Neural Computing and Applications
https://doi.org/10.1007/s00521-022-07233-1

ORIGINAL ARTICLE

New loss functions to improve deep learning estimation of heat transfer

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

Received: 24 March 2021 / Accepted: 29 March 2022

Abstract
Deep neural networks (DNNs) are promising alternatives to simulate physical problems. These networks are capable of eliminating the requirement of numerical iterations. The DNNs could learn the governing physics of engineering problems through a learning process. The structure of deep networks and parameters of the training process are two basic factors that influence the simulation accuracy of DNNs. The loss function is the main part of the training process that determines the goal of training. During the training process, lost function regularly is used to adapt parameters of the deep network. The subject of using DNNs to learn the physical images is a novel topic and demands novel loss functions to capture the physical meanings. Thus, for the first time, the present study aims to develop new loss functions to enhance the training process of DNNs. Here, three novel loss functions were introduced and examined to estimate the temperature distributions in thermal conduction problems. The images of temperature distribution obtained in the present research were systematically compared with the literature data. The results showed that one of the introduced loss functions could significantly outperformance the literature loss functions available in the literature. Using a new loss function improved the mean error by 67.1%. Moreover, using new loss functions eliminated the pixels predictions (with large errors) by 96%.

Keywords Deep convolutional neural networks · Loss function · Heat transfer images · Physical images

1 Introduction

The two-dimensional steady-state heat condition in heat transfer is a simple and fundamental problem. The conduction heat transfer is defined by $\nabla^2 T = 0$, which is the Laplace equation, and T defines the temperature distribution. The Laplace equation is the foundation of many physics' phenomena. Even though it is a simple partial differential, the analytical solution is not existing in most cases. For heat transfer phenomena, the analytical solutions depend on the boundary conditions and heat transfer domain, which in most cases do not exist. While the analytical solutions are very scarce to solve heat transfer equations, there are man
y well-known numerical solutions such as meshless, finite volume, and finite element methods. The overall process of this method consists of partitioning domain solutions to subdomains, creating algebraic equations for subdomains, and finally solving all equations simultaneously with analytical or numerical methods. These methods have two

✉ Mohammad Ghalambaz
m.ghalambaz@gmail.com

Mohammad Edalatifar
m.edalatifar@gmail.com

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

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: 02 May 2022

main drawbacks. First, they are computationally costly methods, especially when the number of subdomains is high. Second, the results of a solved problem cannot be used to simplify the solution of that problem with new boundary conditions and domains.

In recent years, researchers have shown that deep neural networks (DNN) have tremendous power to cluster [1], classify and generate videos, voices, and images [2–5]. The deep neural networks have also been used for the identification of abnormal EEG signals [6], diabetic retinopathy detection [7], and brain tumor detection [8]. A DNN consists of layers, connections between layers, structure, and trainable parameters. In order to estimate data accurately, the trainable parameters must be adjusted. An iterating numerical technique, training process, adjust the parameters by using a dataset collection of samples. Many approaches are suggested to increase the accuracy of deep neural networks. Design new structure of DNNs, using random normal distribution [9, 10], batch normalization technique [11], and rectified linear unit (ReLU) [12] active function are some ideas suggested to improve the accuracy of DNNs.

Nowadays, DNNs have different applications. In 2020, Raissi et al. [13] introduced the DNNs as an artificial intelligence tool with the capability that they can learn physical phenome such as flow over a cylinder. Berg et al. [14] approximated solutions of partial differential equations with a deep feedforward neural network. They suggested a pre-train step for training the network using the available boundary data and increasing hidden layers. The pre-train step reduced the time and required iterations of the training network. Lin et al. [15] enhanced a heat transfer topology optimization problem by employing a combined deep convolutional network. The topology optimization aimed to find an optimized distribution of thermally conductive materials in electronic components. They reduced some of the optimization steps by using the neural network. With this technique, they could accelerate and enhance the optimization process dramatically. Liu et al. [16] constituted a novel hybrid deep neural network to estimate wind speed. Their deep networks compose a stacked denoising auto-encoder and a long short-term memory network. These authors trained the network with real-time big data from the wind farm running log.

Recently, some researchers have investigated the ability of DNNs to simulate heat transfer. Sharma et al. [17] applied a U-Net deep network [18] to estimate heat distribution images. The U-Net is a type of DNN that considers input images locally and globally and estimates output images. They used a middle-size dataset in order to adjust the parameters of their network in a process called training. Their dataset consisted of many square-shaped geometries with various heated boundaries. The DNN was utilized to learn the temperature distribution. In contrast to conventional datasets, their dataset did not have output images. Hence, there were no target images to train the DNNs. These authors utilized a spatial convolution filter based on the finite difference (FD) [19] method to compute the error of estimation by convolving predicted images and the defined filter. By using this technique, datasets were generated at the same time as network training, and therefore, without having a big dataset, they benefited from the advantages of a big dataset. Nevertheless, their dataset only contained square geometries. Moreover, the dataset was not completely predefined, and hence, it was not unique to be used for future investigations.

Farimani et al. [20] used a deep network to estimate heat transfer images. They produced a dataset containing 6230 samples to adjust the parameters of their DNN. Each sample of the dataset consisted of an image of a heated geometry as the input and its temperature distribution as the output image. The boundaries of geometries had different sizes and positions on the domain bonds. Various geometries, including rectangle, disk, annulus, and triangle, were also investigated. The images of the heat distribution, which was applied as the output data, were generated by using the finite difference (FD) method. The main drawbacks of the mentioned dataset were poor diversity of geometries and using few images. The dataset was made of only 6230 images and four shapes.

Sharma et al. [17] and Barati Farimani et al. [20] attempted to estimate temperature distribution in heated geometries with low errors by selecting an appropriate DNN structure. The structure of a DNN and the training process play a significant role in the accuracy of the DNN. So, a well-designed DNN structure and an adequate training process are essential for achieving an accurate DNN. One of the crucial components of a training process is the configuration of the optimizer. An optimizer is a mathematical approach that tunes the parameters of DNN during the training process. It uses a function, loss function, to calculate the error of estimated data, and then, it changes the value of DNN's parameters based on the calculated error.

Most recently, Edalatifar et al. [21] proposed a comprehensive dataset consisting of 44,160 samples of temperature distribution data. Each sample was made of an input and an output image of 64 × 64 pixels. The input image contained two channels, a channel for geometrical specifications and a channel for the thermal boundary conditions. The output images had only one channel that showed the temperature distribution, which was computed by the finite difference method (FDM) [19]. The geometries included square, regular hexagonal, triangular, and regular octagonal figures; the width and height of shapes were changed between 35 and 58 pixels. The investigation

of Edalatifar et al. [22] showed that the design of a loss function significantly affects the training behavior and accuracy of a DNN to estimate the heat images. First, they used the typical mean square error (MES) as the loss function for the training of a DNN. The outcomes showed that very few of the pixels in the images were estimated with huge errors compared to the rest of the pixels. They named these pixels the outlier pixels and their estimated error the outlier error. Since each pixel indicates a physical meaning, the predicted value of every single pixel could be important in future engineering decisions and designs. Thus, the accurate estimation of all pixels is essential. Edalatifar et al. stated that the number of outlier pixels is scarce, and hence, their value could not change the total value of MSE. As a result, the optimizer could not see these outlier pixels and adjust the network parameters accordingly. To eliminate the outlier pixels, these authors introduced the mean of maximum square errors (MMaSE) loss function. They employed MMaSE instead of MSE to train the DNN, and they found that the new loss function could efficiently eliminate outlier pixels and their error. However, the main drawback of using MMaSE was a slight increase in the MSE of the estimated error. Indeed, they concluded that the estimation error of very well-predicted pixels was raised slightly, so DNN could focus on outlier pixels and remove them. Moreover, during the training process, particularly at the initial training epochs, the convergences of DNN trained by MMaSE were slow.

As seen, regardless of the structure of DNN, the design of a loss function is a crucial task for the training of DNN and learning the physical images. In [22], it was found that the outlier pixels could be eliminated by introducing MMaSE, but MSE was raised. Although MMaSE was beneficial to remove the un-acceptable pixels and validate the general response of DNNs, it declined the overall accuracy of the network by the growth of MSE. The present research aims to introduce loss functions to not only eliminate the outlier pixels, but also increase the total accuracy of the network.

The structure of this paper is as follows. In Sect. 2, the new loss functions for physical images will be introduced. Then, in Sect. 3, a convolutional deep neural structure will be introduced. Then, the details of the training process will be discussed in Sect. 4. In Sect. 5, some evaluation indexes will be introduced to evaluate the robustness of the introduced loss functions and the proposed novel neural network structure. Section 6 concerns the database and verification approach. The evaluation indexes will be used in Sect. 7 to evaluate the obtained results and judge the robustness of the introduced loss functions and network structure. Finally, the results will be concluded in Sect. 7.

2 Loss functions

Pixels of some images, for instance, heat images, interpret a physical meaning for a physical quantity. These images represent a physical quantity that could be considered as physical images. In natural images, which show a real-world image such as an object or a landscape, a single pixel barely could provide a meaning, while a group of pixels could determine an object. In contrast to natural images, a pixel of a physical image is valuable because its value represents a physical quantity that could later influence an engineering decision. As mentioned, when an image is estimated, some of its pixels, outlier pixels, could be estimated with an error much larger than other pixels. The outlier pixels seldom could be visible in natural images, so they are not important in the estimation process. However, as mentioned, the outlier pixels in a physical image can influence later decisions and processes. Most generative networks are trained with MSE loss function; however, Edalatifar et al. in [21] represented a new loss function, MMaSE. They trained a deep network with MMaSE to estimate heat distribution images, and the result revealed that MMaSE could reduce outliers dramatically compared to MSE.

Suppose $T_{\mathrm{n,c,r}}$ and $P_{\mathrm{n,c,r}}$ are the 3D matrix of target and predicted images, respectively. Here, n determines images in T and P matrix, and r and c are row and column indexes. As a result, $T_{\mathrm{n,c,r}}$ determines the pixel's value on the (c,r) coordinate of the image n. Using this definition, the square errors (SE) are declared as follows:

$$SE_{n,c,r} = \left(P_{n,c,r} - T_{n,c,r}\right)^2 \tag{1}$$

Here, SE is a matrix with size and dimensions equal to P and T. The mean of the SE's elements is denoted by MSE. Hence, MSE is the mean of square errors of all predicted pixels and calculated as follows [23, 24]:

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

In Eq. (2), N is the total number of the images in T, while R and C, respectively, determine the number of rows and columns of each image (64 for our dataset). Therefore, MSE can be calculated following the computation of SE. Here, SE is created for all pixels in the $T_{n,c,r}$ and $P_{n,c,r}$ with Eq. (1). Then, MSE is calculated as the mean of SE. It must be pointed out that MSE is the most common loss function to train generative DNNs.

To reduce the training time, most deep networks are trained with a group of data (images) instead of the single data in a step of the training process. This group of data is known as a batch. For a batch, MSE is the mean of square

errors of the batch's pixels. There is one major problem with MSE in a batch, which is the overlooking of a few large errors. Indeed, there is a huge number of pixels in a batch. Hence, if few pixels of the batch are estimated with square error very much than MSE, outlier pixels, their variation cannot change MSE value considerably. As a result, the optimizer of the deep network, which adapts parameters of DNN based on the value of the loss function, cannot reduce outlier errors.

A new loss function, MMaSE, is introduced in [21] to overcome such a drawback. This loss function is the mean of the maximum square error of each image (maximum outlier error) in a group of predicted images. MMaSE calculates as:

$$MMaSE = \sum_{n=1}^{N} \frac{1}{N} \max_{0,1} \left(\left(P_{n,c,r} - T_{n,c,r} \right)^2 \right) = mean \left(\max_{0,1} (SE) \right) \quad (3)$$

in which, $\max_{0,1}$ determine maximum elements on the first and second dimensions of a 3D matrix. Thus, $\max_{0,1}(\mathrm{SE})$ denotes the maximum outlier error of each image in SE. MMaSE could be computed using simple steps as 1—Compute SE matrix for all pixels in $T_{n,c,r}$ and $P_{n,c,r}$ using Eqs. (1), 2—Extract the maximum of square errors of each image within SE, and 3—Compute the mean of extracted square errors in step 2 as MMaSE.

It should be noted that MMaSE, which was introduced in [21], is an appropriate parameter to investigate outlier errors in all predicted pixels. Its value depends only on the biggest outlier errors of each predicted image. Therefore, the variation of all pixels in an image instead of the maximum of them does not influence the value of MMaSE. The variation of the outliers is clearly visible by using Eq. (3) as the loss function. Thus, the optimizer could easily adapt the trainable parameters of DNN to reduce the outlier errors, i.e., in [15], a DNN was trained with MMaSE loss function and was shown that the outlier errors were reduced dramatically.

As mentioned, MMaSE was defined as the mean of the biggest square error of each estimated image, and therefore, only one pixel with the biggest estimation error will be used in the computation of MMaSE. This led to a major drawback that most pixels with large errors did not contribute to the computation of MMaSE. Hence, many pixels with moderate and large errors did not have any role in MMaSE's value. Consequently, the optimizer could not follow the variation of all pixels' errors, and they could freely adopt any error value. Hence, as shown in [15], both the mean of errors (MSE and MAE) and variation of errors during the training process were increased. To overcome this drawback, three new loss functions are introduced in this paper. The first one is the mean of multiple maximum square errors (MMuMaSE). It could be computed by using the following four steps:

1. Compute SE for all pixels in $T_{\mathrm{n,c,r}}$ and $P_{\mathrm{n,c,r}}$ using Eq. (1).
2. Sort square errors in SE for each image separately.
3. Extract M maximum square errors of each image.
4. Calculate the mean of selected errors in step 3 as MMuMaSE.

where M is the number of the maximum square errors of each image, which was extracted to calculate MMuMaSE. A comparison between MMaSE and MMuMaSE reveals that if $M = 1$, then MMuMaSE is equal to MMaSE. When the loss function is MSE, all pixels of images participate in the training process. If the loss function is MMaSE, only one pixel of each image will participate in the computation of the loss function and takes part in the training process. However, MMuMaSE extracts M square errors of each image to calculate the loss value. Thus, MMuMaSE as the loss function reveals more pixels with large errors to the optimizer during the optimization process compared to MMaSE. At the same time, MMuMaSE just feeds the impact of the important pixels to the optimizer. As a result, a significant reduction in outlier errors could be expected.

The second loss function, introduced here, could be considered the sum of MSE and MMuMaSE, which can be referred to as MSE + MMuMaSE. This loss function has two main properties. First, the estimation error of all pixels participates in the training process. Second, in contrast to MSE, the variance of outlier errors could be seen. Therefore, by using MSE + MMuMaSE, a notable decrease in mean and outlier errors could be expected compared to any previous loss functions, i.e., MSE, MMaSE, and MMuMaSE.

Finally, the sum of MSE and MMaSE is the third loss function, referred to as MSE + MMaSE. This loss function has properties similar to MMaSE, but it also has its unique features. Indeed, MMaSE focuses on outlier errors more than MMuMaSE, and thus, MSE + MMaSE can be expected to reach lower errors than MSE + MMuMaSE and a higher mean error. In the next section, a novel DNN structure will be proposed. The structure will benefit from a feature with information transfer between its layers.

3 Deep neural network structure

In order to test the impact of various loss functions on the performance of DNNs, a convolutional neural network (CNN) with an auto-encoder [24] structure was adopted as the test case. The CNN will be trained with MSE, MMaSE, MMuMaSE, MSE + MMuMaSE, and MSE + MMaSE

loss functions, and the outcomes will be analyzed. An auto-encoder is a typical type of neural network, which its structure is suitable for a variety of purposes such as noise reduction, dimensionality reduction, and generation data. The auto-encoder has two main parts, an encoder and a decoder. The encoder extracts features from input data, and the decoder re-constructs output data using extracted features.

The layers of a neural network could be either dense or convolutional. A neural network with convolutional layers has less trainable parameters than the same network with dense layers. Neural networks with convolutional layers are known as Convolutional Neural networks (CNN). If CNN contains an auto-encoder structure, it could be called a convolutional auto-encoder [25]. Here, a convolutional auto-encoder will be used as the test CNN. The structure of this CNN is shown in Fig. 1. As seen, this CNN consists of two parts, encoder and decoder. The encoder is made of ten convolution layers where a rectified linear unit (ReLU) [12] as the active function and a batch normalization [11] were added before each layer. The decoder structure is similar to the encoder, but its layers are deconvolutional layers, and they act as an inverse function of convolutional layers. After the last layer of the decoder, a sigmoid activation function was added. This network contains 1,769,729 trainable parameters that must be adapted in the training process.

In summary, the encoder gets input images with a size of $64 \times 64 \times 2$, where two in notation indicates the number of the input channels of an image. The encoder extracts 512 features of 4×4, and then the decoder gets the features and generates output heat distribution in the form of a $64 \times 64 \times 1$ image. The next section represents the details of the training process for the proposed DNN.

4 Training process

Here, the back-propagation method was used to evaluate the loss functions. However, training neural networks with the gradient base approaches will suffer from several limitations in terms of accuracy, processing time, overfitting, quickly falling into local minima, etc. Some recent researchers proposed nature-inspired algorithms such as [26–30] to train neural networks robustly and efficiently. Here, our codes are ready with typical back-propagation, and for the sake of convenience, we used the following approach. It could be a good idea if future researchers use nature-inspired algorithms and discuss the impact of the training approach on the performance and accuracy of DNNs in learning physical images.

A DNN can be mathematically modeled as a function of free parameters (weights of DNN) that maps input data (signal) to output. Hence,

$$Y = f(W, B, X) \tag{4}$$

That W and B are the weights of DNN so-called trainable parameters, which must be adapted during an iteration procedure called the training process. Here, B is known as bias, while X and Y are input and output of DNN, respectively. DNNs are constructed from many layers; each layer is a mathematical function that can be defined as:

$$z^l = f\left(W^l, B^l, a^{l-1}\right) \tag{5}$$

where z^{l} and $a^{\mathrm{l}-1}$ are output and input data of layer l, respectively, and W^{l} and B^{l} are their weights. In Eq. 4, W and B are a collection of all W^{i} and B^{i} for $1 \leq i \leq L$, where L is the number of layers. Calculating the output of DNN is a single-phase procedure consisting of calculating the output of each layer from the first layer to the last one. However, before using DNN, their weights must be tuned to estimate data accurately. Gradient descent is one of the most used machine-learning algorithms to train a DNN. Gradient descent (GD) represents the first time by Cauchy in 1847 [31]. A loss (cost) function is needed for using it, which is a function to calculate the distance between an estimated value and actual value. The cost function can be modeled as:

$$C = f\left(W, B, Y_T, Y_p\right) \tag{6}$$

where Y_{T} is the actual (target) value, and Y_{P} is the estimated value with DNN. The training goal is the adaption of weights (W, B) to minimize as possible as C. The smaller C, the more similar Y_{T} and Y_{P}. Hence, the problem of the training process is defined as:

$$\underset{W,B}{\text{minimize}}\ C = f\left(W, B, Y_T, Y_p\right) \tag{7}$$

Figure 2 shows the relationship between the cost function gradient with respect to a weight and cost function. According to GD method (as shown in Fig. 2), the weights must be changed in the opposite way of the gradient to decrease the loss function value. GD method is a well-accepted approach for training neural networks where its convergence was addressed by Patrick Cheridito et al. [32]. Unfortunately, GD method does not determine the size of the movement of weights, and it only shows the direction of the movement. Therefore, optimization could be achieved only in an iteration process (training process). In each step of the training process, the weights are changed based on the below equations:

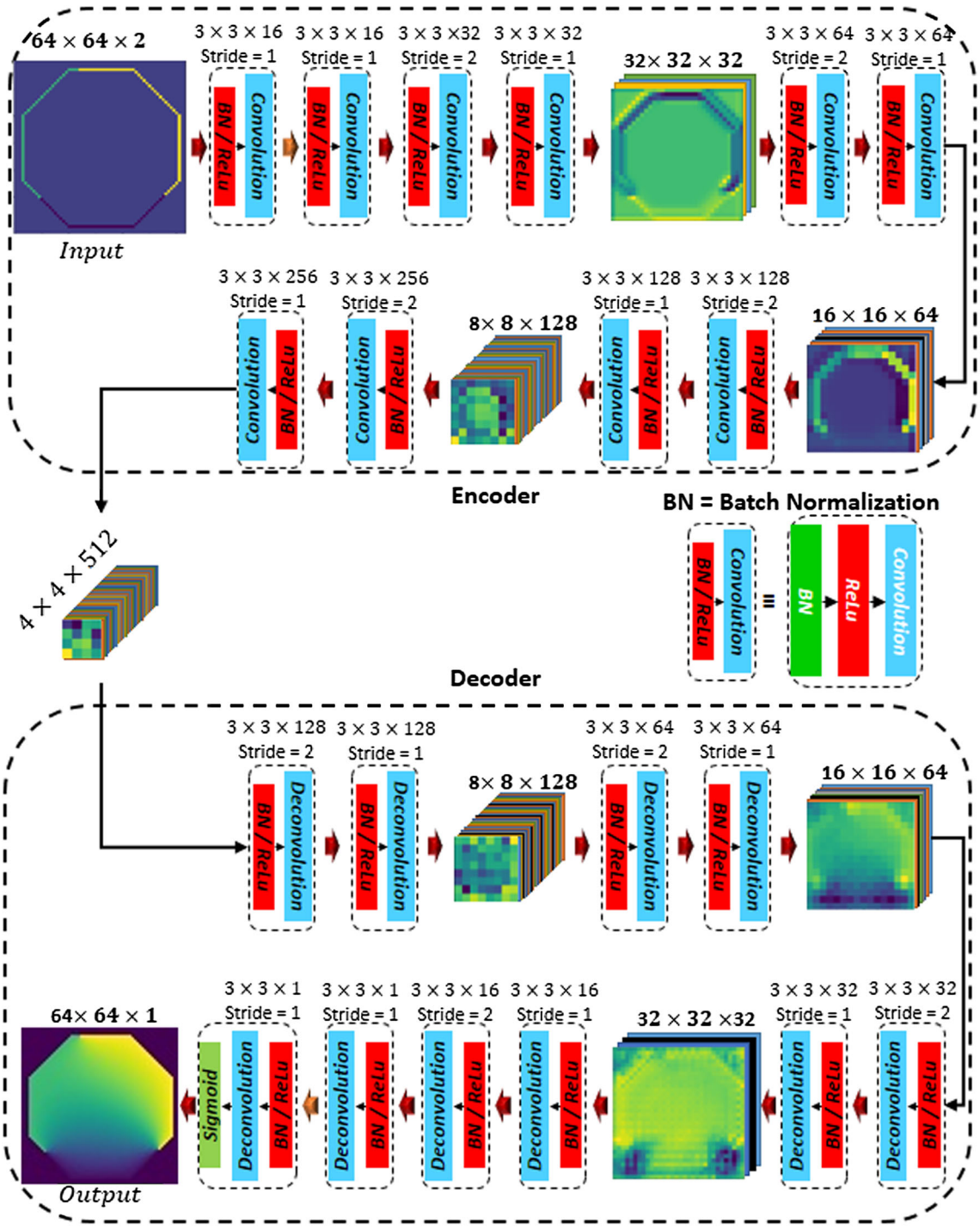


Fig. 1 Structure and characteristics of the deep network used in this paper

$$w_{i,new} = w_{i,old} - \eta \frac{\partial C}{\partial w_i} \tag{8}$$

$$b_{i,new} = b_{i,old} - \eta \frac{\partial C}{\partial b_i} \tag{9}$$

In Eqs. 8 and 9, η is the learning rate, which is a hyperparameter to control the length of movement. Here, w_i and b_i are ith element of W and B. Then, $\partial C/\partial w_i$ is the gradient of the loss function with respect to w_i. Similarly, $\partial C/\partial b_i$ is the gradient of the loss function with respect to b_i. The main part of Eqs. 8 and 9 is the gradient that must be calculated correctly. Back-propagation is a well-known algorithm to calculate gradients in a training process [33, 34].

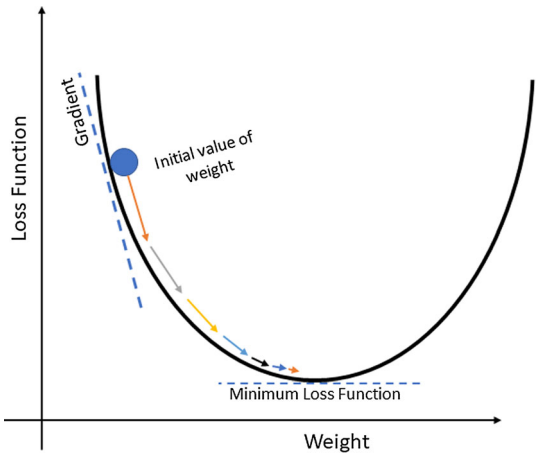


Fig. 2 A scheme of the relation between GD and cost function

The training process has two main phases. The first phase is the forward propagation which consists of passing input data (signal) throughout the DNN and calculating its output (Y_P), and then calculating the loss function. The second phase is the back-propagation. During the back-propagation phase, the estimation error passes from the last to the first layer, and the gradients can be calculated. When the gradients are determined, the weights are changed according to Eqs. 8 and 9. The relationship between loss function and gradients will be addressed in the next section.

4.1 Relationship between loss function and gradients

The full-connection (dense) layer and convolutional layer are more popular layers that make the structure of DNNs. Each layer has its properties and mathematical model. Therefore, the forward propagation and back-propagation are related to the layer's type. For simplicity, suppose a DNN with L dense layers, which is shown in Fig. 3.

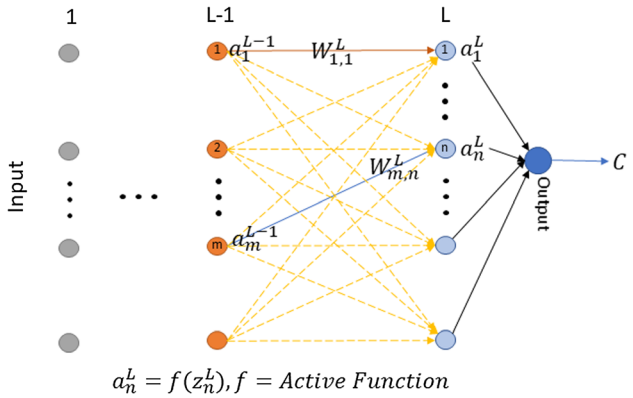


Fig. 3 A DNN consists of L dense layers

Here, $z^L{}_n$ is the output of nth neuron of Lth layer. $a^L{}_n$ is the output of the active function of Lth so, $a^L{}_n = f(z^L{}_n)$ where f is the active function. For simplicity, suppose the active function is identity. So $z^L{}_n = a^L{}_n$ and $\partial a^L{}_n/\partial z^L{}_n = 1$. According to Fig. 3, the gradient of loss function C with respect to $w^L{}_{11}$ is defined as follows:

$$\frac{\partial C}{\partial w_{11}^L} = \frac{\partial C}{\partial a_1^L} \times \underbrace{\frac{\partial a_1^L}{\partial z_1^L}}_{=1} \times \frac{\partial z_1^L}{\partial w_{11}^L} \tag{10}$$

where $w^L{}_{11}$ is the weight between the first neuron of layer L-1 and the first neuron of layer L. According to properties of dense layers, $z^L{}_1$ is calculated as:

$$z_1^L = \sum_i \left(a_i^{L-1} \times w_{i,1}^L\right) + b_1^L \tag{11}$$

Therefore,

$$\frac{\partial z_1^L}{\partial w_{11}^L} = a_1^{L-1} \tag{12}$$

$$\frac{\partial C}{\partial w_{11}^L} = \frac{\partial C}{\partial a_1^L} \times \underbrace{\frac{\partial a_1^L}{\partial z_1^L}}_{=1} \times a_1^{L-1} \tag{13}$$

By recalling Eq. 8:

$$w_{1,1,new}^L = w_{1,1,old}^L - \eta \frac{\partial C}{\partial w_{1,1}^L} = w_{1,1,old}^L - \eta \frac{\partial C}{\partial a_1^L} \times \frac{\partial a_1^L}{\partial z_1^L} \times a_1^{L-1} \tag{14}$$

In general, the gradient of loss function C with respect to a weight between neuron m of layer L-1 and neuron n of layer L is equal to:

$$\frac{\partial C}{\partial w_{m,n}^L} = \frac{\partial C}{\partial a_n^L} \times \frac{\partial a_n^L}{\partial z_n^L} \times a_m^{L-1} \tag{15}$$

By recalling Eq. 7:

$$w_{m,n,new}^L = w_{m,n,old}^L - \eta \frac{\partial C}{\partial w_{m,n}^L} = w_{m,n,old}^L - \eta \frac{\partial C}{\partial a_n^L} \times \frac{\partial a_n^L}{\partial z_n^L} \times a_m^{L-1} \tag{16}$$

4.2 Influence of new loss functions

In Eqs. 13–16, there is a key part, $\partial C/\partial a^L{}_n$, that plays the main role in changing the weights. It is the gradient of the loss function with respect to the output of the last layer. Suppose the last layer has M neurons (M outputs). As MSE is the most popular lost function to optimize generator DNN, suppose again loss function C is MSE:

$$C = \frac{1}{M}\sum_{i=1}^{M}\left(a_i^L - Y_{i,T}\right)^2 \tag{17}$$

$a^{\mathrm{L}}{}_{\mathrm{i}}$ is the output of ith neuron of the last layer, and then it is the output of DNN on neuron i and can be denoted as $Y_{\mathrm{i,P}}$. Hence,

$$a_i^L = Y_{i,P}$$
$$C = \frac{1}{M}\sum_{i=1}^{M}\left(Y_{i,P} - Y_{i,T}\right)^2 \tag{18}$$

Consequently,

$$\frac{\partial C}{\partial a_i^L} = \frac{2}{M}\sum_{i=1}^{M}\left(a_i^L - Y_{i,T}\right) = \frac{2}{M}\sum_{i=1}^{M}\left(Y_{i,P} - Y_{i,T}\right) \tag{19}$$

Recalling Eq. 16:

$$w^L_{m,n,new} = w^L_{m,n,old} - \eta \times \frac{\partial a^L_n}{\partial z^L_n} \times a_m^{L-1} \times \frac{2}{M}\sum_{i=1}^{M}\left(Y_{i,P} - Y_{i,T}\right) \tag{20}$$

$Y_{\mathrm{i,P}} - Y_{\mathrm{i,T}}$ is the error of estimation of *i*th neuron and is shown with e_{i}. Then,

$$ME = \frac{2}{M}\sum_{i=1}^{M}\left(Y_{i,P} - Y_{i,T}\right) = \frac{2}{M}\sum_{i=1}^{M}\left(e_i\right) \tag{21}$$

According to Eqs. 20 and 21, if the loss function is MSE, the new value of weights is dependent on the mean of estimation errors (ME). For an image, as discussed before, the number of outlier occurrences is small (less than 1% of all samples), while M is a large value. Thus, these outlier errors cannot change ME notably. Consequently, when the error values are small, the gradient of such errors diminishes, and as a result, the weights cannot change in a direction to further reduce the outlier errors.

Now, let us suppose MMaSE is the loss function. In such a case, for an image according to MMaSE definition, M = 1, and hence Eqs. 20 and 21 are changed to the following equations:

$$w^L_{m,n,new} = w^L_{m,n,old} - \eta \times \frac{\partial a^L_n}{\partial z^L_n} \times a_m^{L-1} \times 2\left(Y_{k,P} - Y_{k,T}\right) \tag{22}$$

$$MaE = 2\left(Y_{k,P} - Y_{k,T}\right) = 2e_k \tag{23}$$

in which, k index is an output index with a maximum error (outlier error). From these equations, it is clear that the gradient solely relies on the output with the maximum error (MaE) regardless of the error of other outputs. As a result, the change of weights will be in a direction to reduce the maximum error while leaving the other smaller errors alone. This is the reason for an increase in the mean of estimated errors when the loss function is MMaSE. From another point of view, there is a significant difference between the maximum error (outlier error) and mean of errors, and thus the gradient of errors is high. When the gradient of errors is large, the loss function and mean of errors could be reduced slowly. The same conclusion could be valid of MMuMaSE. Now let us suppose the loss function is MSE + MMaSE. In such a case, Eqs. 20 and 21 can be written as follows:

$$w^L_{m,n,new} = w^L_{m,n,old} - \eta \times \frac{\partial a^L_n}{\partial z^L_n} \times a_m^{L-1} \times 2\left[\frac{1}{M}\sum_{i=1}^{M}\left(Y_{i,P} - Y_{i,T}\right) + \left(Y_{k,P} - Y_{k,T}\right)\right] \tag{24}$$

$$ME\&MaE = 2\left[\frac{1}{M}\sum_{i=1}^{M}\left(Y_{i,P} - Y_{i,T}\right) + \left(Y_{k,P} - Y_{k,T}\right)\right] = 2\left[\frac{1}{M}\sum_{i=1}^{M}\left(e_i\right) + e_k\right] \tag{25}$$

It is clear that using MSE + MMaSE, the gradients are a function of both the mean of errors and the outlier errors. In this case, since the error is the summation of the mean error and maximum error, when the mean of errors becomes small, the overall error, e.g., ME&MaE, could significantly change with the change of outlier errors. Consequently, the weights could be changed in a direction to reduce the outlier errors. In a case when the mean error is large, it can change the gradients and reduce the mean errors as well. The same conclusion is true for MSE + MMuMaSE. Now, some evaluation parameters are required to evaluate the benefit of the proposed loss function, which is the subject of the next section.

5 Evaluation parameters

This section aims to study the performance of the trained neural networks. Here, the trained DNNS learned the concept of the temperature distribution due to conduction heat transfer from a dataset of temperature distribution. The conduction heat transfer temperature distributions were produced by solving many cases of conduction heat transfer problems using the well-known finite difference method, which is a numerical solution approach. The dataset contains 44,160 samples of temperature distribution data, in which each sample image was made of an input and an output image of 64 × 64 pixels. The details of this dataset and solution approach are discussed in [21].

Thus, some parameters are needed to investigate mean and outlier errors. In previous sections, MMaSE and MSE were represented as a function to calculate loss estimation. They could also be used as evaluation parameters as well.

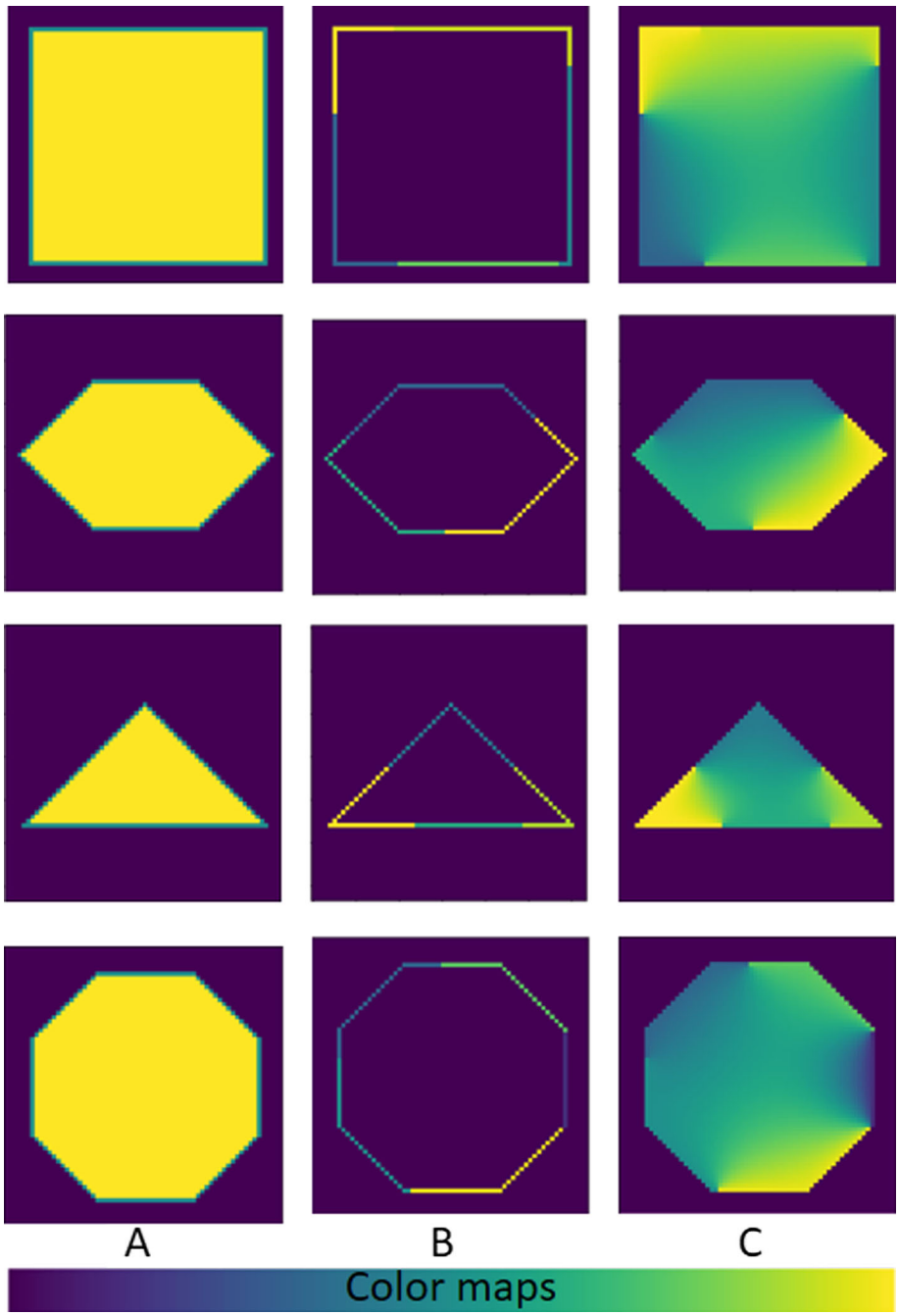


Fig. 4 Some samples of dataset. **A**: input, channel 1. **B**: input, channel 2. **C**: output

Indeed, MSE indicates the mean square of all estimated pixels errors; therefore, it is suitable as an index to explore the mean error. Moreover, MMaSE is the mean of the maximum square error of each estimated image, and thus, it is proper to investigate outlier errors. However, the square error is a nonlinear function and changes the scale of errors. Hence, the absolute error that is a linear function for positive and negative numbers could be a better indicator that was used here. Absolute Error (AE) is defined as follows:

$$AE_{n,c,r} = \left|T_{n,c,r} - P_{n,c,r}\right| \tag{26}$$

Here, $AE_{n,c,r}$ is the absolute error of a pixel on the (c,r) coordinate of the image n. A comparison between Eqs. (1) and (26) shows that the absolute and square operators are the only difference between them. Mean absolute error (MAE) [24] and mean of maximum absolute error (MMaSE) are defined as:

Table 1 Evaluation parameters for five networks

	Evaluation parameter	Net_MSE	Net_MMa	Net_MMuMa	Net_MSE & MMa	Net_MSE & MMuMa
Training data	*MAE*	0.0011	0.0052	0.0021	0.0025	8.964e-4
	MMaAE	0.0611	0.0409	0.0237	0.0172	0.0284
	MSE	1.010e-5	5.975e-5	8.8471e-6	1.261e-5	2.88e-6
	MMaSE	0.0044	0.0019	6.3366e-4	3.115e-4	0.0011
Validation data	*MAE*	0.0013	0.0052	0.0023	0.0026	0.0011
	MMaAE	0.0762	0.0512	0.0377	0.0239	0.0372
	MSE	1.455e-5	6.270e-5	1.12649e-5	1.508e-5	4.931e-6
	MMaSE	0.00748	0.0033	0.0022	0.0008	0.0022
Testing data	*MAE*	0.0013	0.0052	0.0023	0.0026	0.0011
	MMaAE	0.0760	0.0509	0.0374	0.0233	0.0366
	MSE	1.438e-5	6.218e-5	1.1032e-5	1.475e-5	4.732e-6
	MMaSE	0.0074	0.0032	0.0021	7.093e-4	0.0020

$$MAE = \frac{1}{N \times C \times R} \sum_{n=1}^{N} \sum_{c=1}^{C} \sum_{r=1}^{R} \left| P_{n,c,r} - T_{n,c,r} \right| = mean(AE) \tag{27}$$

$$MMaAE = \frac{1}{N} \sum_{n=1}^{N} \max_{0,1} \left(\left| T_{n,c,r} - P_{n,c,r} \right| \right) \tag{28}$$

MAE, similar to MSE, is suitable for checking the mean of estimation error, and MMaAE, similar to MMaSE, is useful for comparing outlier errors. The utilized dataset and verification method will be discussed in the next section.

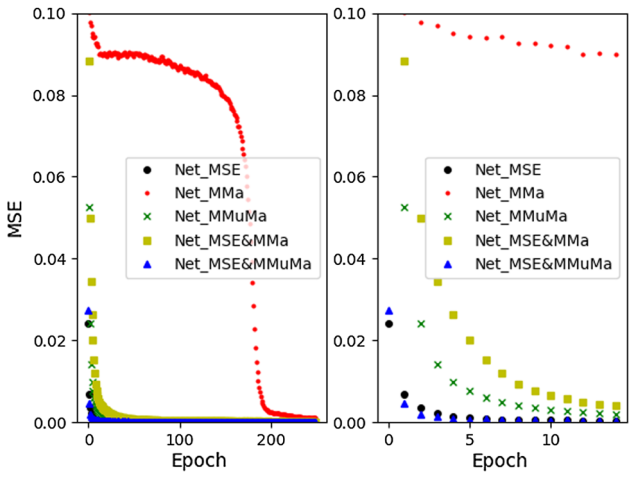


Fig. 5 Variation of MSE of training data within the training process. Left: Epoch 0 to 250. Right: Epoch 0 to 15

6 Numerical method and verification

In this paper, we use a dataset introduced in [21] to train our DNN. It consists of 44,160 samples of heat transfer. Each sample has an input and an output image of 64 × 64. The input images have two channels; the first channels have a triangle, square, regular hexagon, or regular octagon shape. They are binary images that introduce the geometry of the input shape; therefore, the output of shapes is filled with zero, and the shapes are plotted with one. The second channels consist of the boundary of the shapes in the first channel that randomly slices between two and five pieces, and each piece has a random value between 0 and 1. The inside and outside of the boundaries are filled with 0 and 1, respectively. These boundaries are called heat boundaries. The output images have only one channel that shows the heat distribution image. Each output image is similar to its second input channel; its inside of heat boundaries is filled with heat distribution calculated with the finite volume method (FVM). Some samples of this dataset are shown in Fig. 4, where each row indicates one sample. Columns A and B respectively show channel one and two input images, and column C shows output images. All heat boundaries have random width between 35 to 58 pixels, and the center of boundaries are matched to the center of the domain. This dataset has three parts include training, testing, and validation. The training part is 70 percent of all samples that are selected randomly. Each one of the testing and validation parts is 15 percent of all samples, selected randomly, too. The value of input and output images pixels is between zero and one, so we do not have any normalization on the dataset. The dataset is accessible here: https://doi.org/10.17632/rw9yk3c559.2 [21, 22].

DNN of this paper was implemented with TensorFlow [35] and Python. This DNN was trained with Adam

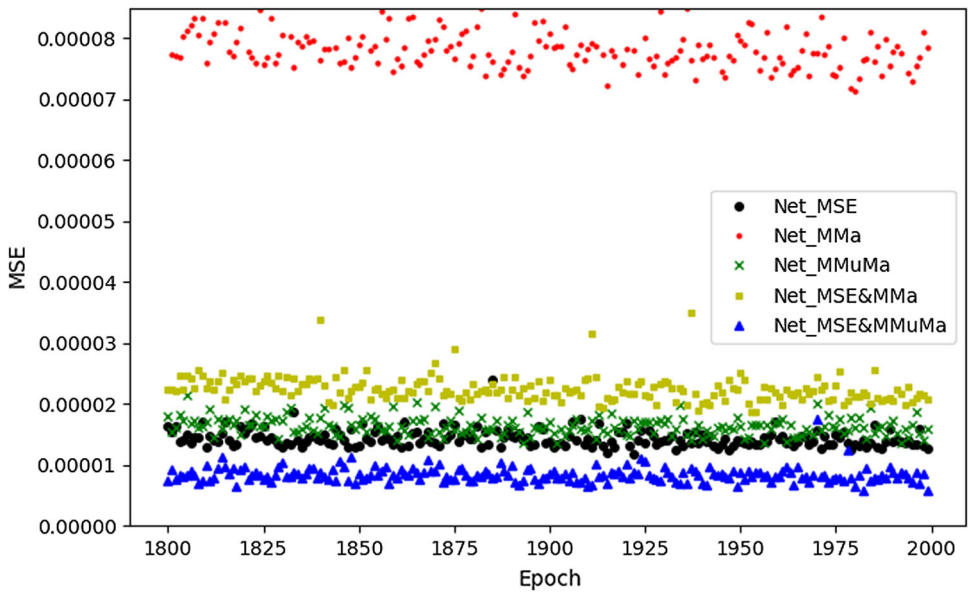


Fig. 6 MSE of train data for the last 200 epochs of the training process

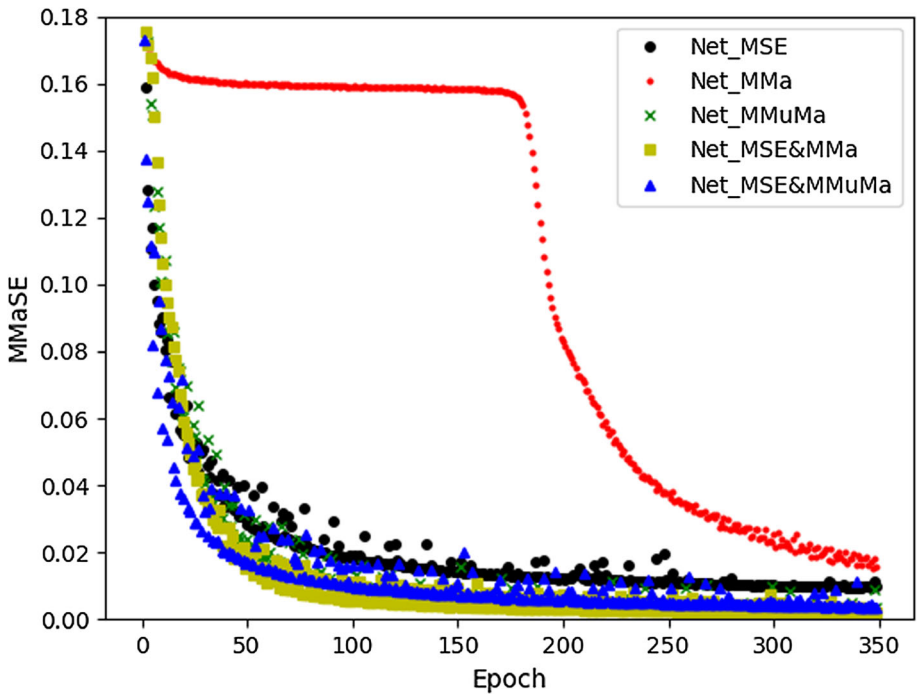


Fig. 7 MMaSE of train data for the first 350 epochs of the training process

optimizer [36] with a learning rate of 0.001. β1 and β2 are two main parameters of Adam that in this paper they were set to 0.9 and 0.999, respectively. The trainable parameters of our DNN were initialized with the Glorot Uniform technique [37]. Each training process for each network was done for 2000 epochs. Instead of using a single sample, batches of 32 samples were used to train DNN. Validation parameters of validation and training data were calculated for each epoch and saved as a history of the training process. The history of the process is useful to explore the speed of reducing error parameters during the training process and comparing loss functions. Due to loss error having irregular variance during the training process, the best training point is when the loss error of validation data has less value, and the DNN parameters were saved at this point.

In this paper, DNN was independently trained five times with MSE, MMaSE, MMuMaSE, MSE + MMaSE, and MSE + MMuMaSE that for simplification refer to them as Net_MSE, Net_MMa, Net_MMuMa, Net_MSE&MMa,

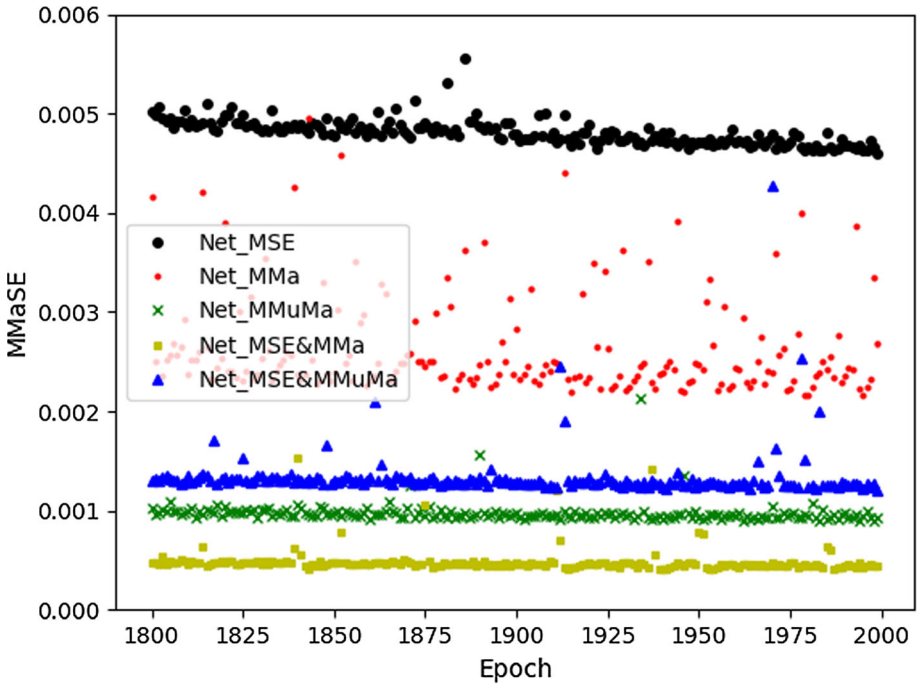


Fig. 8 MMaSE of train data for the last 200 epochs of the training process

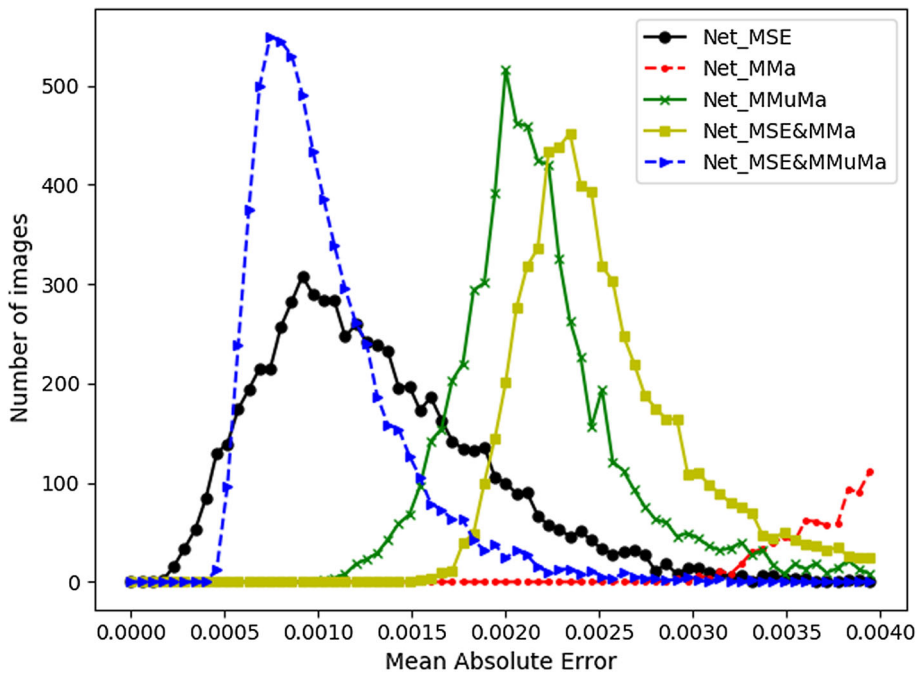


Fig. 9 Frequency distribution chart for predicted test images base on MAE. Each point represents the number of images predicted by MAE in the range 5.7e-05

and Net_MSE&MMuMa. In this study, M = 82 was used when the lost function was MMuMaSE or MSE + MMuMaSE that it is equal to 4 percent of all predicted pixels. The benefit of DNNs trained with proposed loss functions will be evaluated in the results section.

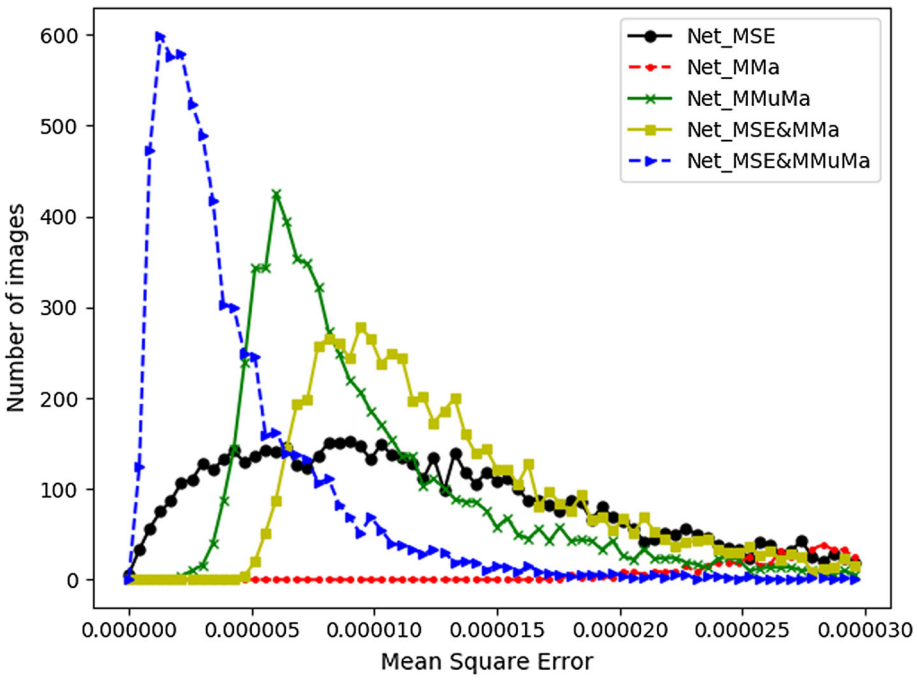


Fig. 10 Frequency distribution curve for predicted testing images base on MSE. Each point represents the number of images predicted by MSE in the range 4.29e-07

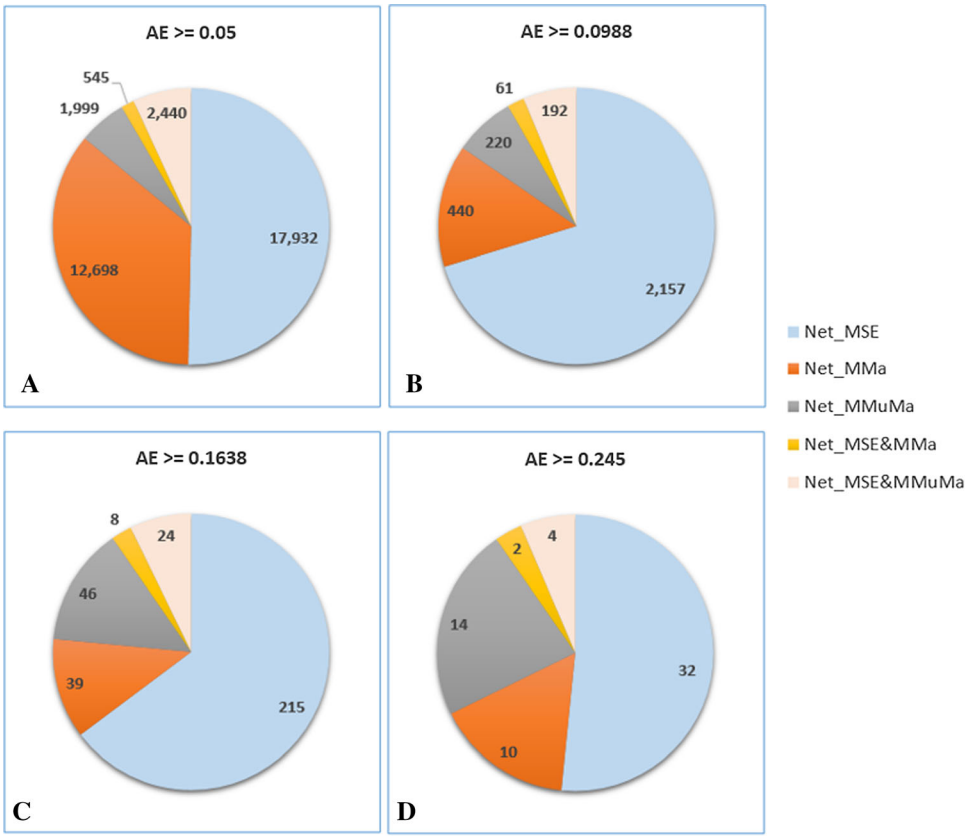
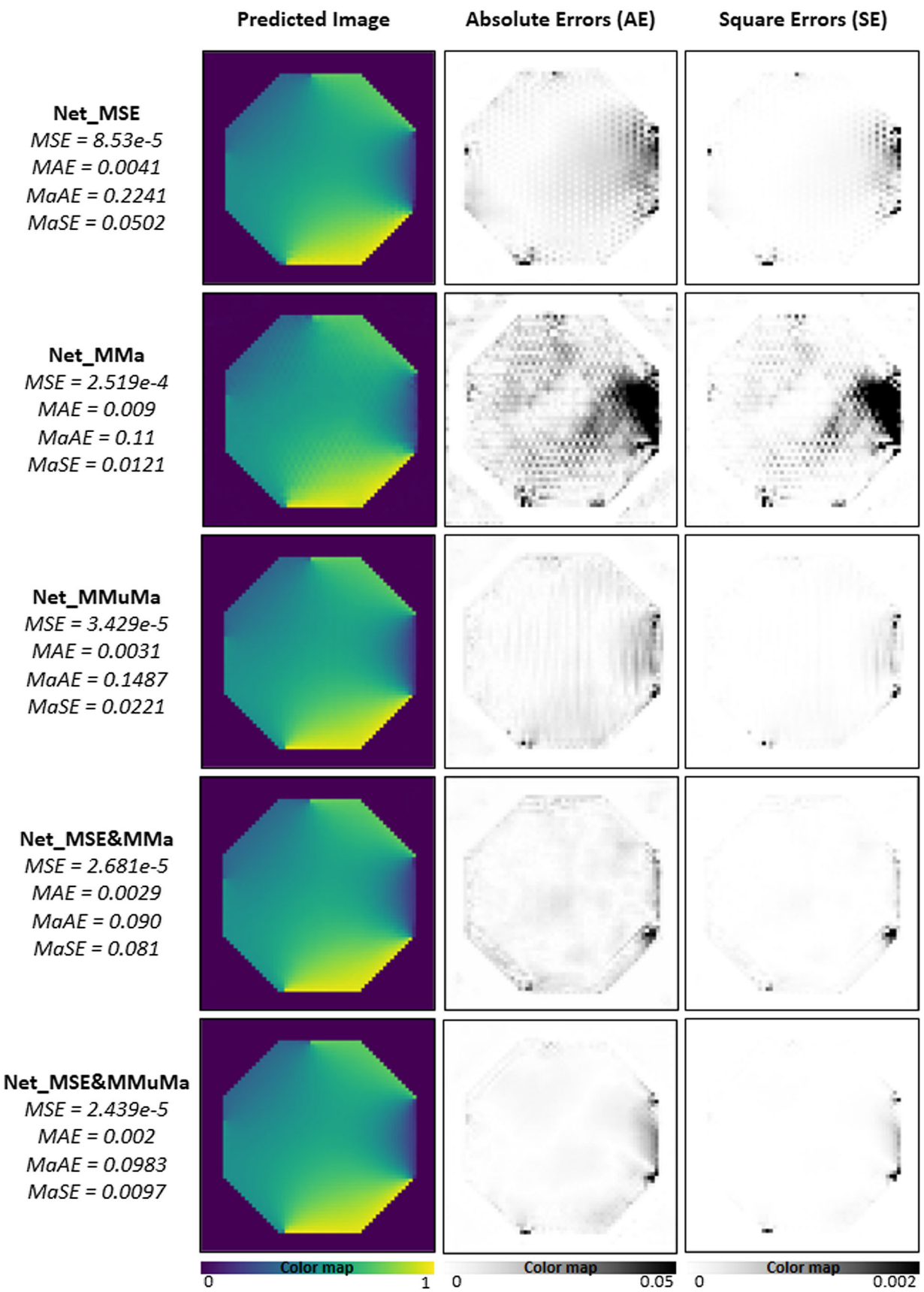


Fig. 11 Pie charts to compare outlier errors. These charts show the number of predicted test data pixels with AE equal and bigger than **A**: 0.05, **B**: 0.0988, **C**: 0.1638, and **D**: 0.245


Predicted Image
Absolute Errors (AE)
Square Errors (SE)
Net_MSE
MSE = 8.53e-5
MAE = 0.0041
MaAE = 0.2241
MaSE = 0.0502
Net_MMa
MSE = 2.519e-4
MAE = 0.009
MaAE = 0.11
MaSE = 0.0121
Net_MMuMa
MSE = 3.429e-5
MAE = 0.0031
MaAE = 0.1487
MaSE = 0.0221
Net_MSE&MMa
MSE = 2.681e-5
MAE = 0.0029
MaAE = 0.090
MaSE = 0.081
Net_MSE&MMuMa
MSE = 2.439e-5
MAE = 0.002
MaAE = 0.0983
MaSE = 0.0097
Color map
0
1
Color map
0
0.05
Color map
0
0.002

◀**Fig. 12** A sample of dataset predicted with all five networks. It has the worst MAE among all images in test data when predicting with Net_MSE. The input and output of these images are shown in the last row of Fig. 2. From left to right, columns 1 to 3 are predicted images with the network, absolute difference of predicted image and actual image in the dataset (AE), and square difference of predicted image and actual image in the dataset (SE). Rows 1 to 5 belong to Net_MSE, Net_MMa, Net_MMuMa, Net_MSE&MMa, and Net_MSE&MMuMa

7 Results and discussion

In this section, the impact of training DNN using different loss functions will be discussed. The accuracy of the trained DNNs for introduced evaluation indexes is the subject of the first subsection. Then, the speed of training, error evaluation, maximum error evaluation, and quality of predictions will be discussed.

7.1 Evaluation parameters comparison

After finishing the training process, the training, testing, and validation data were estimated, and validation parameters including MSE, MAE, MMaSE, and MMaAE were computed. Table 1 shows these results. Some interesting notes can be extracted from Table 1:

1. Net_MSE&MMuMa, DNN that trains with MSE + MMuMa lost function, has the least MSE error. Besides, Net_MSE&MMuMa has fewer MMaAE and MMaSE errors than Net_MSE; therefore, Net_MSE&MMuMa has a smaller outlier error than Net_MSE. Hence, a DNN, which is trained by MSE + MMuMaSE lost function, estimates the results with not only lower outlier errors but also lower mean error compared to a DNN trained with MSE. As a result, MSE + MMuMaSE is the best-introduced loss function and could be employed instead of MSE lost function. Interestingly, these results reveal that MSE + MMuMaSE is suitable for training a deep network to estimate not only physical images but also natural images.
2. The only loss function that estimates data with mean error (MSE and MAE) less than MSE loss function is MSE + MMuMaSE. Net_MSE + MMuMa reduced MSE by 67.1% compared to Net_MSE.
3. Net_MSE&MMa has the least outlier error (MMaSE and MMaAE). However, its mean error (MSE and MAE) is more than Net_MSE and Net_MSE&MMuMa. It predicts test images with an MSE 2.57% larger than Net_MSE. As a result, if the decrease in outlier error is more significant than the mean error, we could use MSE + MMaSE instead of MSE + MMaMuSE and MSE loss functions.

7.2 Speed of training

The main parameter to evaluate training parameters is the speed of reducing errors during the training process. The history of the training process will be used to investigate the rate of reduced errors. Using history, the variation of MSE of training data for the first 250 epochs is shown in Fig. 5 and for the last 200 epochs in Fig. 6. Also, Figs. 7 and 8 show the value of MMaSE for training data in the first 350 and last 200 epochs, respectively.

Figures 5 and 7 show that Net_MMa reduces errors slowly, especially in the first 200 epochs. Also, Fig. 5 shows that Net_MSE&MMuMa is the fastest network in the first 15 epochs to reduce errors.

Moreover, Figs. 6 and 8, which were plotted for the last 200 epochs, show the same results as the previous section. Figure 8 clearly shows the influences of the introduced loss functions on outlier errors. Interestingly, Net_MSE&MMuMa has a small MSE, and Net_MSE&MMa has a small MMaSE. It is worth noticing that MSE and MMaSE (outlier errors) are the highest for Net_MSE.

7.3 Population and distribution of testing images errors

As mentioned, testing data contains 6624 samples that each sample contains an input image and an output image. Evaluated parameters (MSE, MAE, etc.) could be calculated for either all predicted images or each image separately. In Table 1, evaluation parameters are computed for all estimated testing, training, and validation images. Now, a comparison is made based on MSE and MAE errors for each predicted test image separately. Figures 9 and 10 show the frequency distribution curve. In Fig. 9, each marker shows the number of test images predicted with MAE in a range of 5.7e-5. This chart is plotted for MAE between zero and 0.004. Figure 10 is similar to Fig. 9, but it is plotted for MSE from 0 to 3e-5; each marker show number of testing images estimated with MSE in a range of about 4.29e-07.

Figure 9 shows that Net_MSE&MMuMa has estimated a large number of test images with MAE in the distance between 0.0005 and 0.0012. It could also estimate a small number of test images in a range bigger than 0.0012. However, Net_MSE merely has a bit better performance than Net_Mse&MMuMa in the distance of 0 to 0.0004. It is concluded that Net_MSE&MMuMa estimates most images with small errors, while Net_MSE estimates some images

with near-zero error and some of the images with big errors. Hence, Net_MSE&MMuMa could estimate outputs with small errors, while Net_MSE estimates some images very well with near-zero error and some of them with big errors. Note that Net_MMa predicts almost all images with MAE with values bigger than 0.0035, and hence, they are almost out of scale in Fig. 9.

Unlike MAE that all errors have an equal influence on their value, bigger errors have more influence on MSE value. For instance, consider two set of data, (0.3, 0.3, 0.6) and (0, 0, 1.2). Though they have equal MAE, their MSE are 0.18 and 0.48, respectively. Hence, large errors affect MSE more than small errors, and MSE is a suitable parameter for comparing the magnitude of two network errors. Similar to Figs. 9, 10 shows the better performance of Net_MSE&MMuMa. According to Fig. 10, Net_MSE&MMuMa predicts almost 98% of testing images with MSE between 0 and 1e-5.

In summary, Net_MMa, Net_MSE&MMa, and Net_MMuMa do not have a proper performance to decrease mean error compared to Net_MSE and Net_MSE&MMuMa. In contrast, Net_MSE&MMuMa has better performance than Net_MSE to reduce mean and outlier errors.

7.4 Maximum errors

As mentioned, for physical images, correct estimation of even one pixel is important. In this section, all introduced loss functions are compared based on maximum estimation errors of test data. In Fig. 11, four pie charts are illustrated; each of them shows how many pixels are predicted equal and bigger than an AE value for different loss functions. The pie charts are plotted for AE of 0.05, 0.0988, 0.1638, and 0.245. For instance, Fig. 11D shows that Net_MSE&MMuMa predicts only four pixels of the test image with an absolute error equal and bigger than 0.245. Attention to Fig. 11 shows that using Net_MSE&MMa reduced the number of pixels with a high error of 0.1638 and more by 96%.

In the same way, Net_MSE&MMuMa reduced such errors by about 88%. The charts of Fig. 11 beautifully illustrate the influence of the introduced loss function on the outlier errors. These results show that Net_MSE&MMa was more successful than Net_MSE&MMuMa in reducing the outlier errors. However, as reported in Table 1, Net_MSE&MMuMa reduced not only the outlier errors, but also MSE in comparison with Net_MSE. It is while Net_MSE&MMa increases MSE. Therefore, MSE + MMuMaSE is a better loss function than MSE + MMaSE.

7.5 A sample of prediction

A sample of testing data predicted with the five networks is shown in Fig. 12. The input and output of this sample are represented in the last row of Fig. 4. It has the worst MAE value among all test images when they are predicted with Net_MSE. This image is predicted by Net_MSE, Net_MMa, Net_MMuMa, Net_MSE&MMa, and Net_MSE&MMuMa and then illustrated in rows 1 to 5 of Fig. 12, respectively. Column 1 shows the predicted image, column 2 is the absolute difference of actual and predicted image (Absolute Error: AE), and column 3 is the square difference of the actual and predicted image (Square Error: SE). As mentioned in the last section, a square error is a nonlinear function, and it blurs small errors and big amplitudes. Below each column, the color map and range of colors are shown. Evaluation parameters for the predicted image on each row are shown on the right side of Fig. 12.

8 Conclusion

The present study explored the impact of loss functions on the performance and accuracy of DNNs to estimate physical images. A DNN was introduced and trained with MSE loss function to estimate heat transfer images. Estimated images with this network showed that few pixels were predicted with unacceptable errors. These pixels are called outlier pixels, and their errors are called outlier errors. Although there are a few outlier pixels in a group of estimated images, they can lead to huge errors and mistakes in a decision process that uses these estimated images. In this paper, three new loss functions, including MMuMaSE, MSE + MMuMaSE, and MSE + MMaSE, were suggested to reduce outlier errors. Thus, the proposed DNN was trained with these new loss functions as well as MMaSE, which is presented in [22]. The following outcomes are extracted from estimation's results of DNN:

1. The network that was trained with MSE + MMuMaSE (Net_MSE&MMuMa) was estimated images with the minimum mean error compared to other networks. It reduced the MSE of estimation by about 67.1% in comparison to Net_MSE. So, it can be replaced with MSE.
2. The network that was trained with MSE + MMaSE (Net_MSE&MMa) had the least outlier errors. However, Net_MSE&MMuMa had more outlier errors than Net_MSE&MMa. Utilizing Net_MSE&MMa diminished prediction of pixels with large errors about 96%.
3. Comparing results revealed that MSE + MMuMaSE could estimate images with appropriate mean and outlier errors. A Net_MSE&MMa reduced the outlier

errors better than Net_MSE&MMuMa. However, Net_MSE&MMuMa reduced MSE in comparison with Net_MSE. This is while Net_MSE&MMa increases MSE by 2.57% compared to Net_MSE. Hence, Net_MSE&MMuMa is the best loss function that can be used to train deep networks to estimate both physical and natural images.

4. The heat distribution images that were estimated with Net_MSE&MMuMa had sufficient accuracy, and all of the pixels were estimated with small errors.
5. Outcomes of this paper reveal that MSE + MMuMaSE has a powerful ability to train DNNs. It could be used to train deep networks of physical and natural image estimators. A physical dataset was utilized in this paper, so using MSE + MMuMaSE to train DNNs with new structures and big datasets of natural images can be subject to future researches.

Declarations

Conflict of interest The authors clarify that there is no conflict of interest for report.

References

1. Wang J, Jiang J (2020) SA-Net: A deep spectral analysis network for image clustering. Neurocomputing 383:10–23
2. Chan T-H, Jia K, Gao S, Lu J, Zeng Z, Ma Y (2015) PCANet: A simple deep learning baseline for image classification? IEEE Trans Image Process 24(12):5017–5032
3. Qing Y, Zeng Y, Li Y, Huang G-B (2020) Deep and wide feature based extreme learning machine for image classification. Neurocomputing 412:426–436
4. Graves A, Mohamed A-r, Hinton G (eds) (2013) Speech recognition with deep recurrent neural networks. In: 2013 IEEE international conference on acoustics, speech and signal processing. IEEE
5. Pascual S, Bonafonte A, Serrà J (2017) SEGAN: speech enhancement generative adversarial network. arXiv:170309452.2017
6. Yıldırım Ö, Baloglu UB, Acharya UR (2018) A deep convolutional neural network model for automated identification of abnormal EEG signals. Neural Comput Appl. https://doi.org/10.1007/s00521-018-3889-z
7. Hemanth DJ, Deperlioglu O, Kose U (2020) An enhanced diabetic retinopathy detection and classification approach using deep convolutional neural network. Neural Comput Appl 32(3):707–721
8. Woźniak M, Siłka J, Wieczorek M (2021) Deep neural network correlation learning mechanism for CT brain tumor detection. Neural Comput Appl. https://doi.org/10.1007/s00521-021-05841-x
9. He K, Zhang X, Ren S, Sun J (eds) (2015) Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. In: Proceedings of the IEEE international conference on computer vision
10. Han K, Mun YY, Gweon G, Lee J-G (eds) (2013) Understanding the difficulty factors for learning materials: a qualitative study. In: International conference on artificial intelligence in education. Springer
11. Ioffe S, Szegedy C (2015) Batch normalization: Accelerating deep network training by reducing internal covariate shift. arXiv:150203167.2015
12. Agarap AF (2018) Deep learning using rectified linear units (relu). arXiv:180308375.2018
13. Raissi M, Yazdani A, Karniadakis GE (2020) Hidden fluid mechanics: Learning velocity and pressure fields from flow visualizations. Science 367(6481):1026–1030
14. Berg J, Nyström K (2018) A unified deep artificial neural network approach to partial differential equations in complex geometries. Neurocomputing 317:28–41
15. Lin Q, Hong J, Liu Z, Li B, Wang J (2018) Investigation into the topology optimization for conductive heat transfer based on deep learning approach. Int Commun Heat Mass Transf 97:103–109
16. Liu X, Zhang H, Kong X, Lee KY (2020) Wind speed forecasting using deep neural network with feature selection. Neurocomputing 397:393–403
17. Sharma R, Farimani AB, Gomes J, Eastman P, Pande V (2018) Weakly-supervised deep learning of heat transport via physics informed loss. arXiv:180711374.2018
18. Ciresan D, Giusti A, Gambardella LM, Schmidhuber J (eds) (2012) Deep neural networks segment neuronal membranes in electron microscopy images. Advances in neural information processing systems
19. Bergman TL, Incropera FP, Lavine AS, DeWitt DP (2011) Introduction to heat transfer. John Wiley & Sons, USA
20. Farimani AB, Gomes J, Pande VS (2017) Deep learning the physics of transport phenomena. arXiv:170902432.2017
21. Edalatifar M, Tavakoli MB, Ghalambaz M, Setoudeh F (2020) A dataset for conduction heat transer and deep learning. Mendeley Data 1:10–7632
22. Edalatifar M, Tavakoli MB, Ghalambaz M, Setoudeh F (2020) Using deep learning to learn physics of conduction heat transfer. J Therm Anal Calorim 146:1435–1452
23. Nadipally M (2019) Optimization of methods for image-texture segmentation using ant colony optimization. In: Hemanth DJ, Gupta D, Emilia Balas V (eds) Intelligent data analysis for biomedical applications. Academic Press, Cambridge, pp 21–47
24. Botchkarev A (2018) Performance metrics (error measures) in machine learning regression, forecasting and prognostics: properties and typology. arXiv:180903006.2018
25. Masci J, Meier U, Cireşan D, Schmidhuber J (eds) (2011) Stacked convolutional auto-encoders for hierarchical feature extraction. In: international conference on artificial neural networks. Springer
26. Wu C, Khishe M, Mohammadi M, Taher Karim SH, Rashid TA (2021) Evolving deep convolutional neutral network by hybrid sine–cosine and extreme learning machine for real-time COVID19 diagnosis from X-ray images. Soft Comput. https://doi.org/10.1007/s00500-021-05839-6
27. Hu T, Khishe M, Mohammadi M, Parvizi G-R, Taher Karim SH, Rashid TA (2018) Real-time COVID-19 diagnosis from X-Ray images using deep CNN and extreme learning machines stabilized by chimp optimization algorithm. Biomed Signal Process Control 68:102764
28. Rashid TA, Abbas DK, Turel YK (2019) A multi hidden recurrent neural network with a modified grey wolf optimizer. PLoS ONE 14(3):e0213237
29. Rashid TA, Fattah P, Awla DK (2018) Using accuracy measure for improving the training of LSTM with metaheuristic algorithms. Procedia Comput Sci 140:324–333

30. Jabar AL, Rashid TA (2018) A modified particle swarm optimization with neural network via euclidean distance. Int J Recent Contrib Eng Sci IT (iJES) 6(1):4–18
31. Lemaréchal C (2012) Cauchy and the gradient method. Doc Math Extra 251(254):10
32. Cheridito P, Jentzen A, Riekert A, Rossmannek F (2022) A proof of convergence for gradient descent in the training of artificial neural networks for constant target functions. arXiv preprint arXiv:210209924.2021.
33. Rumelhart DE, Hinton GE, Williams RJ (1986) Learning representations by back-propagating errors. Nature 323(6088):533–536
34. Goodfellow I, Bengio Y, Courville A (2016) Deep learning. MIT Press, Cambridge
35. Abadi M, Agarwal A, Barham P, Brevdo E, Chen Z, Citro C et al (2015) TensorFlow: large-scale machine learning on heterogeneous systems
36. Kingma DP, Ba J (2014) Adam: a method for stochastic optimization. arXiv preprint arXiv:14126980.2014
37. Glorot X, Bengio Y (eds) (2010) Understanding the difficulty of training deep feedforward neural networks. In: Proceedings of the thirteenth international conference on artificial intelligence and statistics