Journal of Thermal Analysis and Calorimetry
https://doi.org/10.1007/s10973-020-09875-6

Using deep learning to learn physics of conduction heat transfer

Mohammad Edalatifar[1] · Mohammad Bagher Tavakoli[1] · Mohammad Ghalambaz[2,3] · Farbod Setoudeh[4]

Received: 9 February 2020 / Accepted: 22 May 2020

Abstract
In the present study, an advanced type of artificial intelligence, a deep neural network, is employed to learn the physic of conduction heat transfer in 2D geometries. A dataset containing 44,160 samples is produced by using the conventional finite volume method in a uniform grid of 64 × 64. The dataset includes four geometries of the square, triangular, regular hexagonal, and regular octagonal with random sizes and random Dirichlet boundary conditions. Then, the dataset of the solved problems was introduced to a convolutional Deep Neural Network (DNN) to learn the physics of 2D heat transfer without knowing the partial differential equation underlying the conduction heat transfer. Two loss functions based on the Mean Square Errors (MSE) and Mean of Maximum Square Errors (MMaSE) are introduced. The MMaSE is a new loss function, tailored for the physic of heat transfer. The 70%, 15%, and 15% of images are used for training DNN, testing DNN, and validation of the DNN during the training process, respectively. In the validation stage, the 2D domain with random boundary conditions, in which DNN has never seen them before, is introduced to DNN. Then, DNN is asked to estimate the temperature distribution. The results show that the DNNs are capable of learning physical problems without knowing the underlying fundamental governing equation. The error analysis for various training methods is reported and discussed. The outcomes reveal that DNNs are capable of learning physics, but using MMaSE as a tailored loss function could improve the training quality. A DNN trained by MMaSE provides a better temperature distribution compared to a DNN trained by MSE. As the 2D heat equation is a Laplace equation, which is practical in multiple physics, the results of the present study indicate a new direction for future computational methods and advanced modeling of physical phenomena, using a big dataset of observations.

Keywords Conduction heat transfer · Deep convolutional neural networks · Deep learning · Laplace equation · Large dataset

List of symbols

T	Temperature (K)
T_i	Isothermal boundary conditions at the boundaries of the domain (K)
x	x-Cartesian coordinate (m)
y	y-Cartesian coordinate (m)
T	Temperature field, actual temperature distribution image (target images)
P	Predicted temperature distribution image (K)
N	Number of images in a collection of images
R	Number of rows in an image
C	Number of columns in an image

Greek symbols

β	Parameters of Adam optimizer
$\beta 1$	First parameter of Adam optimizer
$\beta 2$	Second parameter of Adam optimizer

✉ Mohammad Bagher Tavakoli
m-tavakoli@iau-arak.ac.ir

Mohammad Edalatifar
m.edalatifar@gmail.com

Mohammad Ghalambaz
mohammad.ghalambaz@tdtu.edu.vn

Farbod Setoudeh
f.setoudeh@arakut.ac.ir

[1] Department of Electrical Engineering, Arak Branch, Islamic Azad University, Arak, Iran

[2] Metamaterials for Mechanical, Biomechanical and Multiphysical Applications Research Group, Ton Duc Thang University, Ho Chi Minh City, Vietnam

[3] Faculty of Applied Sciences, Ton Duc Thang University, Ho Chi Minh City, Vietnam

[4] Faculty of Electrical Engineering, Arak University of Technology, Arak, Iran

Published online: 31 July 2020

Subscripts

i	The segment of the boundary condition
n	Sigma index for summation on a collection of images
r	Sigma index for summation on all rows
c	Sigma index for summation on all columns

Abbreviations

SE	Square Error
MSE	Mean Square Errors
MMaSE	Mean of Maximum Square Errors
MaSE	Maximum of Square Errors
AE	Absolute Error
MAE	Mean Absolute Errors
MMaAE	Mean of Maximum Absolute Errors
MaAE	Maximum of Absolute Errors

Introduction

The two-dimensional steady-state heat conduction is a simple typical problem in the context of heat transfer designs. The Laplace equation introduces the conduction heat transfer in the form of $\nabla^2 T = 0$ where T is the temperature distribution. The Laplace equation not only represents the conduction heat transfer phenome but also introduces much more physical phenomena such as pressure distribution in a fluid or mass diffusion.

Although the Laplace equation is a simple partial differential equation, there is not a general analytical solution for this equation. The availability of any analytical solution depends on the geometry of the heat transfer domain and the boundary conditions. Despite the lack of analytical solutions, there are various numerical approaches, which can easily solve the heat equation. Well-known methods such as the finite difference method, finite volume method, finite element method, or the meshless methods can deal with this problem robustly. However, numerical methods generally discretize the domain of solution into subdomains (grid or mesh), and then they construct an algebraic equation for each subdomain. Finally, the set of algebraic equations has to be solved either analytically or iteratively. Solving the set of algebraic equations is a computationally costly procedure, mainly when the number of grid points (algebraic equations) is high.

Moreover, for each new problem, the set of algebraic equations has to be solved from an initial guess, and the knowledge of the calculations cannot be passed to a new domain and problem. It means that for each different problem, the algebraic equations shall be contracted and solved individually. The iterative numerical approaches are typical for solving heat equations. Such methods have been utilized in many engineering applications. Using interactive numerical methods, various aspects of heat transfer such as heat transfer in phase change materials for domestic applications [1], heat transfer of nanomaterials in porous spaces [2], heat transfer in channels [3] and helical pipes [4], heat transfer in nanofluids [5], and heat transfer in biofluids [6] have been investigated in recent years.

In recent years, Deep Neural Networks (DNNs) have shown enormous ability to extract features and generate images, using many hidden layers. Until 2006, the available neural networks suffer a lack of many hidden layers due to inadequate techniques to train the many hidden layers. However, Hastad and Goldmann [7] demonstrated the importance of an increased number of layers in a neural network.

In contrast to conventional neural networks, the deep neural networks are capable of producing realistic solutions and achieve state-of-the-art computational performance by using a data-driven approach [8]. Hinton et al. [9] represented the restricted Boltzmann machine (RBM) method, which was an unsupervised layer-by-layer learning algorithm. After a short time, Bengio et al. [10] introduced a process to train multilayer deep networks. They used RBM as a pre-training method before starting supervised training process. The RBM acts as a regularizer and initial parameters of the network [11]. After that, deep networks were used to classification [12], regression [13], dimensional reduction [14], and many other issues.

The classification and generation are two main types of DNN applications. Image recognition and classification [15], speech recognition [16], face recognition [17], sentence classification [18], material recognition [19], and person re-identification [20] are samples of the DNN applications for classification and recognition. In the field of generation, the text-to-image synthesis [21], speech enhancement [22], and image-to-image transactions [23] are some of the applications. DNNs can be employed to physical phenomena by using observed or simulated data to learn the physical behavior where the actual physical model is complicated or unknown [8].

There are various types of DNNs, which benefit from a large number of hidden layers in their structures. The Convolutional Neural Networks (CNNs) are a well-known and powerful type of DNNs, inspired by biological models. The CNNs have been used for pattern recognition tasks such as handwritten numeral recognition and face recognition [24]. Moreover, CNNs are capable of automatic extractions of the salient features with a certain degree of shape distortions and shifts for the input characters [25].

A CNN consisted of an input layer, several intermediate layers (hidden layers), and an output layer. In a CNN, the pixels of a picture are delivered to an input layer and passes to the intermediate layers like a flow until they reach the

output layer. The structure of the intermediate layers can be designed based on the application of CNNs.

Since the introduction of CNNs, many approaches have been developed to improve the capability and accuracy of these neural network, or accelerating training process, i.e., adding batch normalization between layers [26], using Rectified Linear Unit (ReLU) [27] as the activation function, and initializing the masses of a network with a random normal distribution [28, 29]. Moreover, many DNNs with convolutional layers and unique structures such as AlexNet [30], GoogleNet [31], Generative Adversarial Networks (GAN) [32], and ResNet [33] were suggested to increase the accuracy of CNNs.

Very recently, the use of deep networks has been considered for the estimation of heat and fluid transfer issues. Sharma et al. [34] used a fully convolutional encoder–decoder network adapted from the U-Net architecture [35] to estimate temperature distribution over a flat square plate with random temperatures on the bound of the plate. Based on the Finite Difference Method (FDM) [36], they convolved a special filter with the output of the network to determine error value for each point on the plate. Farimani et al. [8] suggested an extension of GAN, Conditional Generative Adversarial Networks (cGAN) [37], to estimate heat transfer. GAN has been successfully used for texture mapping, style transferring, text-to-image translation, and image-to-image translation in previous publications [32]. Authors trained cGAN network with a dataset containing 6230 training samples, generated by numerical FDM, for various temperature boundary conditions, two-dimensional geometries (annulus, disk, triangle, and rectangle), different domain sizes, and domain position within a 64×64 grid domains.

The dataset produced by Sharma et al. [34] contains only a square heat boundary with random heats on each side, and hence, the dataset was limited in terms of diversity of geometry and boundary conditions. Training a deep model without the need for a big actual dataset was the main advantage of their work. With this technique, they could produce data during the training process and utilize the advantage of a big dataset. However, by using this technique, there is no unique dataset available for future examinations. Meanwhile, Farimani et al. [8] studied the conduction heat transfer by using a small dataset, consisted of only three bounded shapes.

Regardless of the dataset, almost all previous works used Mean Square Error (MSE) as a loss function to train their models. However, there are many other error estimators for training a deep neural network. An adequate selection of an error estimator can notably affect the resolution of an expert network for a specific task. Investigation of the effect of different error estimators is one of the objectives of the present study.

As mentioned, a partial differential equation is required to introduce the physic of heat transfer, and then a numerical approach is needed to solve the partial differential equation. However, there are many applications in which the exact governing equation may be unknown. Moreover, the numerical approach shall be repeated from an initial guess and for each different problem. It means that the solution of a heat transfer problem produces much of knowledge about the behavior of heat equation, which could not be passed to another problem with different geometry and boundary conditions. In a very recent study, Raissi et al. [38] introduced a framework to teach DNNs the physics of Navier–Stokes equations from an image database computational flow field over a cylinder in a channel. The results demonstrate the capability of DNNs in learning the hidden physics of Navier–Stokes equations from the images. However, as the DNNs just learn the physics of Navier–Stokes equations, obtaining a flow field requires computational steps as usual.

The present work aims to teach a DNN to learn the behavior of the heat transfer through a big dataset of heat transfer images with no requirement of a computational step, as was required in the study of Raissi et al. [38]. In this approach, a DNN just looks at many solved heat problems and then learns the behavior of the heat transfer. Indeed, the knowledge of many previously solved heat problems will be transferred into a DNN during a training process, without exposing the actual differential equation of the heat transfer to the DNN. Then, the expert (the trained) DNN will be utilized to solve new heat transfer problems directly with no iteration or construction of algebraic equations. Hence, the advantage of the present method is that it omits the computational iterative step of solving the governing equations for the temperature field. The input of the expert DNN will be a geometry with boundary conditions, and the output will be the image of temperature distribution. The present study is an early attempt to teach DNN the physics of a transport phenomenon using a big dataset of heat transfer images without revealing the underlying differential equation. Another contribution of the present study is producing a public benchmark dataset for general training of future deep neural network models.

Mathematical model and systematic analysis

The present study aims to teach a DNN to learn the physics of 2D conduction heat transfer. Here, an overall view of the present approach will be summarized in four steps. Then, each step of the work will be discussed in detail.

First, a large database of images, representing the temperature distribution in 2D geometries, is required. This database shall contain various geometries with different geometry sizes and different temperature boundary conditions.

The diversity of this database helps DNN to see various situations and solutions of temperature distribution in a 2D domain. Therefore, each initial image shall contain a geometry (the domain of solution) and a defined boundary condition to maintain the diversity of the dataset. To aim this purpose, a code for the generation of various geometries in terms of shapes and sizes, as well as random boundary conditions, is developed. Then, the produced 2D heat transfer problem can be solved by using conventional numerical methods. The numerical solution of the heat transfer problem for each image produces the temperature distribution in that image. Later, these images can be fed to the DNN as training, testing, and validation stages.

In the second step, a structure of DNN shall be defined. Indeed, the DNN is an extensive function with many adjustable parameters and interconnected functions. The structure of a DNN can be designed for a specific task.

Then, the third step is introducing an error estimation indicator, and the fourth step is using a mathematical method, such as the gradient descent, that could be employed to adjust the variables of the DNN. The variables of DNN shall be adjusted to minimize an error function. Hence, introducing an error estimation indicator is an essential step in the present study. The training of DNN can be started by using the dataset of images, the defined structure for the DNN, and the error estimator. The train of DNN is an iterative procedure that tends to reduce the estimation error of the network gradually.

A well-trained DNN is expected with the ability to provide an accurate temperature distribution for any arbitrary given heat conduction problem. The final step of the current work is exploring the capabilities and accuracy of the DNN for the heat transfer problems that DNN has never seen before. Thus, bearing these above steps in mind, the first step is producing a large dataset of 2D images of the temperature distribution, which is the subject of the next subsection.

Dataset

Each sample in this dataset contains a two-channel image as the input and a one-channel image as the output. Both input and output images have 64×64 pixels. The first input channel is used to introduce the boundaries of the geometry, which can adopt the values of 0.5, 1, and 0. The value of 0.5 indicates a pixel at the boundary, the value of 1 denotes a pixel inside of the geometry (the domain of solution), and the value of 0 shows a pixel outside of the geometry.

A sample of the first input channel is depicted in Fig. 1a. As seen, inside each figure is yellow, denoting the value of 1; the outside is dark, indicating 0. There is a narrow green border, which adopted the value of 0.5, and it denotes the boundaries. The second input channel contains the information on the boundary conditions, which

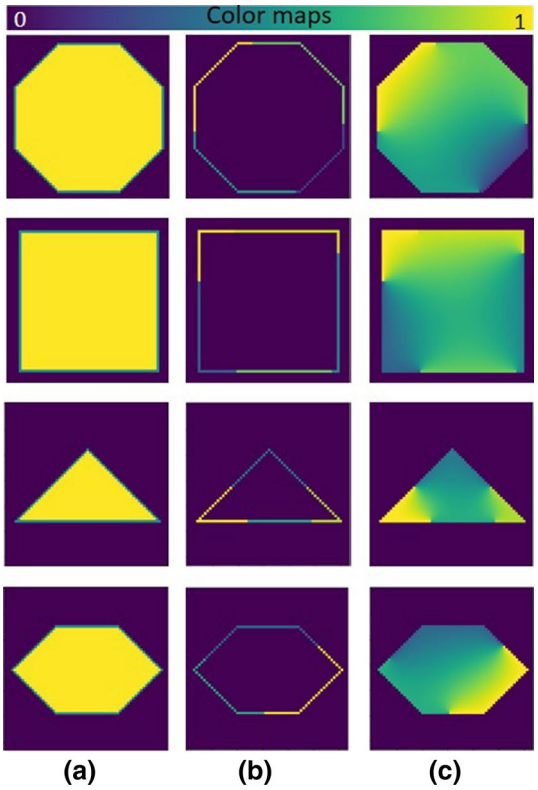

Fig. 1 Some samples of each image channel: **a** input channel for outside, boundary, and inside of domain with value of 0, 0.5, and 1, respectively; **b** input channel for temperature boundary condition; **c** the output with the temperature distribution inside the geometry and null (zero) outside the geometry

are the boundary temperatures. This channel is depicted in Fig. 1b. Inside and outside of the boundary domain is zero, and the numerical value on the boundary denotes the temperature of the boundary, which is normalized in the scale of zero to one. It is assumed that each boundary can consist of two to five random segments with random temperature values in the range of zero to one. For example, the first row in Fig. 1 shows a hexagonal, in which the inside, outside, and the boundary of the geometry are clearly defined. Figure 1b shows that there are four temperature segments (note that the temperature segments can be randomly between two and five random segments), and the value of each segment is defined with color.

Figure 3c illustrates the output image, containing the boundary and the distribution of the temperature inside the geometry (the domain of solution). The outside of the boundary in the output image is filled with zeros as it is outside of the domain of the solution. For example, the first row of Fig. 1c shows the corresponding temperature distribution for the given boundary conditions. As seen, the temperature of the domain, in the vicinity of each boundary, is close to the boundary's temperature and then

changes with the increase in distance from that boundary. Various approaches, such as Successive Local Linearization Method (SLLM) [39], can be employed to solve the governing equations numerically. Here, the Finite Volume Method (FVM) is employed to solve the temperature distribution in each image with the residual accuracy of 10^{-6}. The details of the computation of temperature distribution will be discussed later.

The variables of the present study are the investigated geometries and location of isothermal boundary condition segments. The variation of geometry and location of imposed boundary conditions help DNN to learn the physics of the heat transfer in various domains and under various configurations.

The geometries of the dataset include square, triangular, regular hexagonal, and regular octagonal shapes, and the width of all shapes is between 35 and 58 pixels. The size of the geometries can be varied randomly, and the height of the triangular geometry is half of its width. The maximum limit of 58 pixels ensures that the geometry fits in the image frame of 64 pixels, as it is assumed all of the images are square with a size of 64 pixels.

As mentioned, the first and second input channels are utilized to introduce the domain of solution and the required boundary conditions. The following partial differential equation is utilized to obtain the temperature distribution:

$$\frac{\partial^2 T}{\partial x^2} + \frac{\partial^2 T}{\partial y^2} = 0 \tag{1}$$

subject to the $T = T_i$ at each segment of the boundary of the geometry where i is the segment of the boundary condition, and T_i is the temperature of the segment. The partial differential equation, Eq. 1, is numerically solved using the FVM. The details of conduction heat transfer and the FVM for solving this equation are well described in fundamental heat transfer textbooks [40, 41]. Therefore, the temperature distribution images which are utilized as the output images of the dataset indeed are the solution of Eq. (1).

The FVM was employed to obtain the temperature distribution in the given geometries and boundary conditions and generated a dataset containing 44,160 sample images. This extensive dataset, which includes various geometries and boundary conditions, is a benchmark dataset. The utilized FVM code for solving Eq. (1), and the corresponding temperature distributions, all of the database images, and other DNN codes are available here: https://doi.org/10.17632/rw9yk3c559.1.

The input images along with the calculated output temperature distributions are a benchmark dataset which can be employed for testing of various aspects of DNN in learning and estimation of heat transfer phenome. In the present study, the dataset images are divided into three categories, in which 15% of the dataset images are randomly extracted as testing data and 15% percent of the remaining images are randomly selected as validation data. The remaining images are saved as the training data. The validation data were used to draw and investigate the evaluation parameters of the training process as well as to determine the best epoch of the training process with a minimal loss value.

The bound width of all shapes of the dataset is between 35 and 58 pixels. In order to examine the generalization capability of the trained DNN, a set of unseen data is also produced. The unseen data have never been introduced to DNN in any step of training or testing. The unseen dataset contains 1920 samples with a bandwidth of 30 pixels. After the training process, both testing and unseen data are given to the network and extracted evaluation parameters and plot charts. The result of these data shows the generalization ability of networks. The testing data are a part of the training process and test the learning quality of the DNN; however, the unseen data are not a part of train or test, and their purpose is the validation of DNN to how well a trained DNN can estimate the temperature distribution in an unseen and completely fresh problem.

Deep Neural Network (DNN) structure

A deep neural network consists of a structure and adjustable (trainable) parameters, layers, and connections. The number of layers and connections can be selected following available well-known structures, or it can be designed by trial and error or a combination of literature works and modifications by trial and error. There are several enormous models of the DNNs, which have been employed for classification and regression tasks.

An autoencoder is a type of intelligence neural networks, which is designed for multi-purpose, i.e., dimensional reduction, image denoising, and generate images [42]. In the present study, the inputs of the network are the images of the domain and boundary conditions, while the outputs are the temperature distribution in the domain. Hence, the autoencoder neural networks are adopted as the main structure of the DNN. A schematic figure of the utilized autoencoder DNN, adopted in the present study, is illustrated in Fig. 2. As seen, an autoencoder is made of three major parts, encoder, decoder, and bottleneck. The encoder extracts features and reduces the dimensions of the input images. It also encodes the input data. A bottleneck is a compressed representation of the input data. Decoder reconstructs the output data from the encoded data of the encoder. The layers of autoencoder can be either fully connection layers or convolutional layers [43]. An autoencoder with convolution layers, a convolutional autoencoder, is adopted in the present study as the

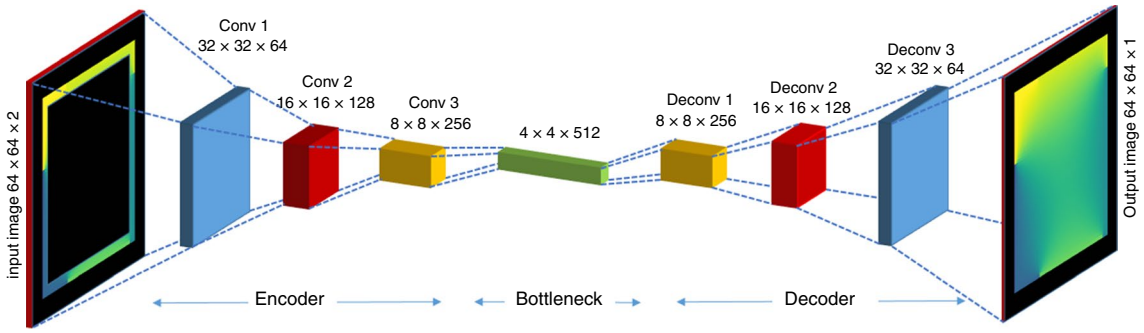

Fig. 2 An example of an autoencoder

main structure of DNNs. However, some modifications are also employed in the structure of autoencoder to increase its potential in learning the physics of heat transfer. The adopted DNN for the present study is depicted in Fig. 4. The layers of the autoencoder are replaced with convolutional residual blocks.

He et al. [33] proposed the residual networks as a new learning framework for classification purposes. They used an individual block, residual block, instead of a cascade of network layers, which is common in typical neural networks. The structure of the residual block is illustrated in Fig. 3a. The advantage of using the residual blocks was that

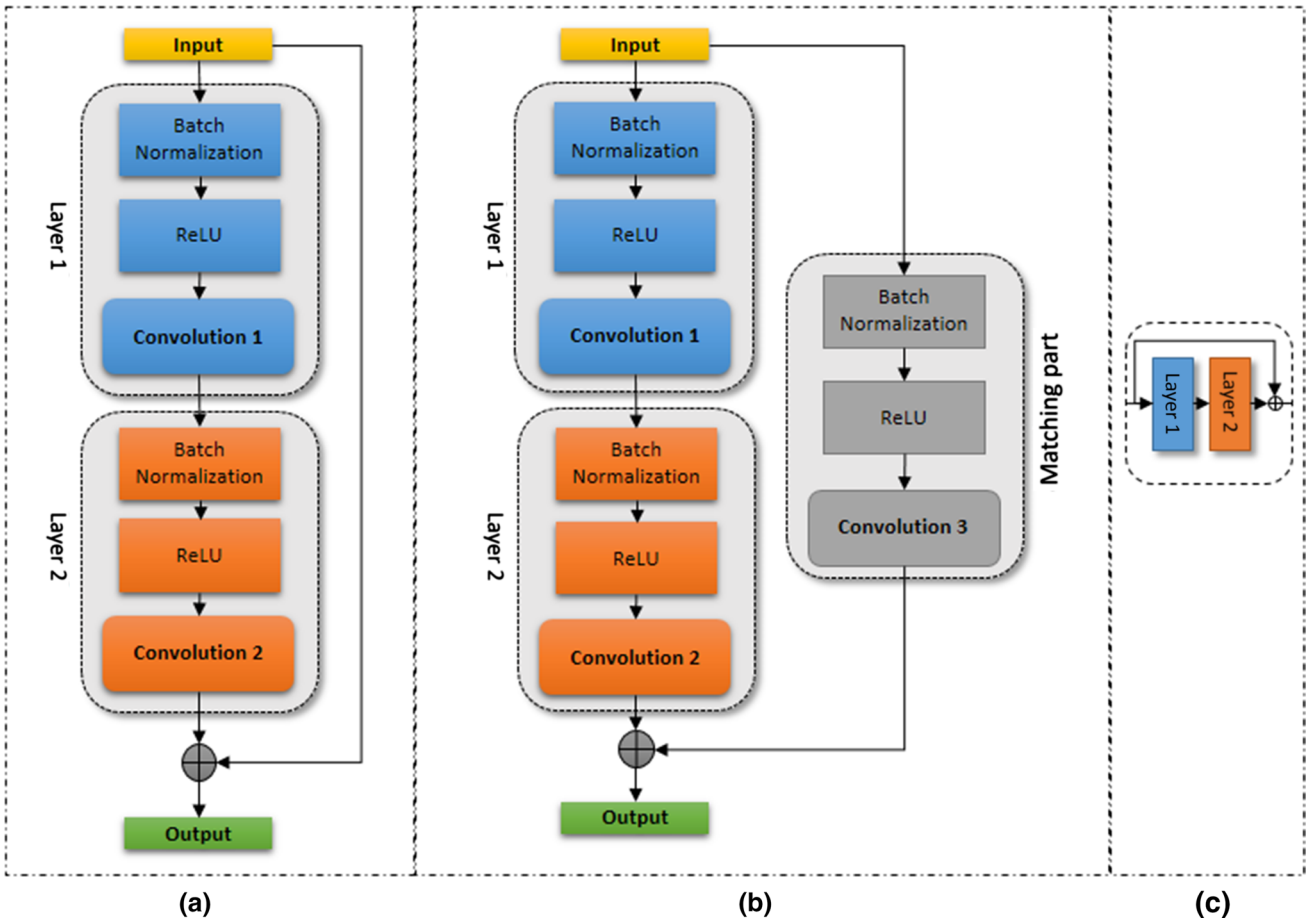

Fig. 3 A structure of a convolutional residual block with shortcuts, **a** simple shortcut for a residual block with the same size of input and output features, **b** a shortcut with a matching block layer for a residual block with different input and output sizes of features maps, **c** a compact view of the a convolutional residual block for convenience

it could replace two consequence layers of a network. These authors achieved a state-of-the-art classification accuracy on the well-known ImageNet dataset by using the residual networks in a network, known as ResNet [15]. Comparing to similar networks with ordinary layers, the ResNet eases the optimization by providing faster convergence at the early stage, exhibiting considerably lower training error, converging faster in general.

It should be noted that using too many layers in a conventional network does not always improve the learning process or the learning error. Indeed, over-increasing the depth of a conventional neural network can unexpectedly elevate the learning error. This is a well-known general issue in typical neural networks as the degradation problem [44]. A ResNet network has solved the degradation problem by using the residual blocks. Hence, by using the residual blocks in the structure of the ResNet network, the learning error will improve or remain constant.

As shown in Fig. 3a, a residual block consists of two cascades of layers that output of the block is the element-wise summation of the first input of layer (input of block) and the second output of layer. The input of a residual block is transferred to the input of the element-wise summation by a shortcut connection. A batch normalization layer and an activation layer are placed before each layer of the block. The batch normalization technique improves the speed, performance, and stability of a network. The frequent activation function for residual block networks is the ReLU activation function, which is followed by a batch normalization layer. Other activation functions, such as tangent hyperbolic and sigmoid, are also possible. However, a computational complexity of computations using a ReLU activation function is minimal compared to other activation functions. Moreover, the convergence rate of a network consisting of ReLU activation functions is much higher than that of the networks with tangent hyperbolic and sigmoid activation functions [30]. The shortcut connection can be a simple connection transferring the input feature to the output of the block to be added to the output data, as shown in Fig. 3a. It can also be a transfer block that acts as a matching part and matches the input and output sizes of a block in case there is a size difference between the input and output of the block. The matching part is depicted in Fig. 3b.

The layers of a residual block can be fully connected or convolutional. The layers of the residual block, depicted in Fig. 2a, are convolutional layers. Hence, the produced residual block is known as a convolutional residual block. Each convolutional layer consists of several filters. The filters are regular matrixes with a typical size of 3×3 or 5×5 for images. These matrixes contain trainable parameters (values), which have to be adjusted during the learning process. The input of a convolutional layer is the input data of a network or the output features of the previous layer, while its output is the extracted features of the layer.

The ResNet is usually utilized for image classifications and is capable of extracting the various types of features from an input image. As the purpose of the present study is to produce the temperature distribution images from the input images, a combination of an autoencoder and residual blocks is employed to benefit from the residual block capability of ResNet and image generation structure of the autoencoder neural network. As a result, the conventional cascade layers of the autoencoder network are replaced by the residual blocks. It is expected that utilizing the residual

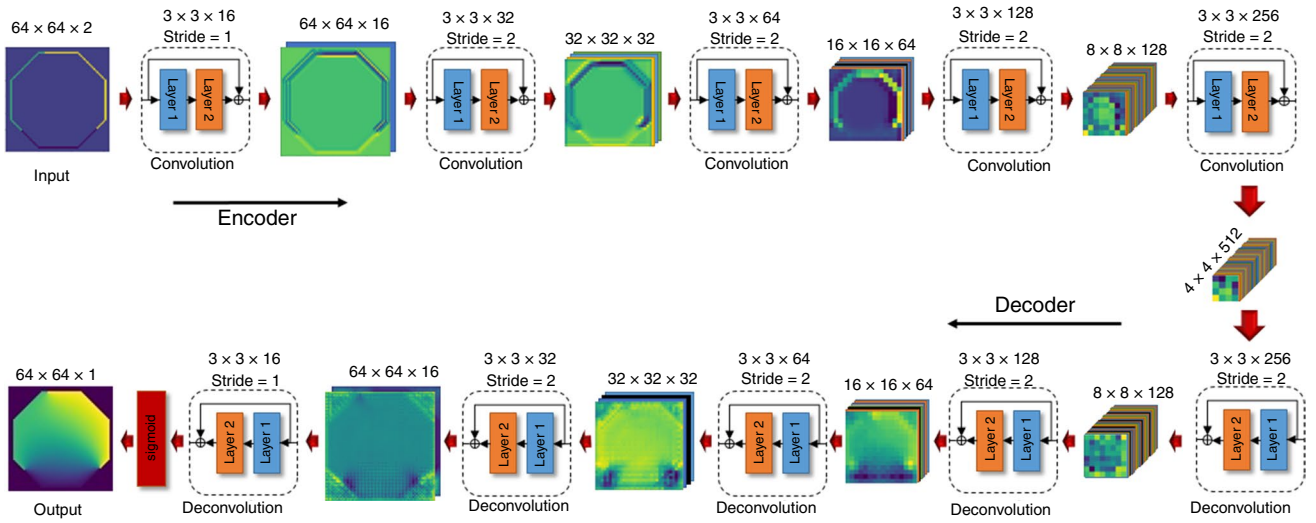

Fig. 4 Structure and characteristics of the DNN utilized in the current study. Stride 1 represents a simple shortcut as shown in Fig. 3a, in which the size of input and output features of the block is equal, and stride 2 indicates a shortcut with a matching layer as depicted in Fig. 3b, in which the size of the output features of a block is half of its input features

blocks in the autoencoder framework improves the accuracy of the image processing and the learning process. Therefore, each of the blocks of the autoencoder of Fig. 2 is replaced by the residual blocks of Fig. 3, which resulted in a deep neural network. The schematic figure of the deep neural network, utilized in the present study, is depicted in Fig. 4.

As shown in Fig. 4, each of the decoder and encoder of the utilized DNN consists of five residual blocks. The residual blocks of the encoder consist of convolutional layers, while the residual blocks of the decoder layer consist of deconvolutional layers. The deconvolutional layers act as a reverse function of the convolutional layers. Except for the first residual block of the encoder layer, each residual block reduces the size of its input feature to half and increases the number of the features by twofold. In the decoder section, in a reverse manner, all of the residual blocks, except for the last residual block, increase the size of the input feature by twofold and reduce the number of the features by half. Therefore, the input images undergo a process of downscaling and extracting features in the encoder part of the neural network. Then, the extracted features are used for upscaling and producing the final temperature distribution in the decoder part of the neural network.

The size of the encoder output is 512 features, in which each of the features has a size of 4×4. The output of the last block of the decoder is passed through a sigmoid activation function to generate an output image. The output of a sigmoid activation function is in the rage of zero to one which is the range of the possible values of the temperature field. Hence, the sigmoid activation function through a nonlinear function maps the output of the last block of the network to an image of temperature distribution.

It is possible to adopt a larger number of residual blocks in a DNN; however, increasing the number of residual blocks increases the number of trainable parameters, which computationally is not desirable. The DNN of the present study, which is depicted in Fig. 4, involves 1,857,554 trainable parameters. These parameters have to be adjusted in the learning process to estimate a correct output temperature distribution in an output image.

In summary, as shown in Fig. 4, the images of the geometry (domain) and boundary conditions enter the autoencoder (DNN) with the size of 64×64 pixels (matrix of size 64×64) and they go through the five encoder residual blocks to reach 512 features of size 4×4. Then, they enter the decoder part of the DNN and through five stages, an image with a size of 64×64 is produced. This 64×64 will be the temperature distribution in the given geometry and boundary conditions.

Loss function and evaluation parameters

As mentioned, the present study aimed to show that deep learning models can be used for the rapid estimation of heat transfer phenomena without knowledge of the underlying constitutive equations. A neural network requires adjusted network parameters to estimate a correct and accurate output. The parameters of a neural network can be adjusted during a training process by using an optimizer. The optimizer employs a numeric process, which is known as a training process, to train the network by adjusting its parameters. An optimizer continuously computes the difference between the real output data and the estimated data, the estimation error. Then, it modifies the neural network parameters to reduce the estimated error. The estimation error can be determined using a function, known as a loss function. In summary, an optimizer uses the loss function to determine the estimation error, and then, it employs the estimation error to adjust the neural network parameters in a way to reduce the estimation error of the network. Thus, the selection of an adequate loss function is a crucial step in the training process as it directly affects the behavior of the optimizers and the training process.

The square error is defined as the square of the difference between the real value (target value) of output and the estimated value of that output. The output of the neural network is an image of temperature distribution in the present study. Hence, the pixels are the data that have to be estimated. Therefore, the square error of a pixel is equal to the square of the actual value of a pixel and the estimated value of that pixel. As a result, the mean of square error denotes the average of the square errors of all of the pixels of an image. The MSE can also be introduced for a collection of images, i.e., testing data in the dataset, which in that case, it will be the average of the square errors for all pixels of the collection images.

Most of the neural networks, which are used to generate or estimate the data, employ the MSE as the loss function. The number of pixels in a collection of images is too many, and if the neural network estimates a few numbers of the pixels with too large square errors, e.g., much larger than MSE, they do not affect the overall MSE. These pixels with too large square errors can be considered as outlier pixels, and unfortunately, they cannot be detected by the optimizer during the training process with MSE as an error estimator. The square error of outlier pixels is referred to as the outlier error in this study for convenience. As the optimizer cannot detect such abnormal pixels in the outlet image (temperature distribution), the optimizer will not try to adjust the parameters of the neural network to remove such rare considerable mistakes in the final temperature distribution.

The presence of outlier errors is not crucial in the processing of most of the natural images, but the accurate estimation of all of the pixels in an image, in which the pixels denote a physical meaning or contribute to physical computations, is essential. Thus, introducing a new loss function, capable of dealing with the outlier errors, demanded adequate training of a neural network. Here, it is assumed that the loss function is the maximum value of the square error (MaSE) of all estimated pixels for an image. In the case of a collection of images, the mean of the MaSEs (MMaSE) is the loss function. This way, the optimizer can clearly see the variation of outlier pixels. The optimizer will try to adjust the neural network parameters to reduce the maximum value of the square errors of an image as much as possible. Using MaSE as the loss function, no image pixel can be estimated with an error larger than MaSE. Hence, using MaSE as the error estimation leads to an estimation of a temperature distribution without abnormal temperature values.

The introduced loss functions of MSE, and MaSE can be adopted as an indicator for the evaluation of the neural network outcomes. However, there are also other indicators for the evaluation of the neural network, which will be discussed later. Here, the mathematical details of the computation of MSE and MaSE will be discussed.

Mean Square Error (MSE)

A black and white (grayscale) image can be represented by a 2D matrix, in which the value of each element of the matrix denotes the intensity of a pixel. In contrast, a 2D matrix of data for a physical phenomenon, for instance, a 2D matrix of the temperature of a surface, can be represented as an image. By taking I as a 2D matrix, which represents an image, $I_{\mathrm{r,c}}$ denotes the value of a pixel at the row r and column c. In the case of a collection of images such as the set of test images of the current dataset, a 3D matrix is required to present such a set of images. In the present dataset, T denotes the value of the temperature in the output images. In a collection of images, the value of a pixel can be denoted by three subscripts of n (the image number), r (row), and c (column) as $T_{\mathrm{n,r,c}}$. This topic can be extended to the color images or multi-channel images; in that case, the matrixes with higher dimensional are demanded. In the current research, the output images are just one-channel gray images; hence, the 3D matrixes are adequate. Here, T denotes the actual temperature distribution, evaluated by the numerical method. The estimated temperature distribution, which is the outcome of the neural network, can be represented by P. Here, P is also a 3D matrix with the same characteristics as T. Considering T and P as the actual and the estimated temperature distributions, the Square Error (SE) can be evaluated as:

$$\mathrm{SE} = (T - P) \odot (T - P) \tag{2}$$

where $\odot$ is the element-wise product. As mentioned, SE denotes the square value of the difference between the actual temperature and the estimated temperature, and it is a 3D matrix with the same size as T or P. The error for a pixel of an image can be accessed by $\mathrm{SE}_{\mathrm{n,r,c}}$. Hence, the SE of a pixel can be evaluated as $\mathrm{SE}_{\mathrm{n,r,c}} = (T_{\mathrm{n,r,c}} - P_{\mathrm{n,r,c}})^2$.

MSE is the mean of the SE of a collection of images and can be introduced as:

$$\begin{aligned}\mathrm{MSE} &= \frac{1}{N \times R \times C} \sum_{n=1}^{N} \sum_{r=1}^{R} \sum_{c=1}^{C} \left(T_{\mathrm{n,r,c}} - P_{\mathrm{n,r,c}}\right)^2 \\ &= \frac{1}{N \times R \times C} \sum_{n=1}^{N} \sum_{r=1}^{R} \sum_{c=1}^{C} \mathrm{SE}_{\mathrm{n,r,c}}\end{aligned} \tag{3}$$

where N is the number of images in a collection of images, in which each image is made of R rows and C columns of pixels. For example, in the test images, which are a part of the dataset of the present study, there are 6624 images ($N = 6624$), and each image consists of 64 rows and 64 columns of pixels. Hence, the total number of pixels of the dataset is 27,131,904 pixels. As a result, SE consists of 27,131,904 elements, in which each element is the square of the difference between an actual and estimated value of a pixel. Here, MSE shows the average of all SE elements. Thus, as mentioned, the existence of a few numbers of abnormal errors, outlier errors, cannot be adequately detected in MSE loss function.

Mean of maximum square errors (MMaSE)

The mean of maximum square errors (MMaSE) for a collection of images can be determined with the following steps. The first step is to compute the square error for each of the collection images in the form of a matrix of square errors. Then, the maximum value of the square errors of each image is selected as the error of that image. The summation of the maximum errors of the images divided by the number of the images results in the mean of the maximum errors of the images, which here is the MMaSE. Mathematically, MMaSE can be introduced as follows:

$$\mathrm{MMaSE} = \frac{1}{N} \sum_{n=1}^{N} \max_{1,2} \left(\left(T_{\mathrm{n,r,c}} - P_{\mathrm{n,r,c}}\right)^2 \right) = \frac{1}{N} \sum_{n=1}^{N} \max_{1,2} \left(\mathrm{SE}_{\mathrm{n,r,c}}\right) \tag{4}$$

where $\max_{1,2}$ denotes the implication of the maximum operator on the first and second dimensions of the matrix of square errors of an image. Indeed, $\max_{1,2}$ indicates the maximum of square error of an image.

Considering the test collection images of the present study, which are 6624 images, the computation of the

MMaSE requires only 6624 square error elements to compute the mean error, MMaSE, by performing the mean operator over the set of maximum error of the images of the group of images. However, as mentioned in the previous section, MSE requires 27,131,904 square error elements to compute the mean error (MSE) by employing the mean operator over the entire square errors of images. Moreover, an optimizer, which uses the MMaSE as its loss function, can easily sense the outlier errors since the MMaSE performs the mean operator just over the maximum square errors of a collection of images. The disadvantage of using MMaSE as the loss function is the fact that it will only provide data regarding the maximum errors for the optimizer, and the optimizer is totally unaware of other square errors, which did not participate in computations of MMaSE. Hence, the optimizer cannot control such square errors, and consequently, the temperature of such pixels. As a result, the square errors of an image can be changed unconditionally in the range of zero to the maximum square error of an image.

Maximum Square Errors (MaSE)

The Maximum Square Errors (MaSE) is the maximum of square error in a SE matrix. The SE matrix may be computed for an image or a collection of images. Either way, MaSE is the maximum value of the elements of the SE matrix, i.e., the maximum square error of an image or a set of images. In the case of a collection of images, consisting of only one image, the MaSE will be identical to MMaSE. The MaSE can be computed mathematically as follows:

$$\mathrm{MaSE} = \max\left(\left(T_{\mathrm{n,r,c}} - P_{\mathrm{n,r,c}}\right)^2\right) = \max(\mathrm{SE}) \tag{5}$$

The MaSE is only practical as a characteristic for validation of the estimated results of the neural network. It should be noted that MSE and MMaSE are practical as the loss function and as the validation of the neural network.

Absolute evaluation parameters

The Absolute Error (AE) is useful as it does not change the scale of the errors. This is while the square error was a nonlinear function, which diminishes the normal errors. Hence, square error changes the actual scale of errors. Thus, using an AE, the actual scale of errors remains unchanged and this is an essential advantage for judging the outcomes of a neural network. The absolute error is introduced in the following mathematical form:

$$\mathrm{AE} = |T - P| \tag{6}$$

As seen, AE is very similar to SE but the absolute operator replaces the squared operator. Hence, the concept of AE can be extended to Mean Absolute Errors (MAE), mean Maximum Absolute Errors (MMaAE), and Maximum Absolute Errors (MaAE), accordingly as follows:

$$\mathrm{MAE} = \frac{1}{N \times R \times C} \sum_{n=1}^{N} \sum_{r=1}^{R} \sum_{c=1}^{C} \left|T_{\mathrm{n,r,c}} - P_{\mathrm{n,r,c}}\right| \tag{7}$$

$$\mathrm{MMaAE} = \frac{1}{N} \sum_{n=1}^{N} \max_{0,1}\left(\left|T_{\mathrm{n,r,c}} - P_{\mathrm{n,r,c}}\right|\right) \tag{8}$$

$$\mathrm{MaAE} = \max\left(\left|T_{\mathrm{n,r,c}} - P_{\mathrm{n,r,c}}\right|\right) \tag{9}$$

Numerical method and verification

To produce dataset and other issues, NumPy [45], Matplotlib [46], and scikit-image [47] were used. The deep network of this study was implemented in Keras [48] with Google TensorFlow [49] backend. The codes are written in Python, and the Adam optimizer [50] is adopted for the training of the DNNs. The learning rate is fixed as 0.001, and the parameters of $\beta 1$ and $\beta 2$ are fixed as 0.9 and 0.999, respectively. The β parameters are a part of the training process, and more details can be found in [50]. The training process is commenced after initiating all of the network parameters using the Glorot uniform technique [51]. The DNN is trained using 2000 epochs of the training data. In each epoch, all of the training data are utilized in the format of batches data. In this regard, the training data are randomly categorized in batches, in which each batch contains 32 samples of data. Here, each sample of data consists of one input image with two channels and one single-channel output image. Then, these batches are utilized as training data instead of using individual images. Using batch data accelerates the training process because each time 32 samples are fed into the training process instead of feeding images individually. At the end of each epoch, the validation parameters are computed for both of the validation data and training data and recorded as the training history for later analysis.

It should be noted that the training process is not monotonic, as it depends on the validation data. Hence, loss function for validation data can increase or decrease at each stage of the training process. The training process consists of 2000 epochs, and a DNN with minimal loss function for validation data is adopted as the ultimate trained DNN after completing the training process regardless of its epoch number.

Results and discussion

When it comes to working with the physical images, the magnitude of each pixel indicates a physical value. Hence, the magnitude of each individual pixel is important. In contrast, in a natural image, a group of pixels shows a meaning, and an individual pixel can barely be of any significant sense. Thus, the evaluation of a neural network for the generation of images with physical meaning can be fundamentally different from natural images. Hence, introducing an appropriate loss function (as mentioned in the previous section) as well as the evaluation functions for dealing with the physical images is an essential part of the present study.

Moreover, selecting a structure of the DNN, capable of generating high-quality temperature distribution of output images, is another important aspect of the present research. Hence, the current work aims to address the influence of the loss function on the quality of the generated temperature distributions. To aim this purpose, the MSE and MMaSE are selected as the loss functions for the training of DNN. Therefore, a DNN trained by MSE as loss function is referred as Net_MSE, and similarly, a DNN trained by MMaSE as the loss function is referred as Net_MMaSE. It should be noted that the structure of the DNN for both cases of Net_MSE and Net_MMaSE is identical, and only the adopted loss function, which is employed during the training process, is different.

After the training process, the Net_MSE and Net_MMaSE are investigated in the post-processing step, where the temperature distributions (the output images) for all of the four collections of training data, testing data, validation data, and unseen data are estimated. Using the estimated data, the validation parameters are computed and reported in Table 1.

As mentioned, a small collection of images are produced to be used as the unseen data. The unseen data are utilized to measure the generalization capability of a DNN. The generalization capability of a DNN refers to the ability of the DNN to how well it can employ the learned physical concept to the problems, which has never seen before during the learning process. The purpose of a good DNN model is to generalize the learned concept well and be able to employ it to any data from the problem domain. This allows the neural network to make future estimations on fresh problems. The unseen collection of images in the present study is different from the set of training data. The difference is that the width of the thermal boundaries of the images in the dataset was in the range of 35–58 pixels, while the width of the thermal boundaries in the unseen images is selected equal to 30 pixels. It should be noted that the width and height of the images are linked and changing the width of the images will change the height of the images accordingly. As the DNNs have never seen the unseen data, a DNN with better validation parameters can provide a better generalization capability.

Comparison of MSE and MMaSE

Table 1 shows the characteristic parameters of the trained DNN for both cases of Net_MSE and Net MMaSE. The outcomes are reported for the best trained DNN during the training process. The results are categorized into four categories of training, validation, testing, and unseen data. For each category, the evaluation parameters of five various parameters of MAE, MMaAE, MSE, MMaSE, and MaAE are computed and summarized in Table 1.

As seen, the MSE and MAE parameters of Net_MSE network are slightly smaller than that of Net_MMaSE in terms of training, testing, and validation data. Thus, it can be concluded that Net_MSE shows a better performance in reducing the average of errors compared to the Net_MMaSE. However, Table 1 reveals that the values of MMaSE and MMaAE for Net_MMaSE are better than that of Net_MSE in the term of training, validation, and testing data. Therefore, Net_MMaSE network can more conveniently deal with the outlier errors. These outcomes were expected as the Net_MSE has been trained using the MSE as the loss function, whose goal was reducing the average errors as much as possible. Accordingly, Net_MMaSE was trained using MMaSE as the loss function, whose goal was reducing the maximum errors as much as possible. Thus, the aim of

Table 1 The characteristic parameters of the trained DNNs for two cases of Net_MSE and Net_MMaSE

Data type	DDN type	MAE	MMaAE	MSE	MMaSE	MaAE
Train	Net_MSE	0.00061	0.0219	2.2509E−06	5.8232E−04	0.9715
	Net_MMaSE	0.0014	0.0167	9.6905E−06	3.0494E−04	0.2377
Validation	Net_MSE	0.000796	0.0326	5.4502E−06	0.0022	0.9964
	Net_MMaSE	0.0015	0.0205	1.0806E−05	5.7199E−04	0.3202
Test	Net_MSE	0.000791	0.0322	4.7280E−06	0.0018	0.9953
	Net_MMaSE	0.0015	0.0202	1.0638E−05	5.1169E−04	0.1619
Unseen	Net_MSE	0.0124	0.7642	0.0055	0.6255	1.0
	Net_MMaSE	0.0077	0.7475	0.0034	0.6056	0.9987

optimizer during the training process of Net_MMaSE was reducing outlier errors.

Attention to the magnitude of MAE of both DNNs demonstrates that the difference between these two networks is just 0.000709 in the case of testing data. This is while the difference between MMaAE and MaAE of these two DNNs is 0.012 and 0.8334, respectively. Hence, the difference between MAE of the DNNs, i.e., 0.000709, is minimal compared to the other differences, i.e., 0.12 and 0.8334. Thus, it can be concluded that Net_MMaSE significantly reduced the outlier errors with the cost of a slight growth of MSE and MAE characteristics. As a result, Net_MMaSE notably reduced the values of other characteristic parameters of MaAE, MMaSE, MMaAE, and MaSE.

Here, MaAE indicates the value of the maximum error among all of the estimated pixels. The values of MaAE for the Net_MSE and Net_MMaSE are 0.9953 and 0.1619, respectively, in terms of testing data. These outcomes show that Net_MMaSE estimated all of the pixels of the output images with a maximum absolute error of less than 0.1619 for the testing data. The MaAE error of Net_MSE reveals that at least a pixel of the testing data has been estimated with an absolute error of 0.9953, which is a considerable outlier error. More details about the total number of such outlier errors will be discussed later.

Following the results of Table 1, it can be concluded that the Net_MSE estimated the images with a smaller overall error, but it also made considerable mistakes in estimating some of the pixels in terms of outlier errors. For example, one of the images of the temperature distribution, which is estimated by Net_MSE, is depicted in Fig. 5. The actual temperature distribution is illustrated in Fig. 5a as a reference image, while Fig. 5b shows the estimated temperature distribution of Net_MSE. The absolute error of some of the outlier pixels is written in Fig. 5b to highlight the outlier pixels and their absolute errors. It is vivid that the outlier pixels are completely dark; however, the target image indicates that these pixels shall be yellow. The absolute error

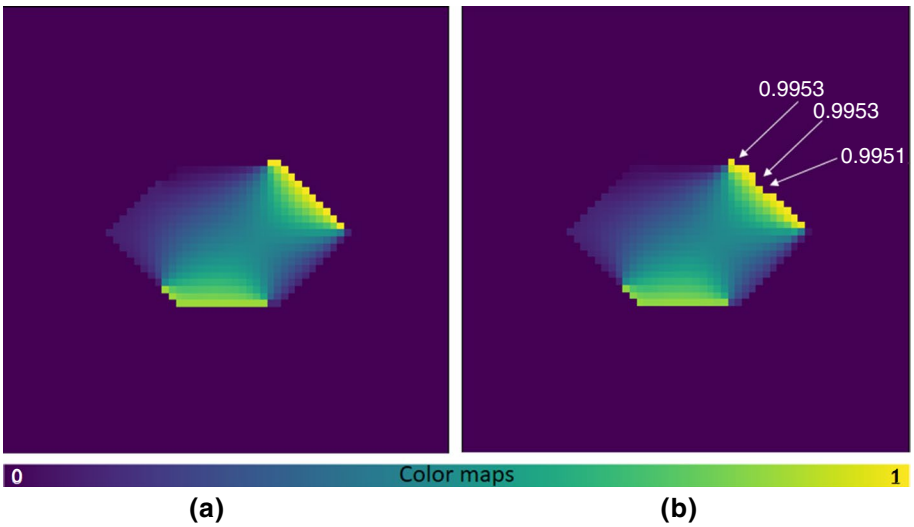

Fig. 5 A sample of estimated image containing outlier pixels; **a** actual image (target), and **b** estimated image with Net_MSE

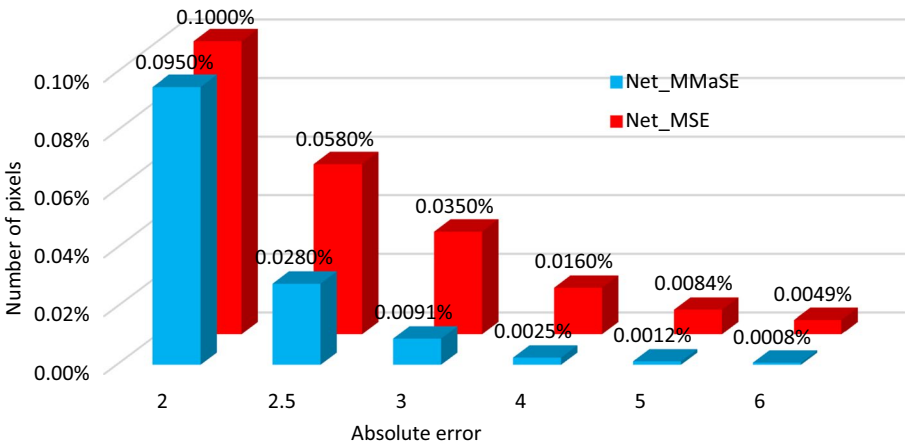

Fig. 6 Cumulative frequency distribution of testing data for some absolute errors

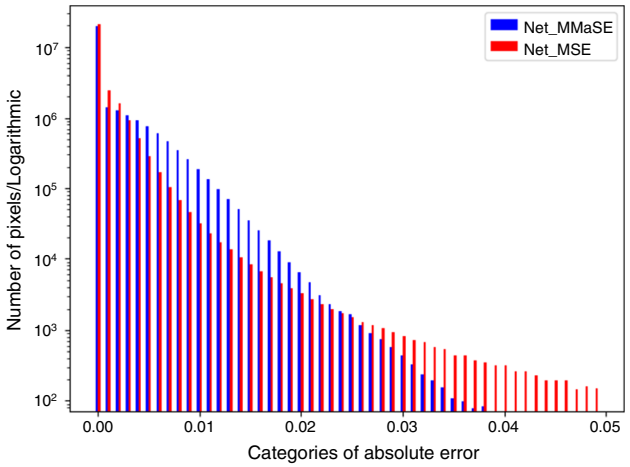

Fig. 7 Frequency distribution curve for testing data from 0 to 0.048. Each bar shows the number of pixels estimated with absolute error in a range of 0.001

of these outlier pixels is about 0.995. As the magnitude of each pixel can be in the normal range of 0–1, the absolute error of 0.995 is a considerable outlier error with no physical estimation value.

One of the important and interesting outcomes of using Net_MMaSE instead of typical Net_MSE for the loss function is obtained for the estimation of the unseen data. As mentioned, the unseen data are input images almost similar to the training data but with smaller geometries. Therefore, the unseen data represent new thermal problems and geometry sizes, which DNN has never seen before. The results of Table 1 reveal that Net_MMaSE outperformed Net_MSE for all of the evaluation characteristics, including characteristics of MSE and MAE. It should be noted that the loss function of Net_MSE was optimized for reducing MSE, and hence, the characteristics evaluation parameters of Net_MSE were better than those of Net_MMaSE for training, validation, and testing data. However, in the case of unseen data, Net_MMaSE outperformed Net_MSE even in terms of MSE and MAE. Hence, a DNN trained by using MMaSE as the loss function provides much better generalizable ability compared to the typical loss function of MSE.

Population and distribution of errors

The overall outcomes of the two DNNs of Net_MSE and Net_MMaSE are compared in Table 1 based on the evaluation parameters. This section aims to provide a more detailed study on the pixels of the testing data, estimated by the trained DNNs. Figure 6 shows a cumulative frequency distribution of the testing data based on the absolute error and the percentage of pixels. This chart shows the percentage of the estimated pixels with an absolute error equal to or bigger than a predefined value. In this chart, each bar corresponds to a value of absolute error, including 0.02, 0.025, 0.03, 0.04, 0.05, and 0.06. The height of each bar is equal to the percentage of the pixels with an absolute error equal to or bigger than the absolute error of the bar. For example, 0.1% of the total pixels of the test data are estimated by Net_MSE with an absolute error equal to or higher than 0.02. Figure 6 is plotted for a few absolute errors for the sake of the comparison of the two DNNs.

Figure 6 shows that 0.058% of the total pixels of the testing data estimated by Net_MSE are within an absolute error of 0.025 or higher, while only 0.028% of the estimated pixels by Net_MMaSE are within the same error. Hence, the number of the estimated pixels by Net_MMaSE is about half of those by Net_MSE for this error range. This figure depicts that the increase in absolute error boosts the difference between the outcomes of two DNNs; as the absolute error increases, the heights of bars for Net_MMaSE drop sharply. This rapid drop indicates that the number of pixels with large errors, the number of outliers, reduced notably in the Net_MMaSE. Thus, the Net_MMaSE has reduced the number of outlier errors notably compared to that of Net_MSE.

Figure 7 shows the frequency distribution curve for both DNNs. This figure displays the number of estimated testing pixels within a bond of absolute error. The absolute errors are studied in the range of 0–0.048 and categorized into 48 categories, and each of the categories is with an absolute error bond of 0.001. The bars are placed at the starting value of each category, and the height of a bar indicates the logarithmic number of each pixel within the error range of that bar. For example, the first bar of Fig. 7 corresponds to the absolute error range of 0.000–0.001 and is placed at zero. Due to the large variation of the number of the pixels, they have been plotted in the logarithmic scale for convenience. For example, the numbers of pixels within an error range of 0.000–0.001 are 20,793,987 and 19322919 for Net_MSE and Net_MMaSE, respectively. Similarly, the numbers of pixels within an error range of 0.001–0.002 are 2,443,122 and 1,418,265 for Net_MSE and Net_MMaSE, respectively.

As shown in Fig. 7, Net_MSE estimated a larger number of pixels compared to that of Net_MMaSE for an absolute error in the range of 0–0.002 and 0.024–0.048. However, Net_MMaSE estimated more pixels in the range of 0.024–0.048 compared to Net_MSE. This behavior indicates that Net_MSE tried to reduce the average error of pixels as much as possible, and hence, the number of pixels estimated by Net_MSE and with an error close to zero is the highest. However, this behavior also failed Net_MSE to estimate a large number of pixels with an adequately low error, and hence, the number of pixels with a significant error of 0.024 or higher is more than that of Net_MMaSE. On the other hand, Net_MMaSE has reduced the number of pixels with very low errors in the range of absolute error of

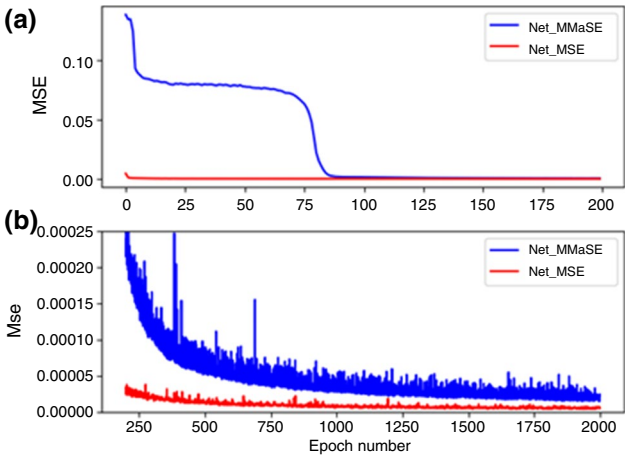

Fig. 8 MSE of training data for each epoch of the training process. **a** First 200 epochs. **b** from epoch 201–2000

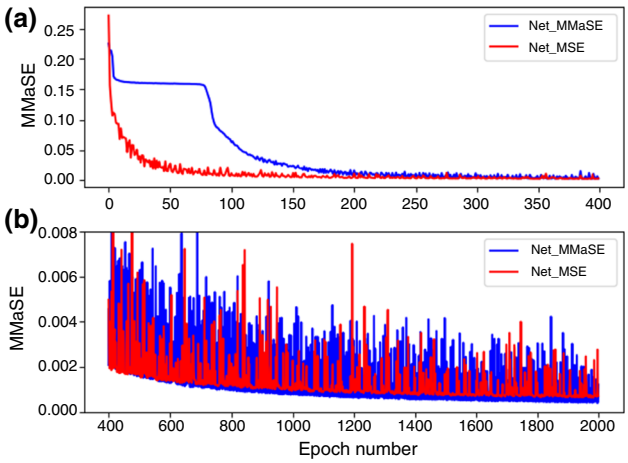

Fig. 9 MMaSE of training data for each epoch of the training process. **a** First 400 epochs, **b** from epoch 401–2000

0.000–0.002, but it has not only succeeded to increase the number of pixels with an adequate error of 0.02–0.024 but also reduced the number of pixels with a significant error of 0.024 and higher compared to that of Net_MSE.

Figure 7 is plotted for the maximum absolute error of 0.048. The total values of MaAE for both of Net_MSE and Net_MMaSE are reported in Table 1. The values of MaAE are 0.9953 and 0.1619 for Net_MSE and Net_MMaSE, respectively. Hence, in the case of plotting the outcomes for the total range of the absolute error, the height of the bars for the case of Net_MMaSE drops to zero for absolute errors larger than 0.1619, while the bars will be continued for Net_MSE until the absolute error of 0.9953. In conclusion, the Net_MMaSE significantly reduced the outlier errors by reducing the number of pixels with an absolute error close to zero.

Speed of training

The training speed of a DNN is one of the important evaluation characteristics of a loss function. A good loss function leads to a DNN with a lower estimation error for a fixed number of epochs. Figures 8 and 9 depict the learning history of Net_MSE and Net_MMaSE DNNs for training data. Figures 8 and 9 are plotted for MSE and MMaSE, respectively. Figures 8a and 9a show the first few hundreds of the epochs for convinces, while the outcomes for the rest of epochs are plotted in Figs. 8b and 9b. Figures 8a and 9a demonstrate that Net_MSE is much faster than Net_MMaSE at pioneer epochs when the learning process commences, and it reduced both MSE and MMaSE parameters notably.

Moreover, the fluctuations of Net_MMaSE are much more significant than that of Net_MSE, which is because of the nature of the MMaSE loss function. MMaSE solely monitors the maximum square error of an image, and hence, there are only a few maximum errors in a collection of images, which are used for computing MMaSE. Accordingly, changing each of these few pixels could change the overall MMaSE considerably. The behavior of MMaSE is absolutely opposite to the nature of MSE, which incorporates all of the pixels of a collocation of images in the computation of the loss function, and consequently, changing only a few outlier pixels does not affect MSE notably.

The MSE is small in most of the training epochs for the Net_MSE during the training process, and similarly, the MMaSE is smaller than MSE for the Net_MMaSE for most of the epochs. These observations are in agreement with the selected loss functions of these DNNs.

Study of estimated images

Here, some examples of the estimated images using the Net_MSE and Net_MMaSE are illustrated and compared. Figure 10 depicts four images from the testing data. The images in the first and second row, respectively, show the images with the largest values of the MAE and MaAE, which are estimated by Net_MSE. The images of the third and fourth row, respectively, correspond to the largest MAE and MaAE estimated by Net_MMaSE. The columns from left to right show the target image (the actual CFD solution), the estimated image by Net_MSE, the estimated image by Net_MMaSE, the absolute error of the estimated image by Net_MSE, and the absolute error of the estimated image by Net_MMaSE, respectively. The color map of each image is added below the image for convenience. The range of the color maps for the fourth and fifth columns is identical for the sake of better comparison.

Moreover, the corresponding values of MAE and MMaAE are reported above each image. The absolute error for the fourth and fifth columns is computed as the absolute

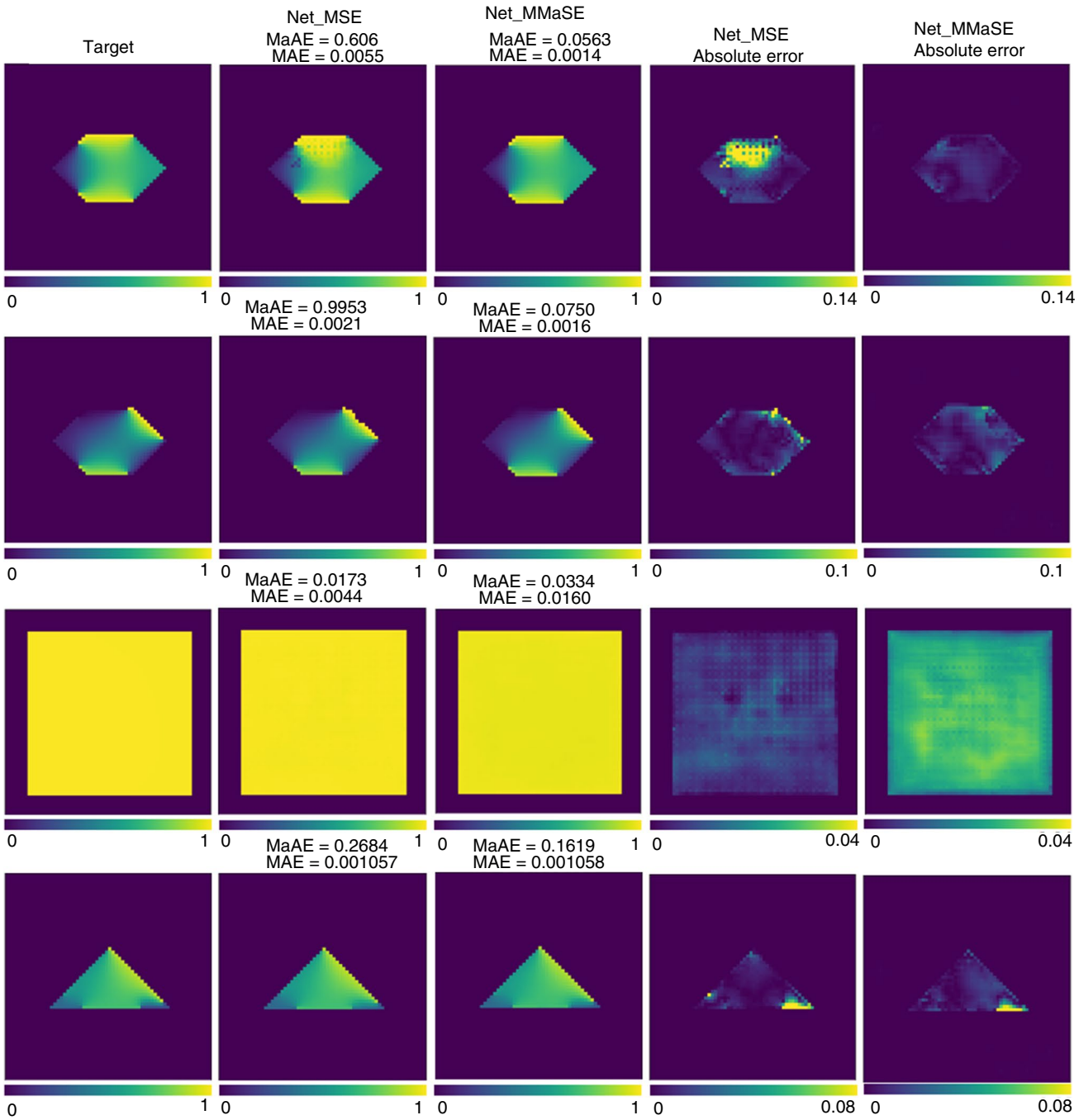

Fig. 10 Some samples of estimated test images by Net_MSE and Net_MMaSE. Row1: The image with the worst MAE among the estimated images by Net_MSE. Row2: The image with the worst MaAE among the estimated images by Net_MSE. Row3: The image with the worst MAE among the estimated images by Net_MMaSE. Row4: The image with the worst MaAE among estimated images by Net_MMaSE

difference between the estimated image and the target image. Hence, the absolute error in the fourth column is the absolute difference between the first column and second column, and similarly, the absolute error in the fifth column is the absolute difference between the first column and the third column.

Figure 10 shows that the images in the fifth column are darker than the fourth column for all rows except the third row. A darker color denotes a lower error, and hence, as seen, the absolute error of the estimated images by Net_MMaSE is better than the images estimated by Net_MSE, except the third row. The yellow pixels in the fourth and fifth

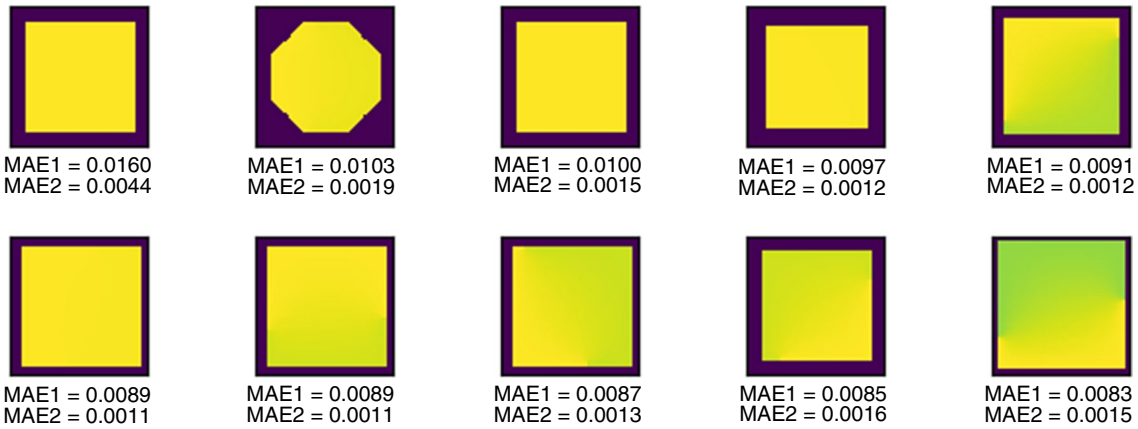

Fig. 11 Some target images of the test data with the worst MAE when they were estimated by Net_MMaSE. MAE1 and MAE2 denote the MAE of an image when estimated by Net_MMaSE and Net_MAE, respectively

columns show the significant absolute errors compared to the other pixels. The number of yellow pixels in the fourth column is much more than those in the fifth column, except for the third row. Indeed, the yellow pixels in the fourth and fifth columns denote the outlier pixels. Thus, from the results of Fig. 10, it can be concluded that both DNNs have been successful in estimating the overall temperature distribution, but the Net_MMaSE could provide much uniform and low error distribution by reducing the magnitude of the outlier errors.

Attention to the images of the third row reveals that the target image is almost a mono-color image. Checking the input channel for the boundary conditions shows that there are only two input temperatures at the boundaries, which are very close to the values of 0.9937 and 0.9918. Hence, the target image, including the boundary conditions and the temperature distribution, appears as a semi-mono-color image with no apparent distinction between the boundaries and temperature distribution inside the image. The reason for such inadequate estimation of MAE and MaAE of the results in the third row is followed by investigating ten testing images estimated by Net_MMaSE in Fig. 11.

Figure 11 shows the estimated testing images by Net_MMaSE with the most MAE errors. Below each image, the MAE-value-estimated Net_MMaSE is written as MAE1, and similarly, the MAE corresponding to Net_MSE is written as MAE2. As seen, in all of the images, MAE1 is larger than MAE2. It is evident that the temperature variations at the boundaries of these images are minimal. Thus, it can be concluded that an image with low-temperature variation at the boundaries weakens the estimation capability of a Net_MMaSE.

Conclusions

In the present study, the deep neural networks were utilized to learn the physics of conduction heat transfer in 2D geometries. A big dataset of various geometries and temperature boundary conditions were constructed. The actual temperature distribution in each geometry, along with the boundary conditions, was obtained by using the conventional finite volume method in a 64×64 structured grid. Then, the geometry and the corresponding temperature boundary conditions, along with actual computed temperature distribution, were used as the collection of big data to teach the DNN. An autoencoder DNN with a block residual structure was tailored for the present study to learn the physic of conduction heat transfer. Two different loss functions were introduced to be used during the learning process. One of the loss functions, MSE, was based on the typical average square error estimation conventional in the context of DNNs for natural image generation. The other loss function MMaSE was introduced based on the maximum square error related to the physical needs of the DNNs. Finally, using the structure of the DNN and the introduced loss functions, the estimation capability of DNNs for learning and estimation of the temperature distributions in 2D geometries was addressed. The primary outcomes of the present study can be summarized as follows:

1. The DNN, which was trained by MSE as the loss function (Net_MSE), estimated the images with a lower average error compared to a DNN, which was trained by MMaSE as the loss function (Net_MMaSE). However, a portion of the estimated by Net_MSE was pixels with high absolute errors, outlier errors. The Net_MMaSE notably reduced the number and magnitude of the outlier

pixels by the cost of a slight increase in the average error of the pixels.

2. The estimated images by Net_MMaSE contain fewer and smaller outlier errors compared to Net_MSE, and hence, they are much similar to the target imaged compared to that of Net_MSE.
3. In the early stages of the training, the convergence rate of Net_MMaSE was much lower than that of Net_MSE, but the convergence rate of Net_MMaSE is raised by the continuation of the training process. Eventually, the ultimate value of MMaSE for Net_MMaSE was lower than that of Net_MSE. In the entire training process, the reduction rate of MSE for Net_MSE was better than that of Net_MMaSE.
4. Both of the DNNs were successful in learning the physics of heat transfer and estimating the overall temperature distribution. However, the results, estimated by the customized loss function of MMaSE, were much better than that of typical MSE. Moreover, the generalization capability of Net_MMaSE was much better than Net_MSE.
5. The results of the present research demonstrate the general capability of DNNs in learning the physics of heat transfer. The trained DNNs were capable of generating images with accurate temperature distribution without knowing the underlying governing partial differential equation. Hence, DNNs are promising for learning more advanced physics, such as convective heat transfer and hydrodynamic of fluids.

The outcomes demonstrated that a tailored loss function, MMaSE, could lead to a DNN with much better estimations. Hence, introducing new hybrid loss functions and improving the internal structure of DNNs for a better estimation can be subject to future studies.

References

1. Sheikholeslami M, Jafaryar M, Shafee A, Babazadeh H. Acceleration of discharge process of clean energy storage unit with insertion of porous foam considering nanoparticle enhanced paraffin. J Clean Prod. 2020;261:121206.
2. Sheikholeslami M, Arabkoohsar A, Babazadeh H. Modeling of nanomaterial treatment through a porous space including magnetic forces. J Therm Anal Calorim. 2019;140:1–10.
3. Shafee A, Sheikholeslami M, Jafaryar M, Selimefendigil F, Bhatti M, Babazadeh H. Numerical modeling of turbulent behavior of nanomaterial exergy loss and flow through a circular channel. J Therm Anal Calorim 2020; pp. 1–9.
4. Sheikholeslami M, Arabkoohsar A, Jafaryar M. Impact of a helical-twisting device on the thermal–hydraulic performance of a nanofluid flow through a tube. J Therm Anal Calorim. 2020;139(5):3317–29.
5. Shafee A, Bhatti M, Muhammad T, Kumar R, Nam ND, Babazadeh H. Simulation of convective MHD flow with inclusion of hybrid powders. Tc. 2020;10:2.
6. Waqas H, Khan SU, Bhatti M, Imran M. Significance of bioconvection in chemical reactive flow of magnetized Carreau-Yasuda nanofluid with thermal radiation and second-order slip. J Therm Anal Calorim. 2020;140:1–14.
7. Håstad J, Goldmann M. On the power of small-depth threshold circuits. Comput Complex. 1991;1(2):113–29.
8. Farimani AB, Gomes J, Pande VS. Deep learning the physics of transport phenomena. 2017. arXiv preprint arXiv:170902432.
9. Hinton GE, Osindero S, Teh Y-W. A fast learning algorithm for deep belief nets. Neural Comput. 2006;18(7):1527–54.
10. Bengio Y, Lamblin P, Popovici D, Larochelle H, editors. Greedy layer-wise training of deep networks. In: Proc. advances in neural information processing systems. vol. 19. 2006. p. 153–60.
11. Erhan D, Manzagol P-A, Bengio Y, Bengio S, Vincent P, editors. The difficulty of training deep architectures and the effect of unsupervised pre-training. In: Artificial intelligence and statistics. Clearwater Beach, Florida, USA, 2009. p. 153–60.
12. Bengio Y, Delalleau O. Justifying and generalizing contrastive divergence. Neural Comput. 2009;21(6):1601–21.
13. Hinton GE, Salakhutdinov RR, editors. Using deep belief nets to learn covariance kernels for Gaussian processes. In: Advances in neural information processing systems. vol. 20. 2008. p. 1249–56.
14. Hinton GE, Salakhutdinov RR. Reducing the dimensionality of data with neural networks. Science. 2006;313(5786):504–7.
15. Russakovsky O, Deng J, Su H, Krause J, Satheesh S, Ma S, et al. Imagenet large scale visual recognition challenge. Int J Comput Vis. 2015;115(3):211–52.
16. Graves A, Mohamed A-R, Hinton G, editors. Speech recognition with deep recurrent neural networks. In: 2013 IEEE international conference on acoustics, speech and signal processing. IEEE; 2013. p. 6645–9.
17. Sun Y, Liang D, Wang X, Tang X. Deepid3: Face recognition with very deep neural networks. 2015. arXiv preprint arXiv:150200873.
18. Zhining L, Xiaozhuo G, Quan Z, Taizhong X, editors. Combining statistics-based and cnn-based information for sentence classification. In: 2016 IEEE 28th international conference on tools with artificial intelligence (ICTAI). IEEE; 2016. p. 1012–8.
19. Wang T-C, Zhu J-Y, Hiroaki E, Chandraker M, Efros AA, Ramamoorthi R, editors. A 4d light-field dataset and cnn architectures for material recognition. In: European conference on computer vision. Springer; 2016. p. 121–38.
20. Wu L, Shen C, Hengel Avd. Personnet: Person re-identification with deep convolutional neural networks. 2016. arXiv preprint arXiv:160107255.
21. Zhang H, Xu T, Li H, Zhang S, Wang X, Huang X et al., editors. Stackgan: text to photo-realistic image synthesis with stacked generative adversarial networks. In: Proceedings of the IEEE international conference on computer vision. 2017. p. 5907–15.
22. Pascual S, Bonafonte A, Serrà J. SEGAN: Speech enhancement generative adversarial network. 2017. arXiv preprint arXiv:170309452.
23. Choi Y, Choi M, Kim M, Ha J-W, Kim S, Choo J, editors. Stargan: unified generative adversarial networks for multi-domain image-to-image translation. In: Proceedings of the IEEE conference on computer vision and pattern recognition. 2018. p. 8789–97.
24. Matsugu M, Mori K, Mitari Y, Kaneda Y. Subject independent facial expression recognition with robust face detection using a convolutional neural network. Neural Netw. 2003;16(5–6):555–9.

25. Yu N, Jiao P, Zheng Y, editors. Handwritten digits recognition base on improved LeNet5. In: The 27th Chinese control and decision conference (2015 CCDC). IEEE; 2015.
26. Ioffe S, Szegedy C. Batch normalization: Accelerating deep network training by reducing internal covariate shift. 2015. arXiv preprint arXiv:150203167.
27. Agarap AF. Deep learning using rectified linear units (relu). 2018. arXiv preprint arXiv:180308375.
28. He K, Zhang X, Ren S, Sun J, editors. Delving deep into rectifiers: surpassing human-level performance on imagenet classification. In: Proceedings of the IEEE international conference on computer vision. 2015. p. 1026–34.
29. Han K, Mun YY, Gweon G, Lee J-G, editors. Understanding the difficulty factors for learning materials: a qualitative study. In: International conference on artificial intelligence in education. Springer; 2013. p. 615–8.
30. Krizhevsky A, Sutskever I, Hinton GE, editors. Imagenet classification with deep convolutional neural networks. In: Advances in neural information processing systems. 2012. p. 1097–105.
31. Szegedy C, Liu W, Jia Y, Sermanet P, Reed S, Anguelov D et al., editors. Going deeper with convolutions. In: Proceedings of the IEEE conference on computer vision and pattern recognition. 2015. p. 1–9.
32. Goodfellow I. NIPS 2016 tutorial: Generative adversarial networks. 2016. arXiv preprint arXiv:170100160.
33. He K, Zhang X, Ren S, Sun J, editors. Deep residual learning for image recognition. In: Proceedings of the IEEE conference on computer vision and pattern recognition. 2016. p. 770–8.
34. Sharma R, Farimani AB, Gomes J, Eastman P, Pande V. Weakly-Supervised Deep Learning of Heat Transport via Physics Informed Loss. 2018. arXiv preprint arXiv:180711374.
35. Ronneberger O, Fischer P, Brox T, editors. U-net: Convolutional networks for biomedical image segmentation. In: International conference on medical image computing and computer-assisted intervention. Springer; 2015. p. 234–41.
36. Bergman TL, Incropera FP, Lavine AS, DeWitt DP. Introduction to heat transfer. Hoboken: Wiley; 2011.
37. Mirza M, Osindero S. Conditional generative adversarial networks. 2014;9:24. arXiv:170902023.
38. Raissi M, Yazdani A, Karniadakis GE. Hidden fluid mechanics: learning velocity and pressure fields from flow visualizations. Science. 2020;367(6481):1026–30.
39. Bhatti MM, Shahid A, Abbas T, Alamri SZ, Ellahi R. Study of activation energy on the movement of gyrotactic microorganism in a magnetized nanofluids past a porous plate. Processes. 2020;8(3):328.
40. Jaluria Y. Computational heat transfer. New York: Routledge; 2017.
41. Incropera FP, Lavine AS, Bergman TL, DeWitt DP. Principles of heat and mass transfer. Amsterdam: Wiley; 2013.
42. Baldi P, editor. Autoencoders, unsupervised learning, and deep architectures. In: Proceedings of ICML workshop on unsupervised and transfer learning. 2012.
43. Masci J, Meier U, Cireşan D, Schmidhuber J, editors. Stacked convolutional auto-encoders for hierarchical feature extraction. In: International Conference on Artificial Neural Networks. Springer; 2011. p. 52–9.
44. Monti RP, Tootoonian S, Cao R, editors. Avoiding degradation in deep feed-forward networks by phasing out skip-connections. In: International conference on artificial neural networks. Springer; 2018. p. 447–56.
45. Oliphant TE. A guide to NumPy. New York: Trelgol Publishing USA; 2006.
46. Hunter JD. Matplotlib: a 2D graphics environment. Comput Sci Eng. 2007;9(3):90.
47. Van der Walt S, Schönberger J, Nunez-Iglesias J, Boulogne F, Warner J, Yager N, et al. scikit-image: image processing in Python. PeerJ. 2014;2:e453.
48. Chollet F, others. Keras. 2015. https://keras.io.
49. Abadi M, Agarwal A, Barham P, Brevdo E, Chen Z, Citro C, et al. TensorFlow: Large-scale machine learning on heterogeneous systems. 2015.
50. Kingma DP, Ba J. Adam: A method for stochastic optimization. 2014. arXiv preprint arXiv:14126980.
51. Glorot X, Bengio Y. Understanding the difficulty of training deep feedforward neural networks. In: Proceedings of the thirteenth international conference on artificial intelligence and statistics. 2010. p. 249–56.