Compact Representation and Identification of Important Regions of Metal Microstructures Using Complex-step Convolutional Autoencoders

Dharanidharan Arumugam[1] and Ravi Kiran[2]

Abstract

In this study, we propose a complex-step convolutional autoencoder to identify the regions that are important in a metal microstructure for compact representation and secure sharing. Firstly, the architecture of a convolutional autoencoder is designed for the compact representation of microstructural images. The designed autoencoder achieved a high image compression ratio of 32 without loss of important information. Secondly, an in-home developed model agnostic sensitivity analysis using complex step derivative approximation is implemented on convolutional autoencoders to identify regions of the microstructure that are important for reconstruction. Finally, saliency maps that highlight the importance of pixels for reconstruction are generated for three grades of dual-phase structural steels. The saliency maps indicated secondary phase regions and grain boundaries are important for microstructure image reconstruction. The proposed approach produces more tenable saliency explanations compared to guided backpropagation and layer wise relevance propagation methods. The decoder part of the convolutional autoencoder can be used as a key that could be used to reconstruct the actual microstructure from encoded image information contributing to secure and efficient sharing of microstructure data. The proposed framework is generic and can be extended to identify important microstructural regions for other metals, composites, biomaterials, and material systems.

Keywords: Interpretable AI; ASTM A992; Saliency maps; Pixel relevance; and Convolutional autoencoder.

1. Introduction

The microstructure of a metal dictates the metal's Young's modulus [1, 2], hardness [3, 4], yield strength [5, 6], ductility [7, 8], and in-fire and postfire properties [9, 10]. The microstructure is comprised of microscale constituents such as metallurgical phases, non-metallic defects, and microvoids. The characteristics of these constituents such as average grain size, grain orientation, and composition are determined by the thermos-mechanical manufacturing route, alloy composition, etc. and will govern the constitutive response and fracture behavior of metals [8, 11-13]. Optical microscopy and SEM analysis are widely used to extract microstructural images from metallographic specimens to perform microstructural analysis and characterization. Modern SEMs and optical microscopes are capable of rapidly generating large volumes of high-resolution image

[1] Graduate Research Assistant, Dept. of Civil, Construction & Environmental Engineering, North Dakota State University, Fargo, ND 58105, email: d.arumugam@ndsu.edu
[2] Assistant Professor (corresponding author), Dept. of Civil, Construction & Environmental Engineering, North Dakota State University, Fargo, ND 58105, email: ravi.kiran@ndsu.edu

data. For instance, ZEISS MultiSEM microscope can image a square centimeter material sample area at 4nm pixel level in less than 3 hours and can generate 1.5 terabytes of data per hour. Neural networks (NNs) and machine learning (ML) algorithms are now commonly employed to automate microstructural characterization and to take advantage of the large volumes of microstructural data generated through these modern instruments [14-16]. However, handling of large volumes of microstructural data still poses storage and transfer challenges. A survey conducted among 7700 researchers by springer in 2018 [17] shows 56% of researchers identify copyright issues and cost of sharing are major barriers to data sharing.

One effective and economical solution to storage and transfer problems is the use of downsized data. There exists several lossy image compression methods [18] which can achieve highly compressed image data by losing some image information. However, these methods do not identify and preserve information critical to microstructural characterization and can lose information essential to valid microstructural analysis. Alternatively, deep learning or machine learning methods can be used to obtain lower dimensional (compact) representation of image data. Larmuseau et al. achieved a compact representation of microstructure by combining three neural network structure [19]. The employed network is trained for categorizing different classes of microstructure. Discriminative convolutional neural networks proposed in [20] can shrink the overall feature space spanned by the image data and offer a discriminative lower-dimensional representation of the data. These methods are solely focused on the compact representation of data for classification tasks and do not provide a way to recover the original data and thus facilitate a secure transfer of data. Also, NNs and ML algorithms used for microstructural characterization of compact representation do not provide any insights that could be used by a microscopy expert on the regions that should be captured with high fidelity at the microscopy stage. In other words, the existing methods can only automate the material characterization but cannot inform the analyst on the regions of the microstructure that can improve the fidelity of characterization. The goal of this study is to achieve a compact representation of metal microstructures, identify important regions of metal microstructure and develop a new way to safe and secure transfer of micrograph data.

The research questions addressed in this article are: 1) How to compactly represent the metal microstructure images without losing important information? 2) what are the regions of the microstructure that are crucial for microstructure image reconstruction? 3) what regions of microstructure are important for high-fidelity microstructural characterization and classification of different classes of microstructure and 4) how to safely share large volume of microstructural image data? In this study, we propose a convolutional autoencoder (see Section 2) architecture to compactly represent the metal microstructure and the relevant procedure is detailed in Section 3. A complex-step sensitivity analysis procedure to identify the important regions of the metal microstructure is described in Section 4. Sharing the compact representations of the metal microstructure securely is discussed in Section 5. The generated saliency maps for identifying important pixel regions are explained in Section 6. Saliency explanations produced by the proposed method is compared with existing methods in Section 7. Finally, the conclusions drawn from this study are provided in Section 8.

2. Convolutional autoencoders: applications and mathematical background

Autoencoders are a class of self-supervised neural networks whose main function is to compress the input data to a very high degree and reconstruct the input from the compressed data with a minimal loss of input information [21, 22]. Some important applications of autoencoders include dimensionality reduction of data [23, 24], detection of anomalies [25, 26], extraction of salient features for classification [27, 28], and generation of artificial images [29, 30]. Fig. 1 shows a schematic representation of the architecture of an autoencoder. Sparse [31], contractive [27], denoising [32], and variational autoencoders [33] are some important types of autoencoders.

Convolutional autoencoder (CAE) is a special form of autoencoder [34] which combines the ability of CNNs in efficiently extracting image features and the ability of autoencoders in compactly representing the input data. CAEs have several interesting applications in the field of mechanics. Kim et. al. [35] employed a convolutional variational autoencoder (CVAE) to generate continuous dual-phase microstructure and generated the compact latent features to relate them with yield strength, ultimate tensile strength, and toughness. Lee et. al. [36] predicted the stress state of aluminum alloys and stainless subjected to four-point bending using a CVAE by combining the experimental and finite element behavior in the compact latent space. In the study conducted by Xu and Duraisamy [37], a CAE is successfully applied to predict fluid flow dynamics. The flow data is passed through two levels of CAEs. The first CAE is used to capture the spatial relations and its latent features are used as inputs to the second CAE to capture the temporal evolution. CAEs are also used for the non-linear decomposition of fluid flow by employing two decoder networks [38]. More applications of CAEs in mechanics can be found elsewhere [39-42].

Autoencoders are designed to reconstruct the input by combining two neural network blocks: an encoder and a decoder (refer Fig. 1). The encoder block of the network encodes an input $\boldsymbol{x}$ to a latent output $\boldsymbol{l}$ which is a compressed representation of the input. The dimensionality of the latent output $\boldsymbol{l}$ is usually very small compared to the dimensionality of input $\boldsymbol{x}$. Decoder then builds on the latent output, $\boldsymbol{l}$, and generates output, $\widetilde{\boldsymbol{x}}$, as close as possible to the input, $\boldsymbol{x}$. Autoencoders are trained to minimize the reconstruction loss of network for adequate reconstruction of the original input.

In mathematical terms, an autoencoder consists of an encoder, $f(\boldsymbol{x}, \boldsymbol{\theta})$ that compresses the data and a decoder, $g(\boldsymbol{l}, \boldsymbol{\phi})$ that reconstructs the input in the following way

$$\boldsymbol{l} = f(\boldsymbol{x}, \boldsymbol{\theta}) \tag{1a}$$

$$\widetilde{\boldsymbol{x}} = g(\boldsymbol{l}, \boldsymbol{\phi}) = g(f(\boldsymbol{x}, \boldsymbol{\theta}), \boldsymbol{\phi}) \tag{1b}$$

where, $\boldsymbol{\theta}$ and $\boldsymbol{\phi}$ are trainable parameters of encoder and decoder, respectively. These parameters are fixed by training the autoencoder to minimize the cost function ($\mathcal{C}$)

$$\mathcal{C}(\boldsymbol{x}, \widetilde{\boldsymbol{x}}) = \mathcal{L}(\boldsymbol{x}, \widetilde{\boldsymbol{x}}) + \textit{regularization term} \tag{2}$$

here, $\mathcal{L}(\boldsymbol{x}, \widetilde{\boldsymbol{x}})$ is a reconstruction loss function which is, in general, a mean squared error (MSE) or binary cross-entropy (BCE) averaged over the total number of instances which are calculated as

$$\mathcal{L}_{MSE} = \frac{1}{N}\sum_i^N \left(\boldsymbol{x}^i - \widetilde{\boldsymbol{x}}^i\right)^2 \tag{3a}$$

$$\mathcal{L}_{BCE} = \frac{1}{N}\sum_i^N -\left[\boldsymbol{x}^i \log\left(\widetilde{\boldsymbol{x}}^i\right) + \left(1 - \widetilde{\boldsymbol{x}}^i\right)\log\left(1 - \widetilde{\boldsymbol{x}}^i\right)\right] \tag{3b}$$

The regularization term in Eq. 2 is necessary for deep neural networks to prevent the network from learning identity function and therefore improve the model's ability to generalize the results [21, 31, 43]. Shallow network autoencoders need not involve any regularization term since the bottleneck structure (refer to Fig. 1) itself is sufficient to enforce the regularization [22]. Table 1 shows regularization terms involved in different types of autoencoders. In the case of CAEs, the use of pooling layers prevents the overfitting of network and improves the generalizing ability of the model [44, 45]. Thus, the reconstruction loss can be simply used a cost function [46, 47]. The encoder of a typical convolutional autoencoder consists of alternative layers of convolutions and pooling for most of the network and fully connected layers in the end. A latent layer is usually represented as a fully connected layer (one-dimensional layer). The decoder majorly consists of a series of transposed convolutional layers with some fully connected layers at the start. However, transposed convolutions are shown to have a tendency of producing checkerboard artifacts [48], and hence they are now replaced with alternative layers of upsampling and convolutions. A convolutional autoencoder involves three main operations namely convolution, pooling, and upsampling which are explained subsequently.

2.1 Feature Extraction in CAE through convolutions

Convolution is the process of extracting spatially connected features present in the images with the use of convolutional filters. A convolutional filter comprises of a stack of kernels and the number of kernels present in a filter corresponds to the number of input channels. For example, an RGB image input consists of three channels and requires filters with three kernels whereas grey image input has only one kernel. A kernel is usually a 2D array of learnable weights with a size, $k \times k$. Here, k refers to kernel size and it is computationally rewarding to keep it very small compared to the size of the image input. Though a kernel can have different sizes, a kernel size of 3×3 is standardly used. Larger kernel sizes (sizes more than 3) increase computational costs and make the model less generalizable whereas smaller kernel sizes (sizes less than 3) do not extract high-level image features [49, 50].

Each kernel of a convolutional filter operates on its conjugal channel array of input data and results in individual array outputs corresponding to each kernel-channel pair. Each element in a convolved output array is a sum of Hadamard product between the weights of a kernel and a subarray of input data [51]. The subarray is systematically chosen by sliding a kernel window across both the dimensions of the input array and considering the region of the input spanned by the window at every step. The top-left corner of the kernel window starts from the top-left corner of the input and moves in steps from left to right until the last column is reached, then moves down and again from left to right. This is repeated until the last row of the input array is covered. The step size of the slide movement is called 'stride' and it is of unit step size, in general, for kernel filters. The individual output arrays corresponding to each channel are then summed together and the resulting array is called a feature map. Fig. 2 illustrates the generation of a feature map through convolution.

Inputs to convolutions are usually ‘padded’ which implies the addition of calculated numbers of columns and rows of zeros around the inputs as symmetrical as possible. Padding is necessary when the information available around the edges of the input image is valuable. Padding is generally of two types namely ‘Valid’ and ‘Same’ padding [52]. ‘Valid’ padding is another term for zero padding whereas the ‘Same’ padding refers to the type of padding which results in the convolved output the same size as that of the input (refer to Fig. 3). Feature maps pass through an activation function to impart nonlinearity in the convolutional process. ‘ReLU, ‘Leaky ReLU’, ‘Tanh’, and ‘Sigmoid’ are the widely used activation functions for convolutional autoencoders. The functional expressions for these activation functions are summarized in Table 2. However, for the output layer, ‘Sigmoid’ and ‘Linear’ activation functions are generally used.

A convolutional layer usually consists of several convolutional filters which operate on the input data individually and results in corresponding output feature maps. The number of filters is chosen based on the nature and complexity of the problem and it is usually varied along the procession of neural layers. Increasing the number of filters in a layer increase the extent of abstractions extracted from the input image or feature map. In the case of the encoder, the number of filters is incrementally decreased to obtain only important lower-dimensional features. The filters are then again incrementally increased until the final layer to regenerate the image features of the input.

2.2 Removal of redundancy in CAE using pooling operations

Convolution is followed by a pooling operation in the encoder block of the convolutional autoencoder. Pooling is essentially performed to down sample the feature maps. Pooling aggregates the important abstractions available in the features maps and produces coarser maps that are spatially invariant. This is essential in the case of dimensionality reduction. Pooling also significantly reduces the computational overhead and prevents the overfitting of the model.

Pooling operates on the subarrays of the input and involves a pooling window to select the subarrays similar to the convolutional operation. The pooling window slides across the width and height (from left to right and top to bottom) of the input array with ‘stride’ as the step size. Max pooling and average pooling are the two most popular pooling operations. Max pooling returns the maximum values of subarrays whereas the average pooling returns the mean values of subarrays. Accordingly, max-pooling extracts sharp and prominent features from the input while average pooling captures more complexities present in the input. In the case of convolutional autoencoders, max pooling is preferred to average pooling. However, not much difference is observed between the use of these two functions in terms of the performance of the model. Fig. 4 provides an illustrative example for both max and average pooling. In the example, a standard pooling window size of 2×2 and stride 2 is used and it is evident from the example, that it halves the size of the input along both the dimensions.

2.3 Image regeneration in CAE using upsampling

Upsampling is performed to reverse the effect of pooling. Upsampling is mainly used in the convolutional autoencoder, particularly in the decoder network, to expand the input information and thereby regenerate the original input image. Upsampling in itself does not produce any new information, it just repeats the available input information and increases the image size. In order to build the required features for image generation, it has to be followed by a convolutional layer. In upsampling, each value of the input array gets repeated depending on the size of the upsampling window size. An example that demonstrates upsampling operation is provided in Fig. 5. As seen in the example, upsample window size of 2×2 increases the dimensions of the input by 2. It is important to note that both pooling and upsampling operations do not involve any learnable parameters, unlike convolutions. Therefore, the training of convolutional autoencoders is basically to find the appropriate weights and biases involved with the convolutional layers for adequate image reconstruction.

3. Methodology

A five-step procedure is employed to accomplish the objectives of this study: (1) acquisition of microstructural images of structural steel metallographic specimens, (2) preparation of input data from the acquired microstructural images, (3) configuration and training of a convolutional autoencoder, (4) formation of the convolutional encoder network from the trained convolutional autoencoder network, and (5) computation of pixel relevance and construction of the saliency maps. The schematic of the steps involved is also shown in Fig. 6. These steps are further explained in detail in the following sub-sections.

3.1 Acquisition of microstructural images

The microstructural images used in this work were acquired from metallographic specimens extracted from three different structural steels popularly used in the US – ASTM A36, ASTM A572, and ASTM A992 [53]. All these steels are ferrite-pearlite steels [9, 53]. Seven metallographic specimens were obtained from each type of structural steel totaling twenty-one metallographic specimens. Six specimens of each grade were exposed to elevated temperatures of 500°C, 600°C, 700°C, 800°C, 900°C, and 1000° with a one-hour soaking period to alter the microstructure of the parent metal. After the soaking, the specimens were allowed to undergo natural cooling at room temperature. The remaining specimen of each grade was not subjected to any heat treatment. The images were gathered from 5 to 7 random locations on each specimen using an Amscope® optical microscope. The images were captured at 50X magnification. More details on the preparation of these microstructural images can be found in [9, 54, 55]. In the end, a total of 124 microstructural images with a size of 2080 × 1542 pixels were acquired.

3.2 Preparation of input data for training and testing purposes

The original microstructural images of size 2080 × 1542 pixels are segmented into images of size 256 by 256. Since the segmented images are of small input size and captures the microstructural pattern of the whole unsegmented image, the segmentation helps in obtaining leaner network architecture. The segmentation resulted in 5,704 greyscale images with pixel values ranging from 0 to 255. The segmented images are converted into numerical arrays and partitioned into a training dataset ($\boldsymbol{D}_{train} \in \mathbb{R}^{n\times p\times q\times c}$) and a testing dataset ($\boldsymbol{D}_{test} \in \mathbb{R}^{m\times p\times q\times c}$) with 80:20 split. Here, n and m denote the number of image examples in the training and testing dataset which are 4564 and 1140, respectively, in our case, $p \times q$ denotes the dimension of an input image array which is 256 × 256 and c denotes the number of channels which is 1 since the images are of greyscale. A typical k^{th} example of the dataset (called an instance) is defined as

$$\boldsymbol{X}_k = \begin{matrix} x_{11}^{(k)} & \cdots & x_{1q}^{(k)} \\ \vdots & \ddots & \vdots \\ x_{p1}^{(k)} & \cdots & x_{pq}^{(k)} \end{matrix}$$

where $x_{11,}^{(k)} \cdots , x_{pq}^{(k)}$ are the pixel features and k ranges from 1 to n or m . The grey pixel values of both datasets were then normalized. Normalization involves scaling down the range of pixel intensity values from 0 to 255 to 0 to 1 to improve the performance of the neural network.

3.3 Configuring and training the convolutional autoencoder

In this study, a CAE is employed for the compact representation of the input microstructural images (in the form of arrays). Detailed background on CAEs is presented in Section 2. The CAE network was duly configured to ensure the autoencoder reconstructs the images with minimal loss of input information not only for the learned examples but also for the unseen data. The configuration of the network involves a rigorous selection of a network architecture which kept the reconstruction loss of the network (mean squared error) below 0.0035 during both the training and testing stage. At every trial selection of network architecture, the network was trained for 50 epochs with an input batch size of 32. The training process involves fine-tuning the network parameters, i.e., weights and biases through a stochastic gradient descent algorithm called 'Adam' (Adaptive moment estimation) [56]. The parameters used for the 'Adam' optimization algorithm are presented in Table 3. The final configured CAE consists of 50,353 network parameters.

The CAE model employed in this study was developed using TensorFlow 2.8.0 open-source software library [57]. The encoder block starts with an input layer which receives the input of size 32 × 256 × 256 × 1. The first dimension of the input indicates the batch size (the number of image inputs per batch). The input layer is followed by four numbers of convolutional layers alternated with max-pooling layers. The final max-pooling layer of the encoder outputs the latent variables of dimension 32 × 16 × 16 × 8. Normally, one or more fully connected layers are added at the end of an encoder to construct the latent representation as a one-dimensional tensor, however, the addition of fully connected layers lead to model overfitting (results in large reconstruction loss in the testing stage)[58]. Hence, in our study, we did not use any fully connected layers and maintained the latent variables in the form of feature maps (a three-dimensional tensor). The decoder block takes the latent variables tensors as inputs and reconstructs the original inputs. It comprises four alternating layers of upsampling and convolutional layers with an additional convolutional layer at the end. We used a standard kernel filter size of 3×3 with a stride of 1 and the 'same' padding type for all the convolutional operations. The 'same' padding type allows the model to retain the information near image boundaries. Max pooling operations that halve the inputs along both the dimensions use the standard pool size of 2×2 with stride 1. All convolutional layers use the 'ReLU' activation function except for the final convolutional layer of the decoder block which uses the 'Linear' activation function. The use of 'Linear' activation instead of the 'Sigmoid' function improved the convergence and reconstruction loss of the model. The schematic of the model is shown in Fig. 7 and the model details are summarized in Table 4 and Table 5.

3.4 Formation of the convolutional encoder network from the convolutional autoencoder

A convolutional neural network (CNN) with the same architecture as that of the encoder part of the convolutional encoder was constructed to perform sensitivity analysis using complex step derivative approximation method. The newly constructed CNN takes microstructural images as input and generates latent variable tensors as outputs. The layer modules available in the TensorFlow libraries were customized to accept complex inputs. Accordingly, the CNN uses customized layers such as 'xConv2D' and 'xMaxPool2D' for convolution and max pooling, respectively, on complex inputs. Also, for activation, a modified ReLU function, 'xReLU', was used to activate complex inputs. The weights and biases of the CNN network were transferred from the encoder block of the autoencoder.

3.5 Computation of pixel relevance and construction of the saliency maps

The relevance (attributions) of the pixels ($\boldsymbol{r} = \boldsymbol{s}(\boldsymbol{x})$) of an input image to latent variables ($\boldsymbol{l} = \boldsymbol{f}(\boldsymbol{x})$) were computed using the sensitivity of latent variables to the input pixels. Here, $\boldsymbol{s}$ is the sensitivity function, $\boldsymbol{x} \in \mathbb{R}^{256\times256}$ is an input image array and $\boldsymbol{l} \in \mathbb{R}^{16\times16\times8}$ is a tensor of latent variables associated with $\boldsymbol{x}$, $\boldsymbol{r}$ has the same size as that of input image array $\boldsymbol{x}$ and every element in $\boldsymbol{r}$ maps the relevance or attribution of every element in $\boldsymbol{x}$ as seen in Fig. 8.

Sensitivities were determined through the gradient of latent variables with respect to input pixel features. Thus, the relevance of a pixel feature (r_{ij}) is mathematically defined as the Frobenius norm of the gradient tensor ($\boldsymbol{g}^{ij} = \partial\boldsymbol{l}/\partial x_{ij}$) given as

$$r_{ij} = \left\|\boldsymbol{g}^{ij}\right\|_F = \sqrt{\sum_{u=1}^{16}\sum_{v=1}^{16}\sum_{w=1}^{8}(g_{uvw}^{ij})^2} \tag{5}$$

$$\text{where, } g_{uvw}^{ij} = \partial l_{uvw}/\partial x_{ij},\ u = 1 \ to\ 16, v = 1\ to\ 16 \text{ and } w = 1\ to\ 8 \tag{6}$$

The gradient tensor for each input pixel feature was determined using the complex step derivative approximation (CSDA) method. The mathematical basis involved with the CSDA method is comprehensively explained in Section 4. Computation of gradient tensor of an input pixel feature x_{ij} using the CSDA method involves three steps: 1. Creating a perturbed input array ($\acute{\boldsymbol{x}}$) by adding a very small complex perturbation (ih) to x_{ij}, 2. Determination of latent variables ($\acute{\boldsymbol{l}}$) by feedforwarding the perturbed input array ($\acute{\boldsymbol{x}}$) and 3. Calculation of partial derivatives by dividing imaginary component of obtained complex latent variables with the step size $(\boldsymbol{g} = imag(\boldsymbol{\acute{l}})/h)$. We used a step size (h) of 10^{-24} in our analysis to obtain analytical level accurate gradients. This is further illustrated in Fig. 9.

4. Complex step derivative approximation

In this study, the partial derivatives of latent features with respect to the input pixel features of a convolutional autoencoder are determined using complex step derivative approximation (CSDA). We employed CSDA for its various advantages over its numerical counterparts and also based on the experience we derived from our previous studies [59-62]. One of the important advantages of CSDA is that it is automatable which means that a common step size (h) (which can be very small in the order of 10^{-24}) can be assumed to find an accurate derivative of any function. This is due to the fact that CSDA is free of subtractive cancellation errors which also allows it to obtain analytical level accurate derivatives. In the case of discontinuous functions, CSDA still can find one-sided derivatives which are very useful in the case of neural networks since it employs many discontinuous activation functions [63]. CSDA can also be used to determine first-order and second-order derivatives of tensor functions [64].

CSDA is derived from expanding an analytical function, f, about a very small complex step size, ih (i is a unit imaginary component and h is the step size) using Taylor's series

$$f(x + ih) = f(x) + ihf'(x) - \frac{h^2}{2!} f''(x) - \frac{h^3}{3!} f'''(x) + \cdots \tag{7}$$

Rearranging Eq. 7 and using a quadratic error term, $o(h^2)$, we obtain:

$$f'(x) = \frac{Imag\big(f(x + ih)\big)}{h} + o(h^2) \tag{8}$$

here, $Imag(.)$ extracts the imaginary component of a function.

Finally, the expression in Eq. 9 approximates the first-order derivative of a scalar function after the quadratic error term in Eq. 8 is truncated. It is evident from Eq. 8 that we can obtain analytical precise derivatives by keeping the step size, h, very small (i.e., in the order of 10^{-24}).

$$f'(x) \approx \frac{Imag\big(f(x + ih)\big)}{h} \tag{9}$$

CSDA can be further extended to multivariable functions by expanding the function about one variable at a time. For example, using the above approach we can arrive at the following expression to determine the partial derivative of a k^{th} variable of a multivariable function:

$$\frac{\partial f}{\partial x_k} \approx \frac{Imag\big(f(x_1, x_2 \ldots, x_k + ih, \ldots, x_n)\big)}{h} \tag{10}$$

In the case of multi-output functions, the first derivative(s) of each output function can be obtained using Eq. 9 or Eq. 10 independently and therefore can be computed parallelly.

With this mathematical basis, we have implemented CSDA to convolutional autoencoders in this study. The functional relation between inputs and outputs is implicit in neural network models and is developed through data training. Hence, CSDA is applied to the trained and configured network model. Since the network is trained with real values, a new model with the same architecture and network parameters as that of the trained network but with complex layers and activation functions is used. In our study, we used CSDA on the encoder network, hence we modified convolution, max pooling, and activation functions to deal with complex inputs. Convolutions on the complex inputs are performed as follows

$$conv(\tilde{\boldsymbol{x}}) = conv(Re(\tilde{\boldsymbol{x}})) + iconv(Im(\tilde{\boldsymbol{x}})) \tag{11}$$

where $conv(.)$ is a convolutional operator and $\tilde{\boldsymbol{x}}$ is a complex input array**.** $Re(.)$ and $Im(.)$ are real and imaginary components of inputs respectively**.**

Max pooling on complex inputs involves a complex summation of the maximum value of real components of $\tilde{\boldsymbol{x}}$ and their conjugal imaginary components**.** The implementation of it is shown below:

$$\begin{gathered} maxpool(\tilde{\boldsymbol{x}}) = maxpool\big(Re(\tilde{\boldsymbol{x}})\big) + i\ conjugal\_Im(\tilde{\boldsymbol{x}}) \\ conjugal\_Im(\tilde{\boldsymbol{x}}) = Im(\tilde{\boldsymbol{x}})[argmaxpool\big(Re(\tilde{\boldsymbol{x}})\big)] \end{gathered} \tag{12}$$

$argmaxpool(*)$ the function finds the locations of max pool values and $[*]$ function extracts elements of an array corresponding to the input locations.

The ReLU activation function used in the convolutional layers is modified as follows

$$\phi(z) = \begin{cases} 0 + 0i & if\ Re(z) \leq 0 \\ Re(z) + Im(z) & if\ Re(z) > 0 \end{cases} \tag{13}$$

where, z is the activated scalar input, y is the activated output.

The encoder network of a CAE takes multiple inputs (pixel features) and returns multiple outputs (latent features). Based on Eq. 10, partial derivatives of latent features with respect to each pixel feature are evaluated one at a time. At each evaluation, the pixel feature of interest is added with a complex component (ih), then the pixel inputs are feedforwarded to obtain the corresponding complex latent outputs. The partial derivative of each latent feature is parallelly computed by extracting the imaginary components of complex latent features and dividing them with the step size (h) as per Eq. 10.

5. Compact representation and secured sharing of metal microstructure

The input image of size 256 × 256 × 1 is compactly represented to a latent size of 16 × 16 × 8 using the CAE designed in this study. A compression ratio of 32 was achieved without losing any important information of the microstructure. In other words, an input image was able to be reconstructed just from 3.125% of the original information. Fig. 10 shows some examples of the reconstructed microstructural images together with the original images. It is to be noted that the compact (latent) representation of metal microstructure is an encoded version of metal microstructure. To retrieve back the original information from the latent representation ($\boldsymbol{l}$), the decoder network with the trained weights and biases is required. It is highly unlikely to reconstruct the input from the latent features without knowing the decoder architecture and its weights and biases. Accordingly, the decoder network acts as a security key to decode the latent representation and thus provide a highly safe and secured way of sharing the metal microstructure information.

6. Saliency maps of metal microstructure

A set of microstructural images was selected and relevance matrices of those images were generated using the procedure laid out in Section 3. The elements of a relevance matrix are basically the sensitivities of latent variables to input pixel features of a microstructural image. The high sensitivity of latent variables to an input pixel indicates that a specific input pixel is highly important in generating the respective output of latent variables. Heat maps were plotted using these relevance arrays to gain insights on the latent representation of microstructural images so that pixel regions which are important for reconstruction are identified. These heat maps are known as saliency maps in the literature [65, 66]. Saliency maps are useful in understanding the importance of inputs to the network predictions. In addition to acting as an interpretability tool, salient maps are also play an important role in object detection [67]. The saliency maps developed in the current study highlight the relevance of the input pixels for their latent representation. In other words, the saliency maps highlight the regions of input information passed on to the latent space and used for reconstruction. Fig. 11 shows some examples of saliency maps developed along with their input microstructural images. The saliency maps are smoothened using a gaussian filter with a standard deviation of 1. The maps are plotted using a blue-white-red divergent color map. Thus, red color indicates high relevance, white color indicates moderate relevance and blue color indicates low or no relevance of input pixels.

The saliency maps consistently show the darker pearlite regions (secondary phase regions) and the microstructural boundaries as highly relevant and the greyer ferrite regions as slightly or not relevant for the convolutional autoencoder in constructing the latent representation. The obtained results are consistent with the way autoencoders work. Autoencoders build a highly compressed version of an input image in a low-dimensional latent space. They reconstruct the original image from the information available in the latent layer and information built into the network itself. The information stored in the latent space is unique to each image. The information stored in the network can be considered as base information over which the latent space information is added. If the images of interest are simply composites of number of image features, then the image features which are less frequent and occupying less regions get most likely encoded in the latent space and the features which are more frequent and occupying most regions get stored as base information in the network. This is because latent space stores distinct and a very small part of the input information. Since the pearlite microstructures relatively occupy only a smaller area of an image compared to ferrite microstructures, we find the saliency maps give more relevance to pearlite regions and very low or no relevance to ferrite regions. In the subsequent section, the saliency maps generated through the current approach is compared with other existing attribution approaches.

7. Comparison with other popular approaches

Guided backpropagation (GBP) [68], class activation maps (CAMs) [69], layer wise relevance propagation (LRP) [70] and DeepLIFT [71] are widely used interpretability approaches for convolutional neural network models. GBP is a popular gradient approach which relates the relevance of input pixels to the gradients of the outputs. GBP differentiate itself from the vanilla gradient approach by backpropagating only the positive gradients. Hence, saliency maps constructed using GBP are less noisy and more understandable than the vanilla gradient approach. However, GBP simply act as an edge detector and lacks the ability to discriminate between classes. CAMs are specifically developed to overcome this limitation and successfully applied in many classification tasks. However, CAMs require a global max pooling layer before the soft max layer and cannot be readily applied for all the classification networks. LRP is a heuristic approach which sequentially backpropagates the prediction of the outputs based on the contribution of neurons of the previous layer to the activations of the current layer. Though LRP is free of gradient saturation problem suffered by the gradient approaches, it does not have a strong mathematical basis. It also does not handle negative contributions of the input features properly and requires different propagation rules for different levels of the layers. DeepLIFT backpropagates activation differences to find the contribution of the input features and addresses the problem of negative contributions. DeepLIFT has a theoretical basis unlike LRP. One defining aspect of DeepLIFT which is absent in other approaches is that it also considers the contributions from the network biases. The major limitation of DeepLIFT is that it requires domain expertise for fixing the reference basis to compute the activation differences. Our proposed approach is model agnostic can be applied to any type of machine learning task and handle different types of input data. It does not require any domain expertise for the computation of input attributions (relevance). The proposed approach can also handle negative contributions toward the outputs, and it is capable of handling different activation functions.

To assess the effectiveness of our proposed method, saliency maps are also constructed using GBP and LRP and compared with the saliency maps developing using our method (CPA) in Fig. 11. Since GBP can handle only one output, the L_2 norm of latent outputs is used to find the pixel relevance. For LRP, $\mathcal{Z}^+$ propagation rule is used for all the layers except for the first(input) layer. For the first layer, two different propagation rules are used and considered as separate approaches in the comparison. Accordingly, in Fig. 11, $\mathrm{LRP}_{\mathcal{w}^2}$ and LRP_{β} denote layer wise relevance propagation approach with $\mathcal{w}^2$ and $\mathcal{Z}^{\beta}$ rules for the first layer respectively [72]. The saliency maps developed using GBP shows almost all the regions of the input microstructure are significantly relevant which is rather ambiguous since autoencoder must lose dependency on some input features to obtain a lower-dimensional representation. Furthermore, GBP deems ferrite regions are more relevant to compact representation than pearlite regions and grain boundaries which clearly contradicts the results of all the other three approaches. The saliency explanations given by LRP, especially $\mathrm{LRP}_{\mathcal{w}^2}$, is closer to our approach. The saliency explanations given by LRP_{β} is more scattered than $\mathrm{LRP}_{\mathcal{w}^2}$ and less image regions are deemed important compared to $\mathrm{LRP}_{\mathcal{w}^2}$. This clearly demonstrates saliency explanations offered by LRP is sensitive to the propagation rule used for the first layer. Since LRP is heuristic based, and hence no rational basis is available for selecting appropriate propagation rule. This comparison demonstrates our approach produces more consistent and meaningful saliency explanations when compared to GBP and LRP.

8. Outcomes and Conclusions

The current study aims at providing a deep learning framework to identify the important pixel regions in a metal microstructure for compact representation and secure sharing of microstructural image data. The main outcomes and conclusions of this study are

1) The CAE designed in the study achieves a compression ratio of 32 and is capable of reconstructing original microstructural images just from 3.5% of original data without losing essential information.
2) Since the input metal microstructural data can only be reconstructed through the trained decoder network, the compact representation of the metal microstructure along with the decoder network can be used for secure sharing of microstructural data.
3) A model-agnostic sensitivity approach is proposed to quantify the pixel importance and a complex CNN is introduced to carry out the sensitivity analysis.
4) Saliency maps which highlight the relevance/importance of pixel features for image reconstruction (or compact representation) are generated. The saliency maps showed the grey pearlite regions and grain boundaries are highly relevant. Interestingly, ferrite regions are shown unimportant for reconstruction.
5) The proposed complex perturbation approach produces more tenable and consistent saliency explanations compared to guided backpropagation and layer wise propagation methods
6) The generic framework introduce in the study can be extended to identify important microstructural regions for other metals, composites, biomaterials, and material systems.

Acknowledgment

The Research presented in this presentation was supported by the National Science Foundation through NSF EPSCoR Track-1 Cooperative Agreement OIA #1946202 and CAREER award # 2045538. Any opinions, findings, and conclusions, or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the National Science Foundation.

Data Availability Statement

The raw/processed data required to reproduce these findings cannot be shared at this time as the data also forms part of an ongoing study.

References

[1] S. Montecinos, S. Tognana, and W. Salgueiro, "Influence of microstructure on the Young's modulus in a Cu-2Be (wt%) alloy," *Journal of Alloys and Compounds,* vol. 729, pp. 43-48, 2017.

[2] A. Kupke, P. D. Hodgson, and M. Weiss, "The effect of microstructure and pre-strain on the change in apparent Young's modulus of a dual-phase steel," *Journal of Materials Engineering and Performance,* vol. 26, no. 7, pp. 3387-3398, 2017.

[3] W. Kasprzak, H. Kurita, G. Birsan, and B. S. Amirkhiz, "Hardness control of Al–Si HPDC casting alloy via microstructure refinement and tempering parameters," *Materials & Design,* vol. 103, pp. 365-376, 2016.

[4] A. Vornberger, J. Pötschke, T. Gestrich, M. Herrmann, and A. Michaelis, "Influence of microstructure on hardness and thermal conductivity of hardmetals," *International Journal of Refractory Metals and Hard Materials,* vol. 88, p. 105170, 2020.

[5] J. Gubicza, N. Q. Chinh, T. Csanadi, T. Langdon, and T. Ungár, "Microstructure and strength of severely deformed fcc metals," *Materials Science and Engineering: A,* vol. 462, no. 1-2, pp. 86-90, 2007.

[6] C. Zheng, L. Li, W. Yang, and Z. Sun, "Relationship between microstructure and yield strength for plain carbon steel with ultrafine or fine (ferrite+ cementite) structure," *Materials Science and Engineering: A,* vol. 617, pp. 31-38, 2014.

[7] T. Wang, F. Yong, X. H. Liu, K. X. Wang, Y. X. Du, and F. Zhao, "Enhanced Strength-ductility synergy in Ti-4Al-5Mo-5V-5Cr-1Nb with hierarchical microstructure," *Materials Letters: X,* p. 100168, 2022/09/20/ 2022, doi: https://doi.org/10.1016/j.mlblux.2022.100168.

[8] K. O. Pedersen, I. Westermann, T. Furu, T. Børvik, and O. S. Hopperstad, "Influence of microstructure on work-hardening and ductile fracture of aluminium alloys," *Materials & Design,* vol. 70, pp. 31-44, 2015.

[9] H. U. Sajid, D. L. Naik, and R. Kiran, "Microstructure–Mechanical Property Relationships for Post-Fire Structural Steels," *Journal of Materials in Civil Engineering,* vol. 32, no. 6, p. 04020133, 2020.

[10] P. T. Summers, "Microstructure-based constitutive models for residual mechanical behavior of aluminum alloys after fire exposure," Virginia Polytechnic Institute and State University, 2014.

[11] M. Shabani, M. Emamy, and N. Nemati, "Effect of grain refinement on the microstructure and tensile properties of thin 319 Al castings," *Materials & Design,* vol. 32, no. 3, pp. 1542-1547, 2011.

[12] M. Oliaei and R. Jamaati, "Improvement of the strength-ductility-toughness balance in interstitial-free steel by gradient microstructure," *Materials Science and Engineering: A,* vol. 845, p. 143237, 2022.

[13] Z. Zhang *et al.*, "Effects of phase composition and content on the microstructures and mechanical properties of high strength Mg–Y–Zn–Zr alloys," *Materials & design,* vol. 88, pp. 915-923, 2015.

[14] Q. Xie, M. Suvarna, J. Li, X. Zhu, J. Cai, and X. Wang, "Online prediction of mechanical properties of hot rolled steel plate using machine learning," *Materials & Design,* vol. 197, p. 109201, 2021.

[15] G. Deffrennes, K. Terayama, T. Abe, and R. Tamura, "A machine learning–based classification approach for phase diagram prediction," *Materials & Design,* vol. 215, p. 110497, 2022.

[16] A. A. K. Farizhandi and M. Mamivand, "Processing Time, Temperature, and Initial Chemical Composition Prediction from Materials Microstructure by Deep Network for Multiple Inputs and Fused Data," *Materials & Design,* p. 110799, 2022.

[17] D. Stuart *et al.*, "Practical challenges for researchers in data sharing," 2018.

[18] K. Marlapalli, R. S. B. P. Bandlamudi, R. Busi, V. Pranav, and B. Madhavrao, "A Review on Image Compression Techniques," Singapore, 2021: Springer Singapore, in Communication Software and Networks, pp. 271-279.

[19] M. Larmuseau, M. Sluydts, K. Theuwissen, L. Duprez, T. Dhaene, and S. Cottenier, "Compact representations of microstructure images using triplet networks," *npj Computational Materials,* vol. 6, no. 1, pp. 1-11, 2020.

[20] G. Cheng, C. Yang, X. Yao, L. Guo, and J. Han, "When deep learning meets metric learning: Remote sensing image scene classification via learning discriminative CNNs," *IEEE transactions on geoscience and remote sensing,* vol. 56, no. 5, pp. 2811-2821, 2018.

[21] D. Bank, N. Koenigstein, and R. Giryes, "Autoencoders," *arXiv preprint arXiv:2003.05991,* 2020.

[22] W. H. L. Pinaya, S. Vieira, R. Garcia-Dias, and A. Mechelli, "Autoencoders," in *Machine learning*: Elsevier, 2020, pp. 193-208.

[23] W. Wang, Y. Huang, Y. Wang, and L. Wang, "Generalized autoencoder: A neural network framework for dimensionality reduction," in *Proceedings of the IEEE conference on computer vision and pattern recognition workshops*, 2014, pp. 490-497.

[24] E. Lin, S. Mukherjee, and S. Kannan, "A deep adversarial variational autoencoder model for dimensionality reduction in single-cell RNA sequencing analysis," *BMC bioinformatics,* vol. 21, no. 1, pp. 1-11, 2020.

[25] C. Zhou and R. C. Paffenroth, "Anomaly detection with robust deep autoencoders," in *Proceedings of the 23rd ACM SIGKDD international conference on knowledge discovery and data mining*, 2017, pp. 665-674.

[26] J. Feng, Y. Liang, and L. Li, "Anomaly Detection in Videos Using Two-Stream Autoencoder with Post Hoc Interpretability," *Computational Intelligence and Neuroscience,* vol. 2021, 2021.

[27] S. Rifai, P. Vincent, X. Muller, X. Glorot, and Y. Bengio, "Contractive auto-encoders: Explicit invariance during feature extraction," in *Icml*, 2011.

[28] J. Zabalza *et al.*, "Novel segmented stacked autoencoder for effective dimensionality reduction and feature extraction in hyperspectral imaging," *Neurocomputing,* vol. 185, pp. 1-10, 2016.

[29] C. Nash and C. K. Williams, "The shape variational autoencoder: A deep generative model of part-segmented 3D objects," in *Computer Graphics Forum*, 2017, vol. 36, no. 5: Wiley Online Library, pp. 1-12.

[30] Y. Wu, Y. Burda, R. Salakhutdinov, and R. Grosse, "On the quantitative analysis of decoder-based generative models," *arXiv preprint arXiv:1611.04273,* 2016.

[31] A. Ng, "Sparse autoencoder," *CS294A Lecture notes,* vol. 72, no. 2011, pp. 1-19, 2011.

[32] K. Cho, "Boltzmann machines and denoising autoencoders for image denoising," *arXiv preprint arXiv:1301.3468,* 2013.

[33] D. P. Kingma and M. Welling, "An introduction to variational autoencoders," *arXiv preprint arXiv:1906.02691,* 2019.

[34] Y. Zhang, "A better autoencoder for image: Convolutional autoencoder," in *ICONIP17-DCEC. Available online: http://users. cecs. anu. edu. au/Tom. Gedeon/conf/ABCs2018/paper/ABCs2018_paper_58. pdf (accessed on 23 March 2017)*, 2018.

[35] Y. Kim *et al.*, "Exploration of optimal microstructure and mechanical properties in continuous microstructure space using a variational autoencoder," *Materials & Design,* vol. 202, p. 109544, 2021.

[36] S. M. Lee, S.-Y. Park, and B.-H. Choi, "Application of domain-adaptive convolutional variational autoencoder for stress-state prediction," *Knowledge-Based Systems,* vol. 248, p. 108827, 2022.
[37] J. Xu and K. Duraisamy, "Multi-level convolutional autoencoder networks for parametric prediction of spatio-temporal dynamics," *Computer Methods in Applied Mechanics and Engineering,* vol. 372, p. 113379, 2020.
[38] K. Fukami, T. Nakamura, and K. Fukagata, "Convolutional neural network based hierarchical autoencoder for nonlinear mode decomposition of fluid field data," *Physics of Fluids,* vol. 32, no. 9, p. 095110, 2020.
[39] D. Jana, J. Patil, S. Herkal, S. Nagarajaiah, and L. Duenas-Osorio, "CNN and Convolutional Autoencoder (CAE) based real-time sensor fault detection, localization, and correction," *Mechanical Systems and Signal Processing,* vol. 169, p. 108723, 2022.
[40] Z. Nie, H. Jiang, and L. B. Kara, "Stress field prediction in cantilevered structures using convolutional neural networks," *Journal of Computing and Information Science in Engineering,* vol. 20, no. 1, p. 011002, 2020.
[41] Z. Rastin, G. Ghodrati Amiri, and E. Darvishan, "Unsupervised structural damage detection technique based on a deep convolutional autoencoder," *Shock and Vibration,* vol. 2021, 2021.
[42] F. Ni, J. Zhang, and M. N. Noori, "Deep learning for data anomaly detection and data compression of a long-span suspension bridge," *Computer-Aided Civil and Infrastructure Engineering,* vol. 35, no. 7, pp. 685-700, 2020.
[43] H. Steck, "Autoencoders that don't overfit towards the Identity," *Advances in Neural Information Processing Systems,* vol. 33, pp. 19598-19608, 2020.
[44] H. Gholamalinezhad and H. Khosravi, "Pooling methods in deep neural networks, a review," *arXiv preprint arXiv:2009.07485,* 2020.
[45] V. Turchenko, E. Chalmers, and A. Luczak, "A deep convolutional auto-encoder with pooling-unpooling layers in caffe," *arXiv preprint arXiv:1701.04949,* 2017.
[46] B. Hou and R. Yan, "Convolutional autoencoder model for finger-vein verification," *IEEE Transactions on Instrumentation and Measurement,* vol. 69, no. 5, pp. 2067-2074, 2019.
[47] N. M. N. Leite, E. T. Pereira, E. C. Gurjao, and L. R. Veloso, "Deep convolutional autoencoder for EEG noise filtering," in *2018 IEEE International Conference on Bioinformatics and Biomedicine (BIBM)*, 2018: IEEE, pp. 2605-2612.
[48] A. Odena, V. Dumoulin, and C. Olah, "Deconvolution and checkerboard artifacts," *Distill,* vol. 1, no. 10, p. e3, 2016.
[49] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," *arXiv preprint arXiv:1409.1556,* 2014.
[50] C. Szegedy, V. Vanhoucke, S. Ioffe, J. Shlens, and Z. Wojna, "Rethinking the inception architecture for computer vision," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2016, pp. 2818-2826.
[51] E. Million, "The hadamard product," *Course Notes,* vol. 3, no. 6, 2007.
[52] I. Goodfellow, Y. Bengio, and A. Courville, "Convolutional networks," in *Deep learning*, vol. 2016: MIT Press Cambridge, MA, USA, 2016, pp. 330-372.
[53] H. U. Sajid and R. Kiran, "Influence of high stress triaxiality on mechanical strength of ASTM A36, ASTM A572 and ASTM A992 steels," *Construction and Building Materials,* vol. 176, pp. 129-134, 2018.

[54] D. L. Naik, H. U. Sajid, and R. Kiran, "Texture-Based Metallurgical Phase Identification in Structural Steels: A Supervised Machine Learning Approach," *Metals,* vol. 9, no. 5, p. 546, 2019.

[55] D. Arumugam, D. L. Naik, H. U. Sajid, and R. Kiran, "Relationship between Nano and Macroscale Properties of Postfire ASTM A36 Steels," *Journal of Materials in Civil Engineering,* vol. 34, no. 6, p. 04022100, 2022.

[56] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," *arXiv preprint arXiv:1412.6980,* 2014.

[57] M. Abadi *et al.*, "Tensorflow: Large-scale machine learning on heterogeneous distributed systems," *arXiv preprint arXiv:1603.04467,* 2016.

[58] M. Lin, Q. Chen, and S. Yan, "Network in network," *arXiv preprint arXiv:1312.4400,* 2013.

[59] R. Kiran, L. Li, and K. Khandelwal, "Complex perturbation method for sensitivity analysis of nonlinear trusses," *Journal of Structural Engineering,* vol. 143, no. 1, p. 04016154, 2017.

[60] R. Kiran and K. Khandelwal, "Complex step derivative approximation for numerical evaluation of tangent moduli," *Computers & Structures,* vol. 140, pp. 1-13, 2014.

[61] R. Kiran and D. L. Naik, "Novel sensitivity method for evaluating the first derivative of the feed-forward neural network outputs," *Journal of Big Data,* vol. 8, no. 1, pp. 1-13, 2021.

[62] D. L. Naik and R. kiran, "A novel sensitivity-based method for feature selection," *Journal of Big Data,* vol. 8, no. 1, p. 128, 2021/10/09 2021, doi: 10.1186/s40537-021-00515-w.

[63] D. Wilke and S. Kok, "Numerical sensitivity computation for discontinuous gradient-only optimization problems using the complex-step method," 2012.

[64] K.-L. Lai and J. Crassidis, "Extensions of the first and second complex-step derivative approximations," *Journal of Computational and Applied Mathematics,* vol. 219, no. 1, pp. 276-293, 2008.

[65] K. Simonyan, A. Vedaldi, and A. Zisserman, "Deep inside convolutional networks: Visualising image classification models and saliency maps," *arXiv preprint arXiv:1312.6034,* 2013.

[66] L. Itti, C. Koch, and E. Niebur, "A model of saliency-based visual attention for rapid scene analysis," *IEEE Transactions on pattern analysis and machine intelligence,* vol. 20, no. 11, pp. 1254-1259, 1998.

[67] J. Han, D. Zhang, G. Cheng, N. Liu, and D. Xu, "Advanced deep-learning techniques for salient and category-specific object detection: a survey," *IEEE Signal Processing Magazine,* vol. 35, no. 1, pp. 84-100, 2018.

[68] J. T. Springenberg, A. Dosovitskiy, T. Brox, and M. Riedmiller, "Striving for simplicity: The all convolutional net," *arXiv preprint arXiv:1412.6806,* 2014.

[69] B. Zhou, A. Khosla, A. Lapedriza, A. Oliva, and A. Torralba, "Learning deep features for discriminative localization," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2016, pp. 2921-2929.

[70] S. Bach, A. Binder, G. Montavon, F. Klauschen, K.-R. Müller, and W. Samek, "On pixel-wise explanations for non-linear classifier decisions by layer-wise relevance propagation," *PloS one,* vol. 10, no. 7, p. e0130140, 2015.

[71] A. Shrikumar, P. Greenside, and A. Kundaje, "Learning important features through propagating activation differences," in *International Conference on Machine Learning*, 2017: PMLR, pp. 3145-3153.

[72] G. Montavon, A. Binder, S. Lapuschkin, W. Samek, and K.-R. Müller, "Layer-wise relevance propagation: an overview," *Explainable AI: interpreting, explaining and visualizing deep learning,* pp. 193-209, 2019.

Tables

Table 1: Regularization terms of different autoencoders

Autoencoders	**Regularization terms**
Sparse autoencoders	$\lambda \sum_i \lvert a_i \rvert$
Contractive autoencoders	$\lambda \lVert \boldsymbol{J}(\boldsymbol{x}) \rVert_F^2,\ J(\boldsymbol{x}) = \sum_{ij} \left(\partial g_j / \partial x_i\right)^2$
Variational autoencoders	$\lambda \sum_j KL(q_j(l \mid \boldsymbol{x}) \mid\mid p(l))$
Denoising autoencoders	-

where, $\boldsymbol{x}$ is an input, $\boldsymbol{l}$ is a latent output, λ is the regularization parameter, a_i is i^{th} activation of the latent layer, $\boldsymbol{J}$ is the Jacobian of the partial derivatives, l_j is the j^{th} latent variable, x_i is the i^{th} input feature. KL is the divergence function between the distribution of the code layer ($q_j(\boldsymbol{l}|\boldsymbol{x})$) and true prior distribution ($p(\boldsymbol{l})$) (approximated using a unit gaussian distribution).

Table 2: Functional expressions of different activation functions

Activation Type	**Activation function**
Linear	$\phi(z) = z$
ReLU	$\phi(z) = \begin{cases} 0 & if\ z \leq 0 \\ z & if\ z > 0 \end{cases}$
Leaky ReLU	$\phi(z) = \begin{cases} \beta z & if\ z \leq 0 \\ z & if\ z > 0 \end{cases}$
TanH	$\phi(z) = \tanh(z)$
Sigmoid	$\phi(z) = 1/(1 + e^{-z})$

Table 3: Parameter used in this study for ADAM optimizer

Parameter	β_1	β_2	ϵ	α
Value	0.9	0.999	10^{-8}	0.001

Table 4: Architectural details of encoder block of convolutional autoencoder

Layer	**No of filters**	**Kernel/Pool size**	**Activation function**	**Data Size**
Input	-	-	-	256×256×1
Convolutional Layer 1	64	3×3	ReLU	256×256×64
Maxpooling Layer 1	-	2×2	-	128×128×64
Convolutional Layer 2	32	3×3	ReLU	128×128×32
Maxpooling Layer 2	-	2×2	-	64×64×32
Convolutional Layer 3	16	3×3	ReLU	64×64×16
Maxpooling Layer 3	-	2×2	-	32×32×16
Convolutional Layer 4	8	3×3	ReLU	32×32×8
Maxpooling Layer 4	-	2×2	-	16×16×8

Table 5: Architectural details of decoder block of convolutional autoencoder

Layer	**No of filters**	**Kernel/Pool size**	**Activation function**	**Data Size**
Latent Input	-	-	-	16×16×8
Upsampling Layer 1	-	2×2	-	32×32×8
Convolutional Layer 1	8	3×3	ReLU	32×32×8
Upsampling Layer 2	-	2×2	-	64×64×8
Convolutional Layer 2	16	3×3	ReLU	64×64×16
Upsampling Layer 3	-	2×2	-	128×128×16
Convolutional Layer 3	32	3×3	ReLU	128×128×32
Upsampling Layer 4	-	2×2	-	256×256×32
Convolutional Layer 4	64	3×3	ReLU	256×256×64
Convolutional Layer 5	1	3×3	Linear	256×256×1

Table 6: Architectural details of decoder block of convolutional autoencoder

Figures

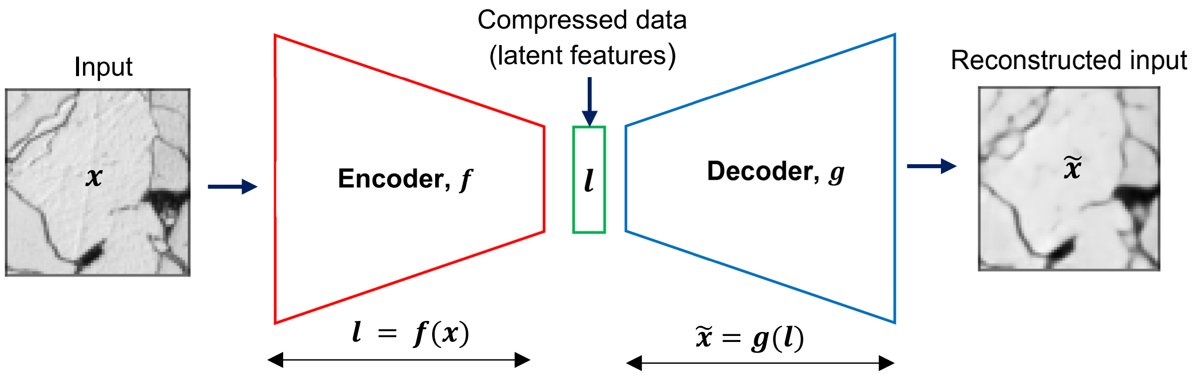


Fig. 1: General schematic of autoencoder's architecture. Encoder, f, maps the input, $\boldsymbol{x}$, to a dimensionally reduced space and generate latent features, $\boldsymbol{l}$, and decoder, g, uses the compressed data, $\boldsymbol{l}$, and outputs reconstructed input, $\boldsymbol{x}$, without significant loss of information.

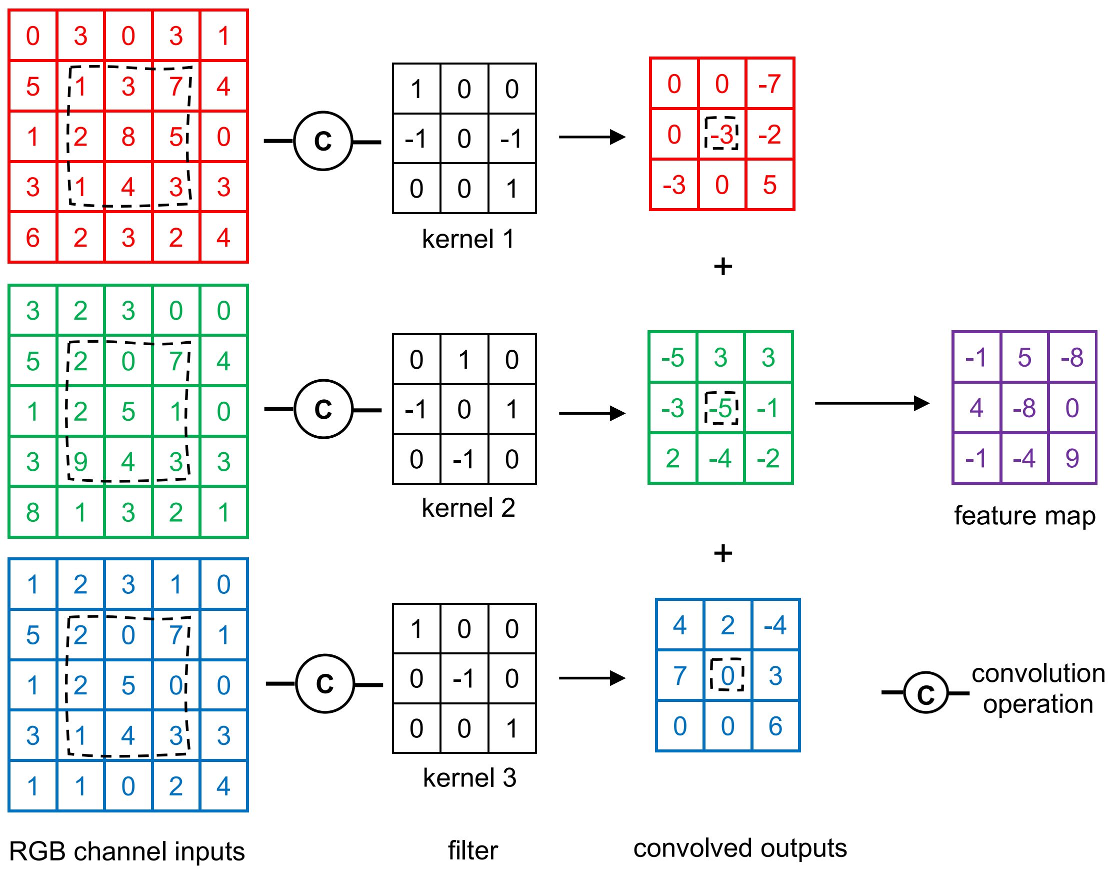


Fig. 2: Convolution of an RGB image input with 3×3 kernels and a unit stride. The three input

channel arrays are convolved with their respective kernels positioned next to them. Dashed boxes on the input shows subarrays enclosed by the kernel window at an arbitrary step. The outputs and their positions results from convolution between the subarrays and the kernels are again indicated using a dashed box on the output arrays. The feature map is obtained finally by summing the convolved outputs together.

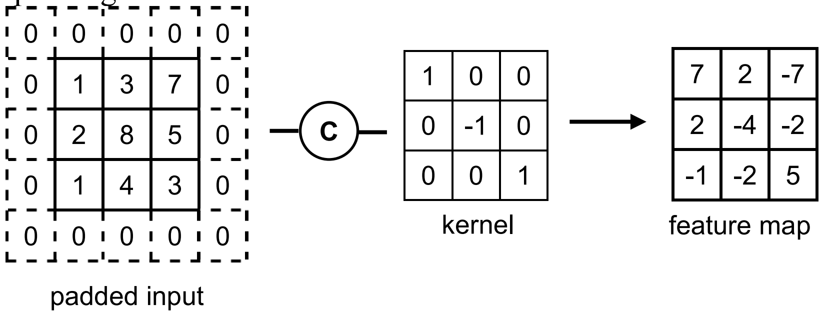


Fig. 3: Convolutional operation involved in 'same' padding type. A row and column of zeros are added around the input sides to produce the feature map with size same as that of the input. A kernel filter of size 3×3 and a unit stride are used for this demonstration.

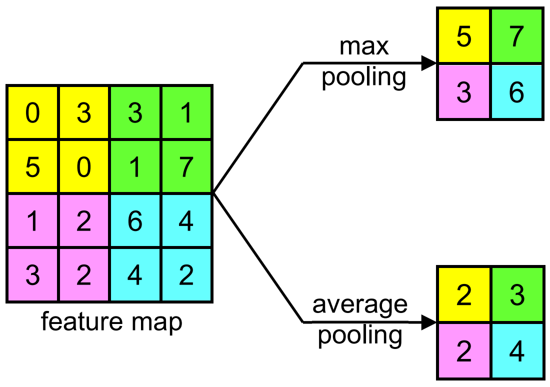


Fig. 4: Illustration of a pooling operation with a pool window size of 2×2 and stride 2. The different colors in the feature map highlight the subarrays captured by the pool window. The max pool and average pool values of those subarrays are shown on the right side. The pooling operation halves the width and height of input feature maps.

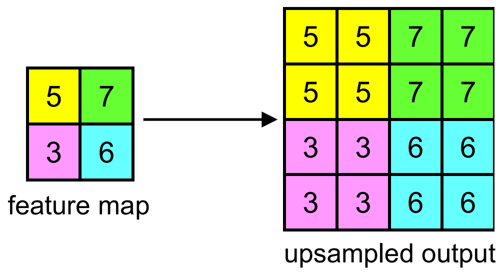


Fig. 5: Illustration of a upsampling operation with a window size of 2×2. Upsampling in this illustrated case involves the repetition of each input values into a 2×2 array and increases the dimensions of the input by the factor of 2.

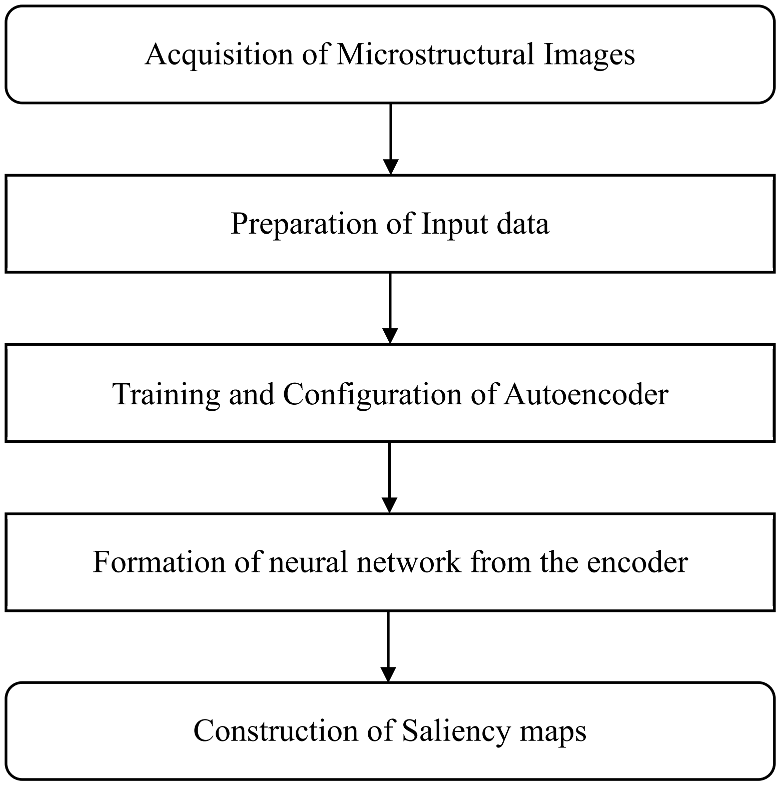

Acquisition of Microstructural Images
Preparation of Input data
Training and Configuration of Autoencoder
Formation of neural network from the encoder
Construction of Saliency maps

Fig. 6: Schematic of the methodology used in this study

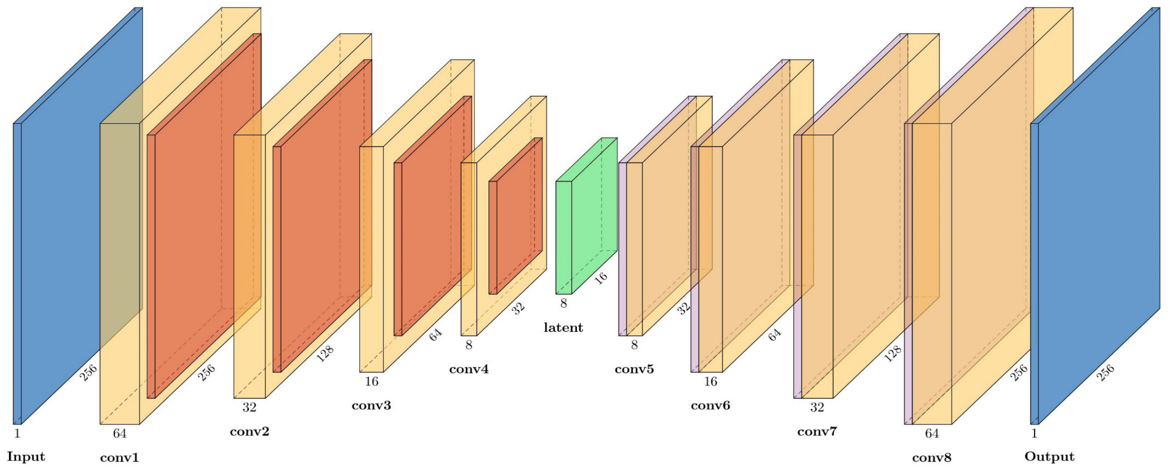
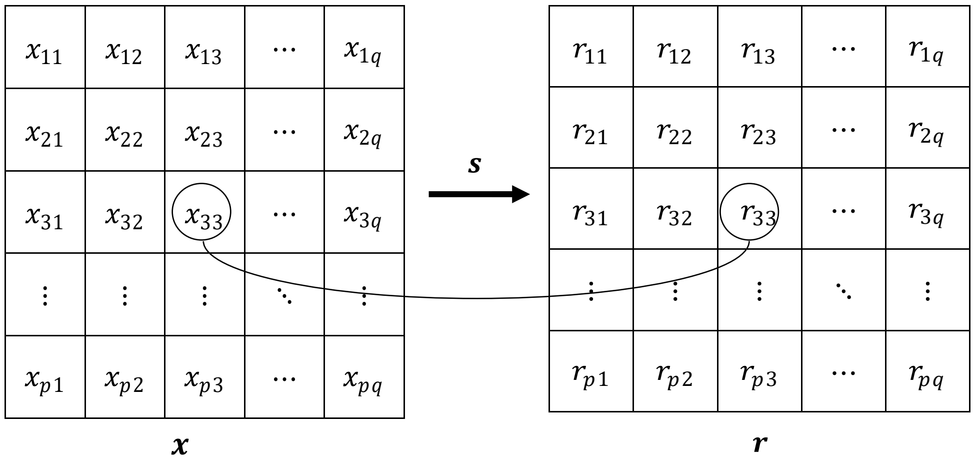
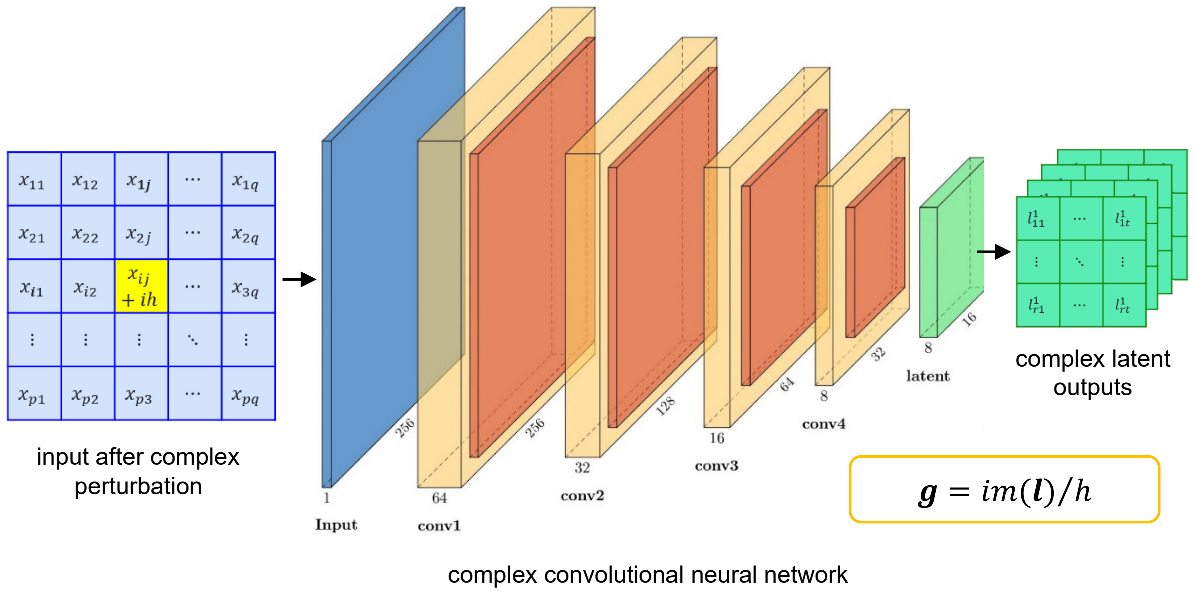


Fig. 7: Architecture of the convolutional autoencoder employed in the study. Yellow, red, and purple blocks are convolutional, max pooling and upsampling layers respectively. The employed network involves 50,353 network parameters. The input size of 256×256×1 is reduced to latent size of 16×16×8 with a compression ratio of 32.



x_{11} x_{12} x_{13} ⋯ x_{1q}
x_{21} x_{22} x_{23} ⋯ x_{2q}
x_{31} x_{32} x_{33} ⋯ x_{3q}
x_{p1} x_{p2} x_{p3} ⋯ x_{pq}
$\boldsymbol{x}$
$\boldsymbol{s}$
r_{11} r_{12} r_{13} ⋯ r_{1q}
r_{21} r_{22} r_{23} ⋯ r_{2q}
r_{31} r_{32} r_{33} ⋯ r_{3q}
r_{p1} r_{p2} r_{p3} ⋯ r_{pq}
$\boldsymbol{r}$

Fig. 8: Relevance matrix mapping the attribution of each pixel features in the input array



x_{11} x_{12} x_{1j} ⋯ x_{1q}
x_{21} x_{22} x_{2j} ⋯ x_{2q}
x_{i1} x_{i2} $x_{ij} + ih$ ⋯ x_{3q}
x_{p1} x_{p2} x_{p3} ⋯ x_{pq}
input after complex perturbation
Input
conv1
conv2
conv3
conv4
latent
complex latent outputs
l_{11}^1 l_{1t}^1 l_{r1}^1 l_{rt}^1
$\boldsymbol{g} = im(\boldsymbol{l})/h$
complex convolutional neural network

Fig. 9: Implementation of complex step derivative approximation to compute partial derivatives of latent features with respect to input pixel features. The input with a perturbed pixel feature (x_{ij}) is feedforwarded on the complex convolutional neural network. The partial derivatives ($\boldsymbol{g}$) are determined by dividing the imaginary components of complex latent outputs ($\boldsymbol{l}$) with step size, h.

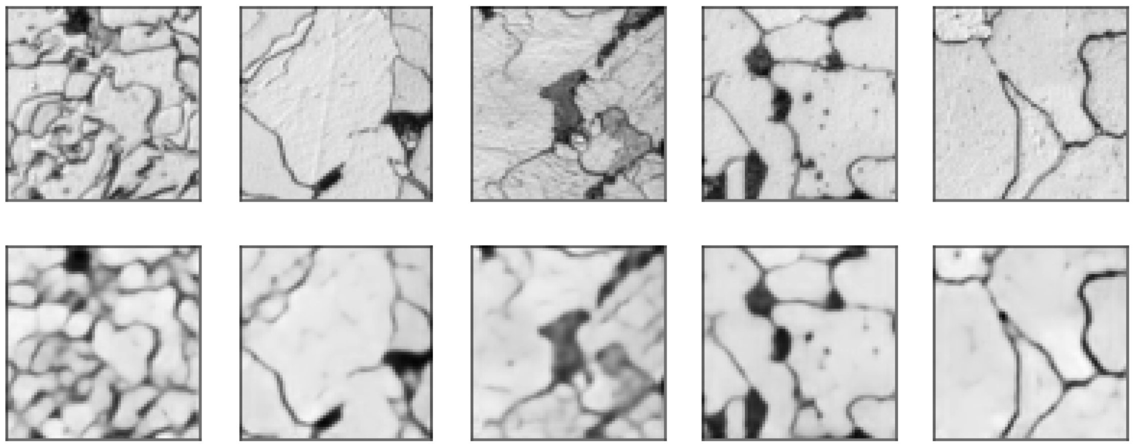
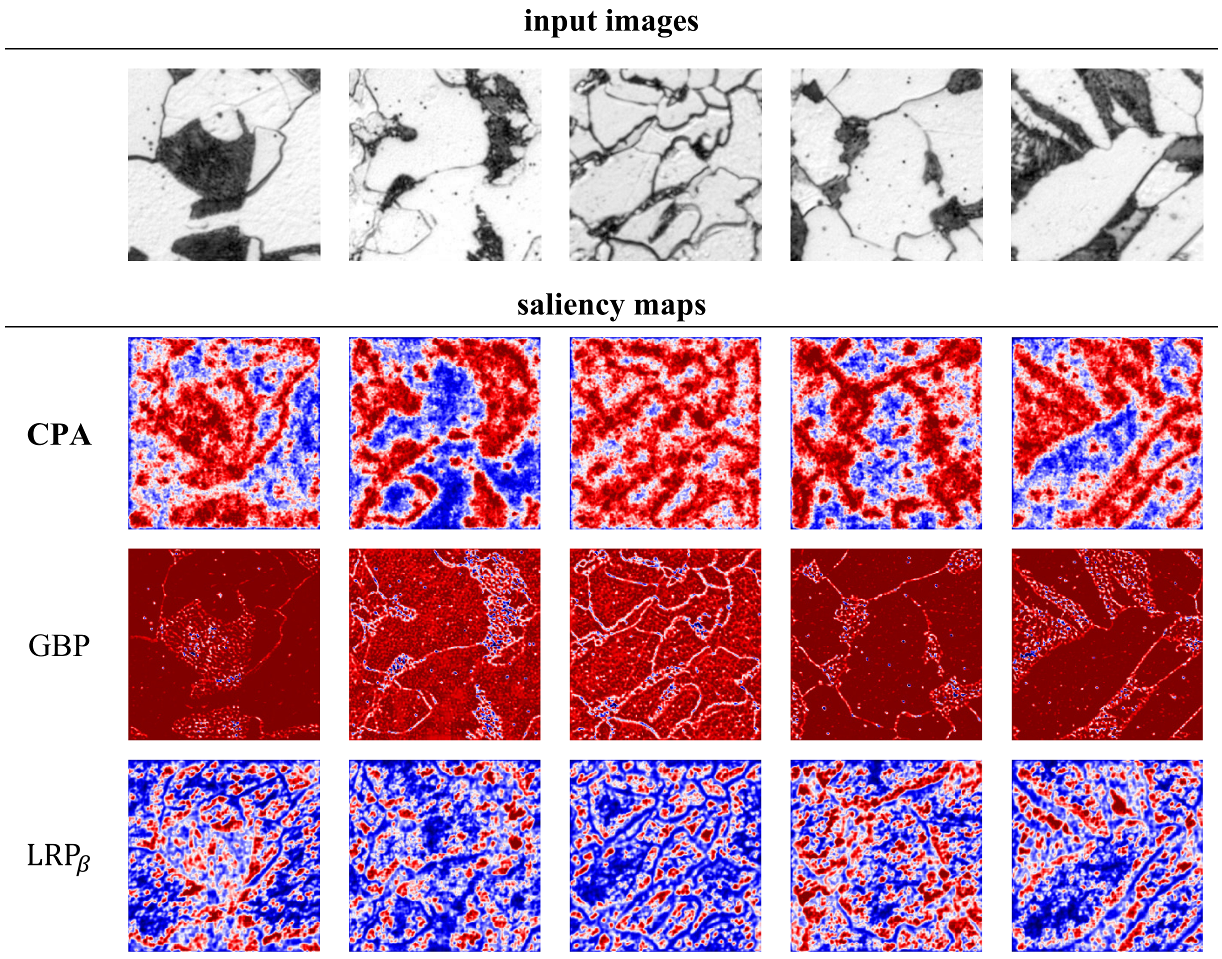


Fig. 10: Top row – Input microstructural images, Bottom row - microstructural images reconstructed by the trained network with a mean squared error reconstruction loss 0.035.



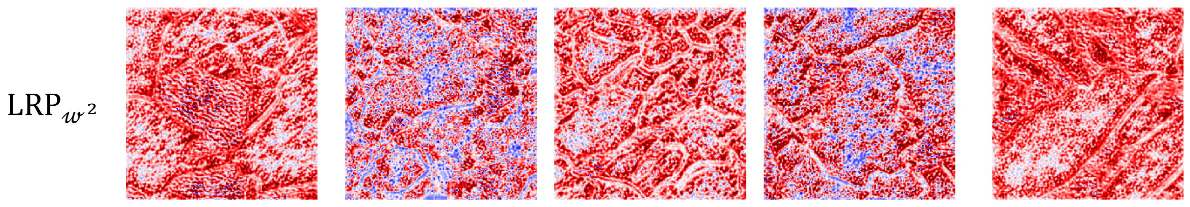


Fig. 11: Saliency maps generated for the trained convolutional autoencoder using the proposed complex perturbation approach (CPA), guided backpropagation (GBP), layer wise propagation with w^2 rule for the first layer (LRP_{w^2}), layer wise propagation with $\mathcal{Z}^{\beta}$ rule for the first layer (LRP_{β}), and along with the input microstructural images.