) 3729–3754.