Interpreting Denoising Autoencoders Employing a Gradient Approach

Dharanidharan Arumugam[1] and Ravi Kiran[2]

Abstract

The goal of this study is to interpret denoising autoencoders by quantifying the importance of input pixel features for image reconstruction. The importance of pixel features is evaluated using the attributions of the pixel features to the latent variables of a denoising autoencoder used for image reconstruction. Pixel attributions are computed using a highly accurate and automatable gradient approach and are plotted as saliency maps. Saliency maps highlight the contribution of the pixels for image reconstruction. Three sanity checks are introduced to verify the fidelity of the generated saliency maps and also to elucidate the influence of inputs on the latent variables. The classification accuracy of images is significantly lowered when the most important pixel regions highlighted by the saliency maps are corrupted validating the proposed approach.

Keywords: Complex step derivative approximation; Saliency maps; Trustworthiness; Pixel attributions; Sanity checks and Deep Neural Networks (DNNs).

1. Introduction

Deep neural networks (DNNs), for their remarkable predictive performance, are now extensively used in various decision-making processes involving health care, law enforcement, finance, and engineering analysis. As the decisions taken by DNNs carry significant social,

[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

political, and economic implications, DNNs are now expected to be fair, transparent, and trustworthy. The concerns related to the embedded biases in the neural networks limit their use in high-stake applications such as medical diagnosis, the autonomous bail system, and autonomous driving [1-4]. In a survey conducted in 2021 across five countries including the United States, Canada, Germany, the United Kingdom, and Australia, only one-third of 6054 participants are willing to trust decisions taken by AI in general and 81% of people believe that the use of AI should be regulated [5]. Since DNNs learn the functional mapping through complex non-linear interaction of numerous network parameters, they remain as black-box models and offer little explanation on how the decisions or predictions are made. To make the DNNs trustworthy, the decisions made by them need to be made more interpretable to humans. Interpretability of DNNs is also desired to establish the causal links and also to identify the important features in a model. Several methods have been developed to enhance the interpretability of DNNs with two broader approaches: construction of interpretable models or white box models and post-hoc interpretation of black-box models.

White-box models use transparent, human-understandable models like linear models, decision trees, and elementary rules-based models (addition, subtraction, etc.) to construct interpretable representations. Generalized linear models (GLMs) [6, 7] allow us to map non-linearity between the input and predictor variables using linear models with the help of link functions. Link functions connect the linear predictor with the actual predictor. Wei et al [8] proposed generalized linear models based on rules-based features for interpretable regression and classification networks. In addition to offering interpretability, rules-based features enable the model to capture non-linearity. Generalized additive models (GAMs) [9, 10] involve linear addition of non-linear smooth functions of predictors to model the non-linearities. Supersparse

linear integer models (SLIM) [11] are made highly interpretable by only using basic elementary operations on the predictors to make predictions. Boolean rules are also used to construct interpretable network models [12]. Construction of white-box models offers more interpretability than the post-hoc approaches, however, the prediction performance is compromised for the sake of interpretability. Also, most of the white-box models are mainly applied to interpret tabular data.

Post-hoc approaches involve the interpretation of already deployed or trained models. Since they are applied to the trained models, the prediction performance of the network is not compromised. Post-hoc approaches are in general attribution-based. Attribution methods find the relevance or importance (saliency) of input features in the prediction of outputs. Attributions can be made on various types of input features such as tabular data, image features, and image pixels. Local interpretable model-agnostic explanations (LIME) and Shapely Additive Explanations (SHAP) are two popular feature attribution methods. They are also model-agnostic approaches that can invariably be applied to any black-box models. LIME [13, 14] perturbs an instance and generates new instances around the original instance, and uses those instances on simple interpretable models such as linear models and decision trees to obtain interpretable representation. SHAP [15] which is inspired by game theory estimates the contributions of both individual and combined effects of features on a particular prediction. Deep neural networks used for image recognition methods are majorly explained through feature visualization and pixel attribution. Feature visualization helps to visualize how a neuron or a layer or a feature map in convolutional neural networks detect image features such as texture, edges, and patterns. Attribution measures the contribution of input features to the output predictions. Attributions, in general, are quantified through the gradients of outputs with respect to the input features. Quantifying the attributions using gradients is justified by the assumption that the model behaves linearly in the local

neighborhood of the inputs. Pixel attribution highlights the importance or saliency of input pixels in regard to classification or image reconstruction. Pixel attributions are usually presented in the size of the input image in the form of heat or contour maps. These maps are popularly known as saliency maps [16]. Zeiler et al [17] proposed a Deconvolutional network (Deconvnet) which reverses the process of a convolutional network to identify the pixel regions important for classification. In Deconvnet, the gradient attributions of neurons are backpropagated even when the activations of those neurons are zero during the feedforward step. Simonyan et al [18] developed image-specific saliency maps based solely on the gradient values of output neurons with respect to the input pixel features for image classification models. In this approach, unlike Deconvnet, the gradient attributions are zeroed out based on the feedforward activations values. In Guided backpropagation [19] both these approaches are combined in evaluating the gradient attributions. Deconvnet and Guided backpropagation do not capture the negative influence of inputs on the outputs since the negative gradients are filtered out. All the three approaches discussed above suffer from the gradient saturation problem where the inputs near the flat gradient regions are incorrectly given with low attributions. Integrated gradients [20] approach addresses this problem by computing the pixel attribution using an average gradient. The pixel attributions are defined as the element-wise product of original input pixel values and the average gradient values. The average gradient is calculated by integrating the gradients over the scaled inputs varied from a baseline value to the original value. The baseline value, in general, is taken at zero. Class Activation Maps (CAMs) constructed from global averaging pooling layers of convolutional neural networks were able to identify the discriminative image regions critical for the classification of the images [21]. Grad-CAM [22] combined the ability of gradient-based techniques to identify salient pixels and the class discriminative ability of CAMs. DeepLIFT proposed by Shrikumar et

al [23] addressed the zeroing of negative gradients in back-propagation-based gradient methods. All these post-hoc approaches are model-specific, i.e., can only be applied to a specific type of DNN.

The majority of the interpretable AI research is focused on the interpretation of neural networks used for regression tasks and convolutional neural networks for image recognition. Not much research is available on the interpretation of autoencoders especially on finding input attributions to the latent space of denoising autoencoders. Autoencoders is a class of unsupervised neural networks popularly used for image reconstruction, image denoising, feature extraction, and dimensionality reduction [24, 25]. Some of the important applications of autoencoders include clustering [26, 27], dimensionality reduction [28, 29], anomaly detection [30, 31], generative modelling [32, 33], and preferences prediction [34, 35]. Understanding the contribution of inputs to the latent space of autoencoders is of great research interest for the following reasons: Autoencoders learn manifold structures of inputs in the lower dimensional latent space [36]. Also, the latent space features in disentangled variational autoencoders act as the generative factors of the input feature space [37]. Furthermore, in certain cases, the latent space of autoencoders used for image reconstruction contains important insights on the decisions or predictions made by neural networks. Most of the existing research on the interpretation of the latent variables involves the modification of either encoder or decoder network simpler explainable networks specific to the problem of interest. Curi. et al [38] restricted certain connections from latent variables nodes to the output nodes to make the latent variables represent the latent traits in cognitive modelling and performed relational analysis between the items of assessment (inputs) and the latent space (latent variables). Sergei et al [39] used an additional linear encoding-decoding structure for the priorly annotated factors for single-cell RNA-seq data to learn the latent representation using prior

knowledge. In Ref.[40], the inputs important for energy demand is analysed using two separate latent spaces: one for the main power information and another for auxiliary power information of the energy demand. Energy demand is predicted from the variables from the two latent spaces and the inputs majorly influencing the energy demand is explained by identifying the inputs significantly influencing the auxiliary latent space. All the three approaches we discussed above are model-specific. Rami et al [41] proposed logic-driven autoencoders which is built with fuzzy logic operators capturing the underlying logical relationships of the model. Although the logic-driven autoencoders improve the transparency of network architecture, it is difficult to identify and implement the under lying logic relationship of the problem.

In this study, we introduce a model agnostic highly accurate gradient-based attribution method to interpret the relevance of input pixels to the latent space of denoising autoencoders for image reconstruction. This gradient approach produces saliency maps outlining the importance (attribution) of each pixel of an image. We have also introduced three sanity checks to verify the fidelity of the generated saliency maps and elucidate the influence of inputs on the dimensionally reduced latent variables. The rest of the manuscript is organized as follows: Section 2 explains the overall research approach adopted in this study, Section 3 talks about the different types of autoencoders, section 4 discusses the complex step derivative approximation method used in the study to calculate network gradients, Section 5 details the implementation of the proposed gradient approach, Section 6 discusses the obtained results and Section 7 provides the conclusions obtained from this study.

2. Research Approach

Post-hoc interpretation of a deep neural network begins after the neural network is trained and configured. So, a denoising autoencoder is configured and trained for the adequate

reconstruction of the input images. Configuration of the autoencoder involves the selection of an appropriate number of hidden layers and the number of neurons in each hidden layer and the training involves the determination of network parameter values (weights and biases). The interpretation of input space for image reconstruction is described through the attributions of input pixels to the latent variables. Attribution of an image pixel is defined as the L_2 norm of the partial derivatives of latent variables with respect to that input pixel. The partial derivatives of the latent variables are calculated using a highly accurate and automatable gradient approach that is discussed in Section 4. The attribution of each pixel of an image is computed by considering one pixel at a time. Ultimately, saliency maps in the form of contours are constructed based on the pixel attribution values. The saliency map of an image input gives the overall picture of the contributions of the input pixels to the latent variables for its reconstruction. In the end, three sanity checks are performed to verify the fidelity of the generated saliency maps. The flowchart outlining the research approach employed in the current study is shown in Fig. 1.

3. Basic principles of important autoencoder architectures

The function of the autoencoder is to reconstruct the original input from a dimensionally reduced latent space and thereby learn the important features in the data. This is done by joining two contrasting neural nets, namely, an encoder and a decoder. The encoder of the autoencoder represents the input ($\boldsymbol{x}$) into a dimensionally reduced latent representation ($\boldsymbol{g}$) called code and the decoder expands the code ($\boldsymbol{g}$) and tries to reconstruct the original input ($\hat{\boldsymbol{x}}$) [24, 25] without significant loss of information. The schematic of a simple autoencoder is shown in Fig. 2. The

performance of the autoencoder is evaluated through the estimation of reconstruction loss or a loss function which computes the difference between the reconstructed input and the original input ($\mathcal{L}(\boldsymbol{x}, \hat{\boldsymbol{x}}) = f(\hat{\boldsymbol{x}} - \boldsymbol{x})$). In general, L_2 norm of the differences (between the original and reconstructed input) or binary cross-entropy averaged over training instances are used to compute the reconstruction loss.

Deep autoencoders are prone to overfitting and may learn the trivial identity function. The problem of overfitting is addressed through the regularization of the network. Different approaches are evolved to regularize the network which gives rise to four important variants of autoencoders which are sparse autoencoders, contractive autoencoders, variational autoencoders, and denoising autoencoders. In sparse autoencoders [42], the activations of selective nodes in the hidden layers are restricted through a regularizer term which penalizes the weights and biases of the selective hidden nodes with the higher activation values. The penalty or regularizer term can be based on L_1 norm of activations or Kullback-Leibler (KL) divergence term. Sparse autoencoders are previously used in the detection of heart diseases, recognition of sign language, and identification of structural damage [43-46]. Contractive autoencoders [47] employ a regularizer that penalizes the Frobenius norm of the Jacobians (matrix consisting of fist order derivatives) of hidden layer activations with respect to the inputs. Frobenius norm is a root squared sum of the derivatives in the Jacobian matrix. This makes the encoded network insensitive to the perturbations in the inputs which effectively contracts the input region. Variational autoencoders [48] focusses on recreating the distribution of the data rather than simply reconstructing the inputs. The code activations are modeled with a probability distribution (prior probability distribution), usually a unit gaussian distribution. The decoder then uses a reparametrized sample from the prior distribution and tries to reconstruct the input with the same prior probability distribution. Variational autoencoders have

been successfully applied in molecular design, image segmentation, biological sequence analysis, and novelty detection [31, 32, 49-53]. The cost functions associated with these autoencoders including denoising autoencoders are presented in Table. 1.

Denoising autoencoders [54] have many important real-world applications such as denoising images, medical videos, cluttered radar images, RNA sequences, and seismic data [54-58]. NASA employs stacked denoising autoencoders to perform bias correction of satellite used for precipitation detection and prediction of precipitation rates [59]. Unlike the other variants discussed earlier, denoising autoencoders do not use any explicit regularizer term. This saves a lot of computational effort when the network parameters are optimized through backpropagation. In denoising autoencoders, the original inputs (cleaner inputs) are added with noise and fed into the encoder network. The encoder network transforms the noisy input ($\tilde{\boldsymbol{x}}$) into latent variables ($\boldsymbol{g}$). The decoder then attempts to reconstruct the cleaner (without any noise) input from the latent variables ($\hat{\boldsymbol{x}}$). Here, the loss function computes the difference between the reconstructed input and the cleaner input ($\mathcal{L}(\boldsymbol{x}, \hat{\boldsymbol{x}}) = f(\hat{\boldsymbol{x}} - \boldsymbol{x})$). The possibility of learning the identity function is avoided here as the target output (cleaner input, $\boldsymbol{x}$) differ from the network input (noisy input, $\tilde{\boldsymbol{x}}$). A properly trained denoising autoencoder learns the approximate structure of the manifold in a lower dimension [36]. The inputs to the autoencoder can be added with different types of noises like Gaussian, salt-pepper, Poisson, and speckle noise [60]. In the current study, a denoising autoencoder with a bottleneck structure with additive Gaussian noise is interpreted employing a gradient approach.

4. Computation of network gradients using Complex-step Derivative Approximation method (CSDA)

The goal of the paper is to generate saliency maps that highlight the important pixels that are responsible for the image reconstruction using denoising autoencoders. The saliency maps are generated by plotting contours of gradients. The gradients are obtained by taking derivatives of the latent variables ($\boldsymbol{g}$) with respect to the original cleaner input pixel features ($\boldsymbol{x}$). An accurate and fully automatable sensitivity method is ideal for the evaluation of these derivatives. Based on the past experience of the authors [61-64], we employed the complex step derivative approximation method (CSDA) [65] to evaluate the gradients of latent variables ($\boldsymbol{g}$) with respect to the input pixel features ($\boldsymbol{x}$). CSDA is used in this study as it offers several advantages over other numerical techniques. The defining feature of CSDA is that it is free of subtractive cancellation errors [66]. This allows us to obtain gradients at analytic level accuracy by choosing a very small step size ($h \ll 10^{-8}$). While other numerical techniques give erroneous estimates when functions are evaluated at a discontinuous point, CSDA can give correct estimates right up to the discontinuities [67]. CSDA can also find one-sided derivatives at discontinuous locations [67]. This is useful especially in the case of neural networks where discontinuous activation functions are often employed. CSDA has been extended to also calculate higher-order derivatives of scalar functions and Jacobians and Hessians of vector-valued functions [68]. Because of these advantages, CSDA has been popularly used in performing sensitivity analysis for engineering optimization [69, 70], approximating gradients of deep neural networks [63], numerically evaluating tangent moduli [62, 71, 72], and implementing second-order Kalman filter for non-linear state estimation [73, 74].

The Taylor's series expansion of an analytic function, f, developed about a real point with x with an imaginary step size ih, is expressed as follows:

$$f(x+ih) = f(x) + ihf'(x) - \frac{h^2}{2!}f''(x) - \frac{h^3}{3!}f'''(x) + \cdots \quad (1)$$

here, $i = \sqrt{-1}$ is the unit imaginary number and h is the real step size. It is to be noted that we assume f is real on the real axis.

When we equate the imaginary component of both sides of the equation (Eq. 1) and truncate terms higher than the first order, we obtain the approximation for the first-order derivative as follows

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

where, $Imag$ implies the imaginary component and $o(h^2)$ refers to the second-order truncation error. The expression given in Eq. 2 is the complex step derivative approximation to calculate the first-order derivative of a scalar function.

The gradients (first-order variation of outputs with respect to the input features) of a neural network is a vector-valued vector function. To obtain the output gradients with respect to the input features using CSDA, a perturbation-based approach is formulated. This approach is implemented in three steps: First, the network is configured and trained for a given classification or regression task. Configuration involves the selection of the appropriate number of hidden layers and the number of neurons in each hidden layer which provide the best possible network performance and generalization ability. Training involves the determination of the network weights and biases. Second, the input features of the trained network are perturbed (added) one at a time with an imaginary step size ih. The input vector with one perturbed input is fed to the neural network and the outputs corresponding to it are obtained through feed-forward operation. Finally, the gradients

of the output vector with respect to the perturbed input are obtained by dividing the imaginary components of complex outputs with the real step size h.

The implementation of CSDA in a simple two-layered neural network (see Fig. 3) and the underlying mathematical operations are illustrated here. The network has an input layer and an output layer. The input layer consists of 3 nodes which take the input $\boldsymbol{x} = \langle x_1 \;\; x_2 \;\; x_3 \rangle^T$ and the output layer consists of 2 nodes which produce the output $\boldsymbol{y} = \langle y_1 \;\; y_2 \rangle^T$. The output layer uses a Rectified Linear Unit (ReLU) activation function. Eq. 3 shows the expression to compute the activation value of the net input, z for ReLU function.

$$ReLU(z) = \begin{cases} 0 & if\ z \leq 0 \\ z & if\ z > 0 \end{cases} \tag{3}$$

The network is first trained for a regression task and the trained network parameters are $\boldsymbol{W} = \begin{bmatrix} w_{11} & w_{12} & w_{13} \\ w_{21} & w_{22} & w_{23} \end{bmatrix}$ and $\boldsymbol{b} = \begin{bmatrix} b_1 \\ b_2 \end{bmatrix}$. The feedforward operation to calculate $\boldsymbol{y}$ from $\boldsymbol{x}$ involves the calculation of the net input, $\boldsymbol{z} = \boldsymbol{W}\boldsymbol{x} + \boldsymbol{b}$, and the application of ReLU activation function to the net input, $\boldsymbol{y} = \boldsymbol{\phi}(\boldsymbol{z})$. The partial derivatives of $\boldsymbol{y}$ with respect to $\boldsymbol{x}$ is determined sequentially by choosing one x_i at a time**.** To calculate the partial derivatives $\partial y_1/\partial x_1$, $\partial y_2/\partial x_1$ and $\partial y_3/\partial x_1$, the input feature x_1 is perturbed (added) with an imaginary step size ih and the feed work operation is carried out. The mathematical operations involved in calculating $\partial y_1/\partial x_1$ is presented here.

The net input of the first node of the output layer is computed as follows,

$$z_1 = w_{11}(x_1 + ih) + w_{21}x_2 + w_{31}x_3 + b_1 \tag{4}$$

This can be rewritten as:

$$z_1 = w_{11}x_1 + w_{21}x_2 + w_{23}x_3 + b_1 + iw_{11}h \tag{5}$$

The ReLU activation function given in Eq. 3 is defined for real inputs. Since, in the current case, the inputs to the ReLU function are complex in nature, the conditions of the ReLU function is modified in the following way to find the activation value of z_1.

$$y_1 = \begin{cases} 0 + 0i & if\ Re(z_1) \leq 0 \\ Re(z_1) + Im(z_1) & if\ Re(z_1) > 0 \end{cases} \tag{6}$$

where, $Re(z_1)$ and $Im(z_1)$ are real components of z_1.

Assuming the case where $Re(z_1) > 0$, y_1 can be written as

$$y_1 = w_{11}x_1 + w_{21}x_2 + w_{23}x_3 + b_1 + iw_{11}h \tag{7}$$

Finally, the partial derivative of y_1 over x_1 is calculated as given below,

$$\frac{\partial y_1}{\partial x_1} = \frac{Im(y_1)}{h} = w_{11} \tag{8}$$

The partial derivative calculated in Eq.8 for this simple neural network is the same as that of the analytic derivative calculated through the chain rule. Although we can implement chain rule in neural networks to obtain analytical derivatives, it will be very challenging in the case of deep convolutional neural networks where the implementation involves backpropagating derivatives through numerous kernel filters and pooling layers. Since CSDA does not involve subtractive cancellation errors, we can automate the process of finding partial derivatives for the entire problem by choosing a common very smaller step size.

5. Implementation of CSDA in a denoising autoencoder to compute pixel attributions

In the current study, the proposed CSDA method to construct saliency maps is implemented in two stages. In the first stage, the preprocessed MNIST handwritten [75] digit images are fed into a denoised autoencoder and the network is trained to reconstruct the denoised images. In the second stage, a feedforward neural network is constructed using the network

parameters of the encoded part of the trained autoencoder network. Then, partial derivatives of each latent feature with respect to each pixel position are computed and used to develop the saliency maps. The schematic of the stages involved in evaluating the complex step gradients is shown in Fig. 4. The details regarding the dataset and the preprocessing involved are described in Section 5.1 and the detailed implementation procedure is given in Section 5.2.

5.1 Preparation of the input image data

The current study utilizes MNIST handwritten digits dataset for training the denoising autoencoder. The digital images are in greyscale (one channel) with an image size of 28×28 pixels. The greyscale values of each pixel range from 0 to 255. The training dataset is denoted by $\boldsymbol{D}_{train} \in \mathbb{R}^{n\times m}$, where n is the number of training image examples and m is the number of pixels in an image example. In the current case, n and m are 60000 and 784 respectively. Similarly, the testing dataset is denoted by $\boldsymbol{D}_{test} \in \mathbb{R}^{l\times m}$, where l is the number of testing images which is $10{,}000$ in our current case. A p^{th} row of the dataset (called a datapoint) is defined as $x^{(k)} = (\, x_{1,}^{(k)} \cdots, x_m^{(k)}\,)$, where $x_{1,}^{(k)} \cdots, x_m^{(k)}$ are the pixel features and k ranges from 1 to p or 1 to r . The gray pixel ranges of the both the datasets $\boldsymbol{D}_{train}$ and $\boldsymbol{D}_{test}$ are normalized by dividing with 255 and kept in the range of 0 to 1. This is done to improve the convergence rate of the gradient descent algorithm.

In denoising autoencoders, the inputs are added with noise and the clean denoised inputs are reconstructed from the noisy inputs. The reconstruction from the noisy inputs makes the denoising autoencoders learn only the important model features and reduces the risk of overfitting. The noisy datasets to be used in the current denoising autoencoder are obtained by adding gaussian noise to the normalized datasets. The training and testing datasets added with noise are denoted as

$\widetilde{\boldsymbol{D}}_{train}$ and $\widetilde{\boldsymbol{D}}_{test}$ respectively. The addition of gaussian noise to an image example is given in Eq. 9.

$$\widetilde{\boldsymbol{x}} = q(\boldsymbol{x} + \kappa \boldsymbol{N}) \tag{9}$$

Here, $\boldsymbol{x}$ is a normalized cleaner image input and $\widetilde{\boldsymbol{x}}$ is a noisy image input. q is a clip function which bounds the pixel intensity values between 0 and 1. κ is a scaling factor which alters the noise variance. Increasing the κ value increases the dispersion and sharpness of the added noise. No noise is added to the images when the λ is set zero. $\boldsymbol{N}$ is an array of independent probability values sampled from a gaussian distribution with zero mean and unit variance and its dimension is same size as that of $\boldsymbol{x}$ and $\widetilde{\boldsymbol{x}}$. Fig. 5 shows some examples of handwritten digit images before and after the addition of noises.

5.2 Evaluation of pixel attributions

A denoising autoencoder is configured and trained to reconstruct the images of handwritten digits of the MNIST database. We employed a trial and error approach to configuring the denoising autoencoder which involves the selection of an appropriate number of hidden layers and neurons. The autoencoder is said to be configured when it can reconstruct the input images without significant loss of information and is generalizable to the unseen images. This is ensured by limiting the binary cross-entropy of the network (measures the reconstruction loss of the network) to 0.1 for both the training and testing dataset. To elucidate, a few examples of the handwritten digit images reconstructed by the configured autoencoder is given in Fig. 6 along with the original images. The schematic of the configured denoising autoencoder is presented in Fig. 4. The encoder of the denoising autoencoder comprises an input layer with 784 nodes (corresponding to the total pixels of a handwritten digit image) and a hidden representation layer with 128 nodes, followed

by a bottleneck layer (called 'code') consisting of 32 nodes. Thus, in the current autoencoder the noise-added 784-pixel information, $\hat{\boldsymbol{x}}$ is compressed into a latent variable vector ($\boldsymbol{g}$) of dimensional size 32. The compressed data in the latent space is then expanded and the original cleaner images are reconstructed through a decoder network. The decoder network follows the inverted form of the encoder network and thus forms a symmetrical autoencoder network. The last layer of the decoder (784 nodes) corresponds to the reconstructed image output, $\hat{\boldsymbol{x}}$. ReLU activation function (see Eq. 3) is used for all the intermediate layers of the network and for the final layer, a Sigmoid activation (Eq. 10, where z is the netput) function is employed.

$$Sigmoid(z) = \frac{1}{1 + e^{-z}} \tag{10}$$

The performance of the network was measured using Binary cross-entropy between the original image pixel intensities and reconstructed image pixel intensities as shown in Eq. 11. p_i in the equation refers to the predicted probability of a pixel being '1'.

$$\mathcal{L}(\boldsymbol{x}, \hat{\boldsymbol{x}}) = -\frac{1}{N}\sum_{i=1}^{N} \log p_i \tag{11}$$

The weights and biases of the encoder and decoder are not tied-in and allowed to vary independently. The network weights are updated based on ADAM (adaptive moment estimation) optimization [76] during the training. The configuration parameters ADAM used in our analysis are presented in Table 2. The model is fitted by running through 100 epochs with a mini-batch size of 25. The final cross-entropy loss of both the training dataset and testing dataset is in the range of 0.08-0.09.

In the second stage, a simple feedforward neural network is constructed using the encoder part of the successfully trained autoencoder. The last layer of this network is the 'code' layer of

the autoencoder. The weights and biases of the encoder are simply transferred to this feedforward neural network. Thus, when a handwritten digital image is fed into this network as pixel features, $\boldsymbol{x} = (x_1, x_2, \dots x_m)$, its dimensionally reduced latent space variables, $\boldsymbol{g} = (g_1, g_2, \dots, g_q)$ are generated as the output. Here, m denotes the number of input pixels and q denotes the number of latent variables which are 784 and 32 respectively for our case.

With the latent variables and the network parameters of the decoder, we can reconstruct the original input images. Therefore, these latent variables contain the necessary information to reconstruct the original image. Hence, we consider the input pixels which significantly contribute to the latent variables are important for the image reconstruction. The contribution of input pixels for the construction of latent space is otherwise known as the pixel attribution to the latent variables. To determine the attribution, R_{x_i}, of an input pixel (x_i) (contribution of a pixel feature for image reconstruction), we employ the CSDA approach discussed in Section 4 and compute the partial derivatives ($\partial g_1/\partial x_i\,, \partial g_2/\partial x_i \dots \partial g_q/\partial x_i$) of the latent features with respect to the input pixel feature. The computation of partial derivatives with respect to x_i through CSDA involves perturbation of x_i, calculation of complex latent outputs $g_1, g_2, \dots, g_q$ and finally, dividing of the imaginary components of latent outputs with step size, h ($\partial g_1/\partial x_i = Im(g_1)/h, \dots, \partial g_n/\partial x_i = Im(g_q)/h$). The step size in the current analysis is taken as $h = 10^{-16}$ to minimize the truncation error (see Eq. 2) without increasing the subtractive cancellation error. To determine a single measure for the attribution (R_{x_i}) of an input, we use the L_2 norm of the partial derivatives of the latent features ($\boldsymbol{g} = \{g_1 \dots . g_{32}\}$) with respect to each original clean input feature $x_{i=1:784}$ which is given as

$$R_{x_i} = \sqrt{\left(\partial g_1/\partial {x_i}^2 + \cdots + \partial g_{32}/\partial {x_i}^2\right)} \tag{12}$$

The attributions of each pixel feature $(R_{x_1}, R_{x_2}, \ldots, R_{x_{784}})$ are computed sequentially using this process and a saliency map of size 28×28 is constructed for the given input image. This entire process of generating the saliency maps is automated through a simple loop sub-routine. Higher saliency values at a pixel position indicate that the corresponding pixel feature significantly influences the latent representation and therefore the reconstruction.

6. Generation of Saliency maps

Saliency maps in the forms of contours highlighting the attributions of each pixel feature to the latent features are constructed for each handwritten image using the procedure described in the previous section. Three examples of such saliency maps constructed are shown in Fig. 7 along with its input images for the purpose of illustration. As seen in these contours, the regions occupied by the handwritten digital strokes show higher attribution values whereas the white regions with no information possess lower attributions. This demonstrates the proposed approach identifies the important pixels necessary for the reconstruction. As the saliency maps are dependent on the input image, the contours slightly differ for each example. However, as the differences are not notably significant, we can make a general interpretation from the local pixel attributions. Furthermore, we have formulated and performed three sanity checks to confirm the validity of the obtained contours. These sanity checks are also useful to gain more understanding of how the input space influences the latent variables.

7. Sanity Checks

Sanity Check-1: training using single-digit image subset: Since the regions occupied by the strokes of the digits will have more attribution values in the saliency maps, it is expected that

saliency contours of the network trained with only one class of digits would closely resemble the shape of those digits. To verify this premise, images of the database pertaining to each digit are separated and the autoencoder is trained with images of digits belonging to only one category. Then, the trained network weights and biases are used to construct the saliency maps. As expected, and it is clearly seen in Fig. 8, the contours look very similar to the shape of the digits that were used to train the network.

Sanity Check-2: spatial translation of digits and saliency maps: The contours of the developed saliency maps are centered in the maps as the handwritten digits used for the training are also centered in the images of fixed size. Hence, if the digits are translated off from the center, the contours should show a similar shift. This is verified by offsetting the digit on the top-left corner of the images and it is carried out by adding rows and columns of white pixels at the bottom and right side of the original image. The original image of size 28×28 pixels is converted into 42×42 pixels with the digits being pushed to the top-left position (see Fig. 9a). The procedure mentioned in section 5 is then repeated and a constructed contour is presented in Fig. 9b. The contour in Fig. 9b clearly shows the corresponding shift of the contours to the top-left position and validates this supposition.

Sanity Check-3: Corrupting the most and the least important pixels: In the above two checks, we have qualitatively shown the developed saliency maps are meaningfully highlighting the pixel attributions. To verify whether saliency values attributed to the pixels for the image reconstruction are valid, we have formulated another sanity check. As in our current case, the digit strokes are the only features present in the image, the pixel positions important for the image reconstruction will also be important for the classification of images. Hence it is expected that if the most important pixels (identified by the saliency maps) of the image dataset are corrupted and

fed into a classifier, the classification accuracy would be decreased. Conversely, if the least important pixels in the input image dataset to a classifier are corrupted, the classification should remain unaltered. To verify this assumption, we identified 10%, 20%, and 30% percentages of the most and least important pixel locations based on a local saliency map and corrupted those pixels. As the local saliency maps do not differ significantly from each other, a local saliency map is randomly chosen as the representative of the global saliency map. The pixels are corrupted based on the grey intensity level, if the grey intensity value is above 100, it is changed to 0 and if the value is less than or equal to100 it is changed to 255. Fig. 10 shows the extent of alteration of images due to the applied corruptions. Fig. 10 clearly indicates the corruptions on the important pixels lead to the significant erasure of digits strokes. These corruptions are made to the testing dataset of the MNIST handwritten images. The classification accuracies of these corrupted testing datasets are determined using a convolutional neural network (CNN) trained on the uncorrupted MNIST handwritten digits. The employed convolutional neural network consists of three convolutional layers alternated with max-pooling layers. The first convolutional layer uses 32 numbers of 3×3 kernel filters and the second and the third layer use 64 numbers of 3×3 kernel filters. ReLU activation function is used in all three convolutional layers for non-linear activation. The last (third) max-pooling layer is flattened to a fully connected layer of size 576. It is further connected to a hidden layer of size 128 with a ReLU activation function. Finally, the hidden layer is connected to a softmax layer of size 10 corresponding to each class of the digits. The summary of the network architecture is presented in Table 3. The classification accuracies of the various corrupted testing datasets are plotted as in Fig.11. As observed in the plot, the classification accuracies are reduced considerably from 99.9% to 40% when the percentage of corrupted important pixels is increased from 10% to 30%. However, the percentage of corrupted least

important pixels had little effect on the classification accuracies. All the three checks described above clearly demonstrate that the proposed gradient attribution method determines the importance of each pixel location for image reconstruction of denoising autoencoders and offers an interpretability measure for data scientists and analysts.

8. Conclusions

In this study, we examined the importance of each individual pixel feature of an image for the image reconstruction. The importance of pixel features was evaluated through the attributions of the pixel features to the latent variables of a denoising autoencoder used for image reconstruction. Saliency maps showing the pixel attribution contours were developed in the end. We also verified the fidelity of the saliency maps using three sanity checks introduced in this study. Through the saliency maps and sanity checks, we were able to identify the important pixel features for the image reconstruction. The key outcomes of this study are listed below:

1) Saliency maps in the form of contours depicting the attribution of each pixel feature to image reconstruction are generated. Higher intensity regions of the saliency maps show the most important image pixels used by the denoising autoencoder for image reconstruction.
2) The first saliency check is performed to verify whether the attribution contours of a single class of digits resemble the shape of the trained digit. This is envisaged because the digit strokes of the same class exhibit less variation in occupying the pixel regions. The saliency maps developed for a denoising autoencoder trained with a single class of digits validate this supposition.
3) Since the attribution contours are influenced by the pixel regions frequently occupied with digit strokes, it is presumed that a translational shift in trained images exhibits a similar shift in the contours of the saliency maps. This is confirmed through the saliency maps of a

denoising autoencoder trained with images whose digits are translated to the top-left corner. The contours in the saliency maps are also translated to the top-left corner.

4) The third saliency check is formulated to validate the pixel attribution values in the context of image classification. A drop in the classification accuracy is expected when the most important pixels (indicated by the attribution values) of the images used for the testing are corrupted. As expected, the classification accuracy dropped drastically from 99.99% to 40% when 30% of the most important pixels are corrupted. Also, the classification accuracy is unaltered when 30% of the least important pixels are corrupted.

As the proposed approach is perturbation-based, it can be used to find the attributions of specific features of the inputs to the outputs. This approach can be readily applied on already deployed networks and can be extended to other deep autoencoders architectures and convolution neural networks. The sanity checks introduced in the current work can be used as a framework to validate the other attributions-based interpretability methods.

Acknowledgment

Research presented in this paper was supported by the National Science Foundation under NSF EPSCoR Track-1 Cooperative Agreement OIA #1946202. 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.

References

[1] C. Rudin, Stop explaining black box machine learning models for high stakes decisions and use interpretable models instead, Nature Machine Intelligence, 1 (2019) 206-215.

[2] C. Rudin, C. Wang, B. Coker, The age of secrecy and unfairness in recidivism prediction, arXiv preprint arXiv:1811.00731, (2018).

[3] A. Vellido, The importance of interpretability and visualization in machine learning for applications in medicine and health care, Neural computing and applications, 32 (2020) 18069-18083.

[4] R. Lian, K. Siau, Values of Trust in AI in Autonomous Driving Vehicles, (2020).

[5] N. Gillespie, C. Curtis, R. Bianchi, A. Akbari, R. Fentener van Vlissingen, Achieving Trustworthy AI: A Model for Trustworthy Artificial Intelligence, (2020).

[6] J.A. Nelder, R.W. Wedderburn, Generalized linear models, Journal of the Royal Statistical Society: Series A (General), 135 (1972) 370-384.

[7] P. McCullagh, J.A. Nelder, Generalized linear models, Routledge, 2019.

[8] D. Wei, S. Dash, T. Gao, O. Gunluk, Generalized linear rule models, in: International Conference on Machine Learning, PMLR, 2019, pp. 6687-6696.

[9] T.J. Hastie, R.J. Tibshirani, Generalized additive models, Routledge, 2017.

[10] A. Chouldechova, T. Hastie, Generalized additive model selection, arXiv preprint arXiv:1506.03850, (2015).

[11] B. Ustun, C. Rudin, Supersparse linear integer models for optimized medical scoring systems, Machine Learning, 102 (2016) 349-391.

[12] S. Dash, O. Günlük, D. Wei, Boolean decision rules via column generation, arXiv preprint arXiv:1805.09901, (2018).

[13] M.T. Ribeiro, S. Singh, C. Guestrin, " Why should i trust you?" Explaining the predictions of any classifier, in: Proceedings of the 22nd ACM SIGKDD international conference on knowledge discovery and data mining, 2016, pp. 1135-1144.

[14] D. Garreau, U. Luxburg, Explaining the explainer: A first theoretical analysis of LIME, in: International Conference on Artificial Intelligence and Statistics, PMLR, 2020, pp. 1287-1296.

[15] S.M. Lundberg, S.-I. Lee, A unified approach to interpreting model predictions, in: Proceedings of the 31st international conference on neural information processing systems, 2017, pp. 4768-4777.

[16] L. Itti, C. Koch, E. Niebur, A model of saliency-based visual attention for rapid scene analysis, IEEE Transactions on pattern analysis and machine intelligence, 20 (1998) 1254-1259.

[17] M.D. Zeiler, R. Fergus, Visualizing and Understanding Convolutional Networks, in, Springer International Publishing, Cham, 2014, pp. 818-833.

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

[19] J.T. Springenberg, A. Dosovitskiy, T. Brox, M. Riedmiller, Striving for simplicity: The all convolutional net, arXiv preprint arXiv:1412.6806, (2014).

[20] M. Sundararajan, A. Taly, Q. Yan, Axiomatic attribution for deep networks, in: International Conference on Machine Learning, PMLR, 2017, pp. 3319-3328.

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

[22] R.R. Selvaraju, M. Cogswell, A. Das, R. Vedantam, D. Parikh, D. Batra, Grad-cam: Visual explanations from deep networks via gradient-based localization, in: Proceedings of the IEEE international conference on computer vision, 2017, pp. 618-626.

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

[24] D. Bank, N. Koenigstein, R. Giryes, Autoencoders, arXiv preprint arXiv:2003.05991, (2020).

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

[26] X. Guo, X. Liu, E. Zhu, J. Yin, Deep clustering with convolutional autoencoders, in: International conference on neural information processing, Springer, 2017, pp. 373-382.

[27] X. Peng, J. Feng, S. Xiao, W.-Y. Yau, J.T. Zhou, S. Yang, Structured autoencoders for subspace clustering, IEEE Transactions on Image Processing, 27 (2018) 5076-5086.

[28] J. Zabalza, J. Ren, J. Zheng, H. Zhao, C. Qing, Z. Yang, P. Du, S. Marshall, Novel segmented stacked autoencoder for effective dimensionality reduction and feature extraction in hyperspectral imaging, Neurocomputing, 185 (2016) 1-10.

[29] W. Wang, Y. Huang, Y. Wang, 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.

[30] C. Zhou, 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.

[31] J. An, S. Cho, Variational autoencoder based anomaly detection using reconstruction probability, Special Lecture on IE, 2 (2015) 1-18.

[32] C. Nash, C.K. Williams, The shape variational autoencoder: A deep generative model of part-segmented 3D objects, in: Computer Graphics Forum, Wiley Online Library, 2017, pp. 1-12.

[33] Y. Wu, Y. Burda, R. Salakhutdinov, R. Grosse, On the quantitative analysis of decoder-based generative models, arXiv preprint arXiv:1611.04273, (2016).

[34] Y. Wu, C. DuBois, A.X. Zheng, M. Ester, Collaborative denoising auto-encoders for top-n recommender systems, in: Proceedings of the ninth ACM international conference on web search and data mining, 2016, pp. 153-162.

[35] S. Sedhain, A.K. Menon, S. Sanner, L. Xie, Autorec: Autoencoders meet collaborative filtering, in: Proceedings of the 24th international conference on World Wide Web, 2015, pp. 111-112.

[36] G. Alain, Y. Bengio, What regularized auto-encoders learn from the data-generating distribution, The Journal of Machine Learning Research, 15 (2014) 3563-3593.

[37] G. Mercatali, A. Freitas, Disentangling Generative Factors in Natural Language with Discrete Variational Autoencoders, arXiv preprint arXiv:2109.07169, (2021).

[38] M. Curi, G.A. Converse, J. Hajewski, S. Oliveira, Interpretable Variational Autoencoders for Cognitive Models, in: 2019 International Joint Conference on Neural Networks (IJCNN), 2019, pp. 1-8.

[39] S. Rybakov, M. Lotfollahi, F.J. Theis, F.A. Wolf, Learning interpretable latent autoencoder representations with annotations of feature sets, bioRxiv, (2020) 2020.2012.2002.401182.

[40] J.-Y. Kim, S.-B. Cho, Explainable prediction of electric energy demand using a deep autoencoder with interpretable latent space, Expert Systems with Applications, 186 (2021) 115842.

[41] R. Al-Hmouz, W. Pedrycz, A. Balamash, A. Morfeq, Logic-driven autoencoders, Knowledge-Based Systems, 183 (2019) 104874.

[42] A. Ng, Sparse autoencoder, CS294A Lecture notes, 72 (2011) 1-19.

[43] P. Rivas, E. Rivas, O. Velarde, S. Gonzalez, Deep Sparse Autoencoders for American Sign Language Recognition Using Depth Images, in: Proceedings on the International Conference on Artificial Intelligence (ICAI), The Steering Committee of The World Congress in Computer Science, Computer …, 2019, pp. 438-444.

[44] C.S.N. Pathirage, J. Li, L. Li, H. Hao, W. Liu, R. Wang, Development and application of a deep learning–based sparse autoencoder framework for structural damage identification, Structural Health Monitoring, 18 (2019) 103-122.

[45] I.D. Mienye, Y. Sun, Z. Wang, Improved sparse autoencoder based artificial neural network approach for prediction of heart disease, Informatics in Medicine Unlocked, 18 (2020) 100307.

[46] B.G. Gebre, O. Crasborn, P. Wittenburg, S. Drude, T. Heskes, Unsupervised feature learning for visual sign language identification, (2014).

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

[48] D.P. Kingma, M. Welling, An introduction to variational autoencoders, arXiv preprint arXiv:1906.02691, (2019).

[49] M. Tavakoli, P. Baldi, Continuous representation of molecules using graph variational autoencoder, arXiv preprint arXiv:2004.08152, (2020).

[50] S. Sinai, E. Kelsic, G.M. Church, M.A. Nowak, Variational auto-encoding of protein sequences, arXiv preprint arXiv:1712.03346, (2017).

[51] E. Puyol-Antón, B. Ruijsink, J.R. Clough, I. Oksuz, D. Rueckert, R. Razavi, A.P. King, Assessing the Impact of Blood Pressure on Cardiac Function Using Interpretable Biomarkers and Variational Autoencoders, in, Springer International Publishing, Cham, 2020, pp. 22-30.

[52] S. Mohammadi, B. O'Dowd, C. Paulitz-Erdmann, L. Goerlitz, Penalized Variational Autoencoder for Molecular Design, (2021).

[53] E. Lin, S. Mukherjee, S. Kannan, A deep adversarial variational autoencoder model for dimensionality reduction in single-cell RNA sequencing analysis, BMC bioinformatics, 21 (2020) 1-11.

[54] P. Vincent, H. Larochelle, Y. Bengio, P.-A. Manzagol, Extracting and composing robust features with denoising autoencoders, in: Proceedings of the 25th international conference on Machine learning, 2008, pp. 1096-1103.

[55] V. Teixeira, R. Camacho, P.G. Ferreira, Learning influential genes on cancer gene expression data with stacked denoising autoencoders, in: 2017 IEEE International Conference on Bioinformatics and Biomedicine (BIBM), IEEE, 2017, pp. 1201-1205.

[56] L. Gondara, Medical image denoising using convolutional denoising autoencoders, in: 2016 IEEE 16th international conference on data mining workshops (ICDMW), IEEE, 2016, pp. 241-246.

[57] K. Cho, Boltzmann machines and denoising autoencoders for image denoising, arXiv preprint arXiv:1301.3468, (2013).

[58] O.M. Saad, Y. Chen, Deep denoising autoencoder for seismic random noise attenuation, Geophysics, 85 (2020) V367-V376.

[59] Y. Tao, X. Gao, K. Hsu, S. Sorooshian, A. Ihler, A deep neural network modeling framework to reduce bias in satellite precipitation products, Journal of Hydrometeorology, 17 (2016) 931-945.

[60] J. Owotogbe, T. Ibiyemi, B. Adu, A comprehensive review on various types of noise in image processing, International Journal of Scientific and Engineering Research, 10 (2019) 388-393.

[61] R. Kiran, L. Li, K. Khandelwal, Complex perturbation method for sensitivity analysis of nonlinear trusses, Journal of Structural Engineering, 143 (2017) 04016154.

[62] R. Kiran, K. Khandelwal, Complex step derivative approximation for numerical evaluation of tangent moduli, Computers & Structures, 140 (2014) 1-13.

[63] R. Kiran, D.L. Naik, Novel sensitivity method for evaluating the first derivative of the feed-forward neural network outputs, Journal of Big Data, 8 (2021) 1-13.

[64] D.L. Naik, R. kiran, A novel sensitivity-based method for feature selection, Journal of Big Data, 8 (2021) 128.

[65] J.N. Lyness, C.B. Moler, Numerical differentiation of analytic functions, SIAM Journal on Numerical Analysis, 4 (1967) 202-210.

[66] J.R. Martins, P. Sturdza, J.J. Alonso, The complex-step derivative approximation, ACM Transactions on Mathematical Software (TOMS), 29 (2003) 245-262.

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

[68] K.-L. Lai, J. Crassidis, Extensions of the first and second complex-step derivative approximations, Journal of Computational and Applied Mathematics, 219 (2008) 276-293.

[69] J. Chun, Reliability-Based Design Optimization of Structures Using Complex-Step Approximation with Sensitivity Analysis, Applied Sciences, 11 (2021) 4708.

[70] A. Callejo, O. Bauchau, B. Diskin, L. Wang, Sensitivity analysis of beam cross-section stiffness using adjoint method, in: International Design Engineering Technical Conferences and

Computers and Information in Engineering Conference, American Society of Mechanical Engineers, 2017, pp. V006T010A058.

[71] M. Tanaka, M. Fujikawa, D. Balzani, J. Schröder, Robust numerical calculation of tangent moduli at finite strains based on complex-step derivative approximation and its application to localization analysis, Computer Methods in Applied Mechanics and Engineering, 269 (2014) 454-470.

[72] H. Liu, W. Sun, Computational efficiency of numerical approximations of tangent moduli for finite element implementation of a fiber-reinforced hyperelastic material model, Computer methods in biomechanics and biomedical engineering, 19 (2016) 1171-1180.

[73] K.-L. Lai, J. Crassidis, Y. Cheng, J. Kim, New complex-step derivative approximations with application to second-order kalman filtering, in: AIAA Guidance, Navigation, and Control Conference and Exhibit, 2005, pp. 5944.

[74] V. Vittaldev, R.P. Russell, N. Arora, D. Gaylor, Second-order Kalman filters using multi-complex step derivatives, in: Proceedings of the AAS/AIAA Space Flight Mechanics Meeting, Kauai, Hawaii, 2012.

[75] L. Deng, The mnist database of handwritten digit images for machine learning research [best of the web], IEEE Signal Processing Magazine, 29 (2012) 141-142.

[76] D.P. Kingma, J. Ba, Adam: A method for stochastic optimization, arXiv preprint arXiv:1412.6980, (2014).

Tables

Table 1: Cost functions of various autoencoders

Autoencoders	Cost function
Sparse autoencoders	$\mathcal{L}(\boldsymbol{x}, \hat{\boldsymbol{x}}) + \lambda \sum_i \lvert a_i \rvert$
Contractive autoencoders	$\mathcal{L}(\boldsymbol{x}, \hat{\boldsymbol{x}}) + \lambda \lVert \boldsymbol{J}(\boldsymbol{x}) \rVert_F^2$ $J(\boldsymbol{x}) = \sum_{ij} \left(\partial g_j / \partial x_i\right)^2$
Variational autoencoders	$\mathcal{L}(\boldsymbol{x}, \hat{\boldsymbol{x}}) + \lambda \sum_j KL(q_j(\boldsymbol{g}\lvert\boldsymbol{x}) \lvert\rvert p(\boldsymbol{g}))$
Denoising autoencoders	$\mathcal{L}(\boldsymbol{x}, \hat{\boldsymbol{x}})$

where, $\mathcal{L}$ denotes the reconstruction loss, $\boldsymbol{x}$ is the input, $\hat{\boldsymbol{x}}$ is the reconstructed input with some information loss, $\boldsymbol{g}$ is the latent vector, λ is the regularization parameter, a_i is i^{th} activation of the latent layer, $\boldsymbol{J}$ is the Jacobian of the partial derivatives, g_j is the j^{th} latent variable, x_i is the i^{th} input feature. KL is the divergence function, $q_j(\boldsymbol{g}|\boldsymbol{x})$ – is the learned distribution of a dimension of the code layer and $p(\boldsymbol{g})$) is a true prior distribution which is assumed to follow unit gaussian distribution.

Table 2: Parametric Values of the ADAM optimizer employed in the study

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

Table 3: Network architecture details of CNN used in the *Sanity Check-3: Corrupting the most and the least important pixels*

Layer	**Layer size**	**Activation function**	**Data Size**
Input	-	-	28×28×1
Convolutional Layer 1	32×3×3	ReLU	28×28×32
Max pool Layer 1	2×2	-	14×14×32
Convolutional Layer 2	64×3×3	ReLU	14×14×64
Max pool Layer 2	2×2	-	7×7×64
Convolutional Layer 3	64×3×3	ReLU	7×7×64
Max pool Layer 3	2×2	-	3×3×64
Fully connected layer 1	576×1	-	576×1
Hidden layer	128×1	ReLU	128×1
Output layer	10×1	SoftMax	10×1

Figures

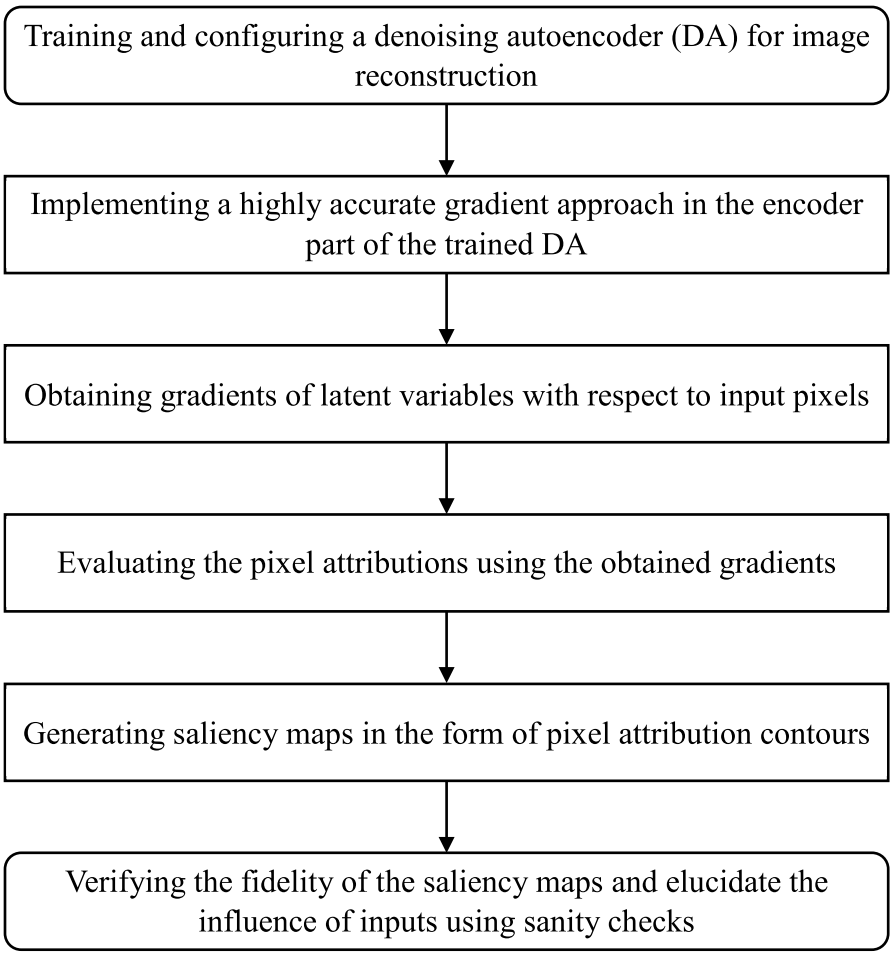


Fig. 1: Flow chart of the research approach employed in the current study.

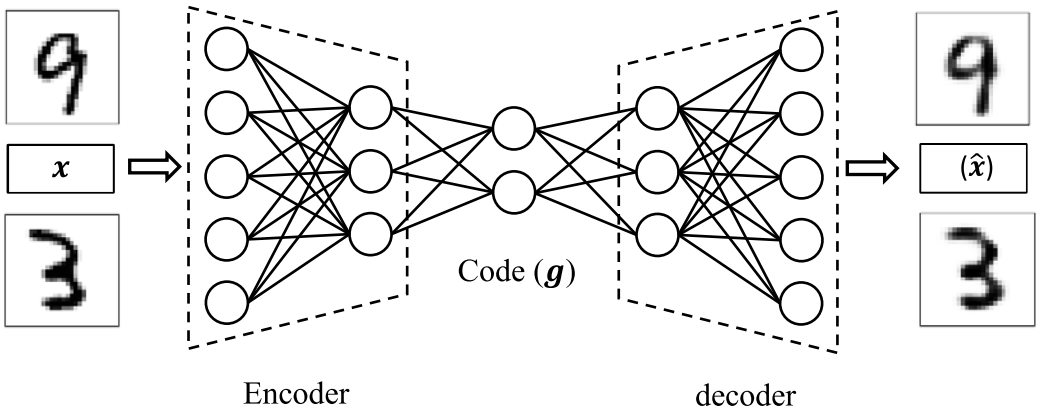


Fig. 2: A schematic of a simple autoencoder. The autoencoder consists of three components: an encoder – compresses the input ($\boldsymbol{x}$), a code layer ($\boldsymbol{g}$) – dimensionally reduced latent space and a decoder – reconstructs the input ($\hat{\boldsymbol{x}}$) with some loss of information.

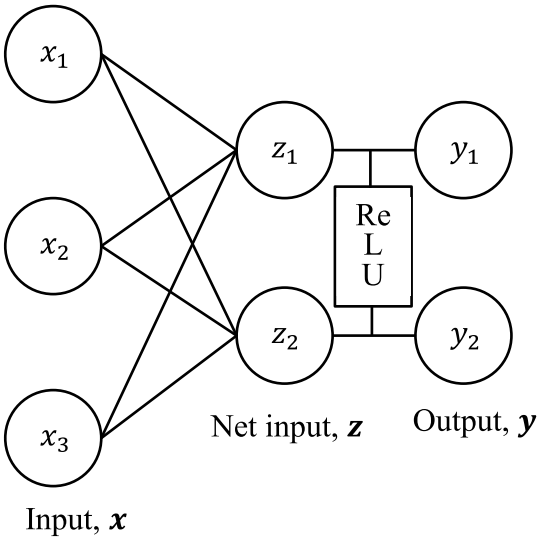
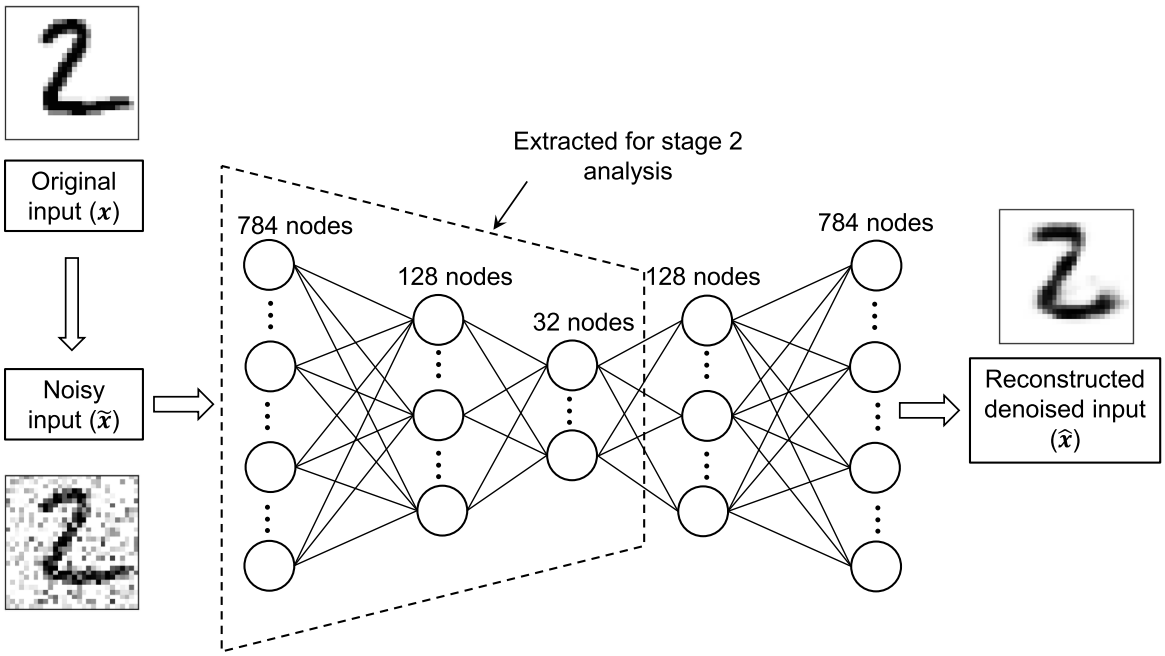
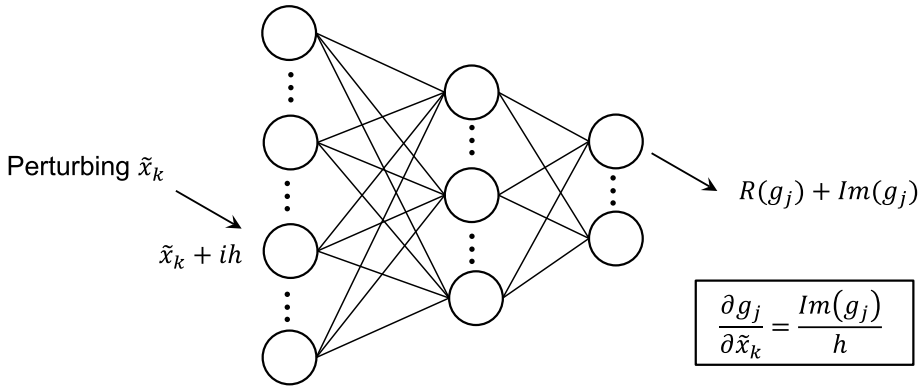


Fig. 3: Structure of the simple feedforward network used to illustrate the implementation of CSDA. The network consists of three input features (x_1, x_2 and x_3) and two outputs (y_1 and y_2). ReLU function is used to activate the net input values (z_1 and z_2).



Stage 1: Training and configuration of denoising autoencoder



Stage 2: Perturbation of input nodes to evaluate complex step gradients

Fig. 4: Schematic of the stages involved for evaluating partial derivatives of latent variables with respect to input features. Stage 1 involves training and configuration of the denoising autoencoder and Stage 2 involves sequential perturbation of inputs with an imaginary step size and ih and calculation of partial derivatives of latent variables with respect to the input features.

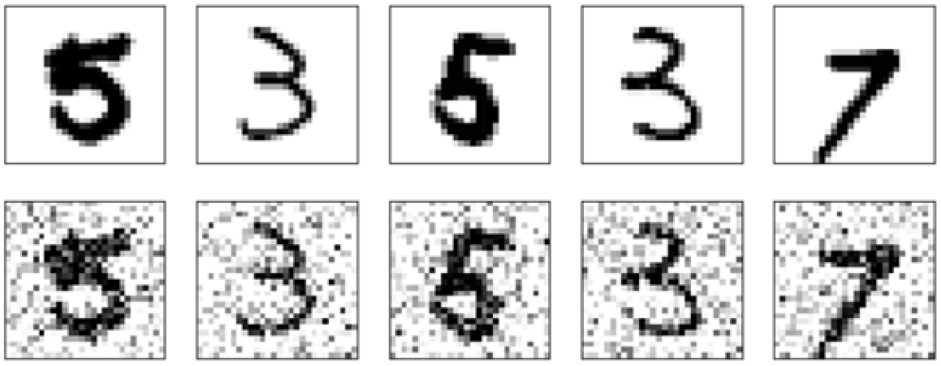


Fig. 5: Handwritten digit images before (top row) and after (bottom row) the addition of gaussian noises

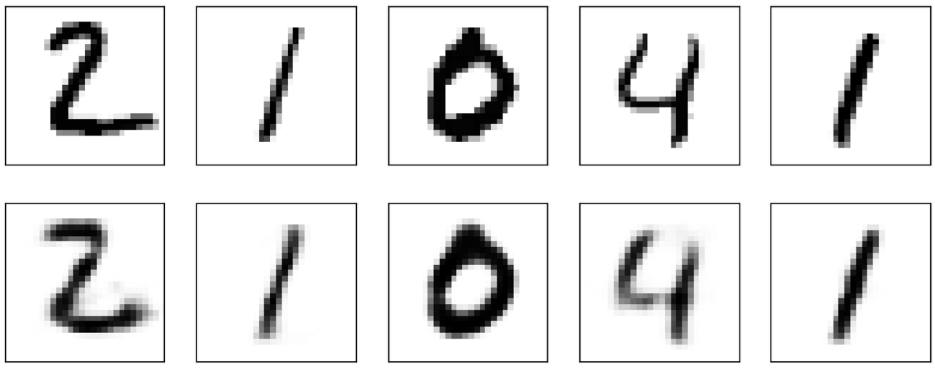


Fig. 6: Top row shows the original input handwritten images, and the bottom row shows the reconstructed handwritten digit images by the configured denoising autoencoder. The structure of the autoencoder is configured such that the binary cross entropy of training and testing is less than 0.1

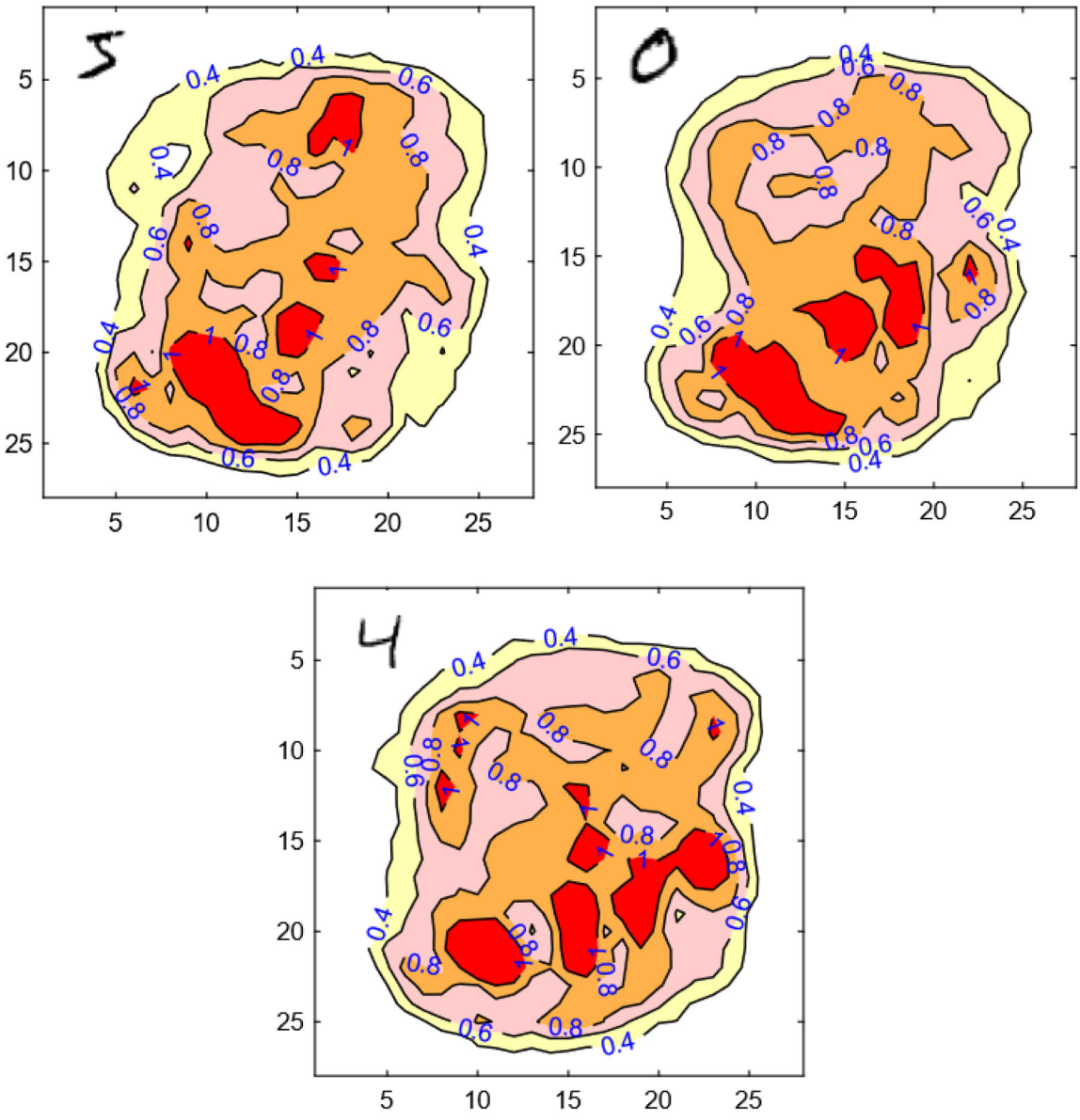


Fig.7: Image based saliency maps for three examples of MNIST handwritten digits. The input images are shown on the top-left corner of the maps. Red regions of the maps indicate the high attribution values of the pixel features and white regions indicate the almost zero attributions of pixel regions.

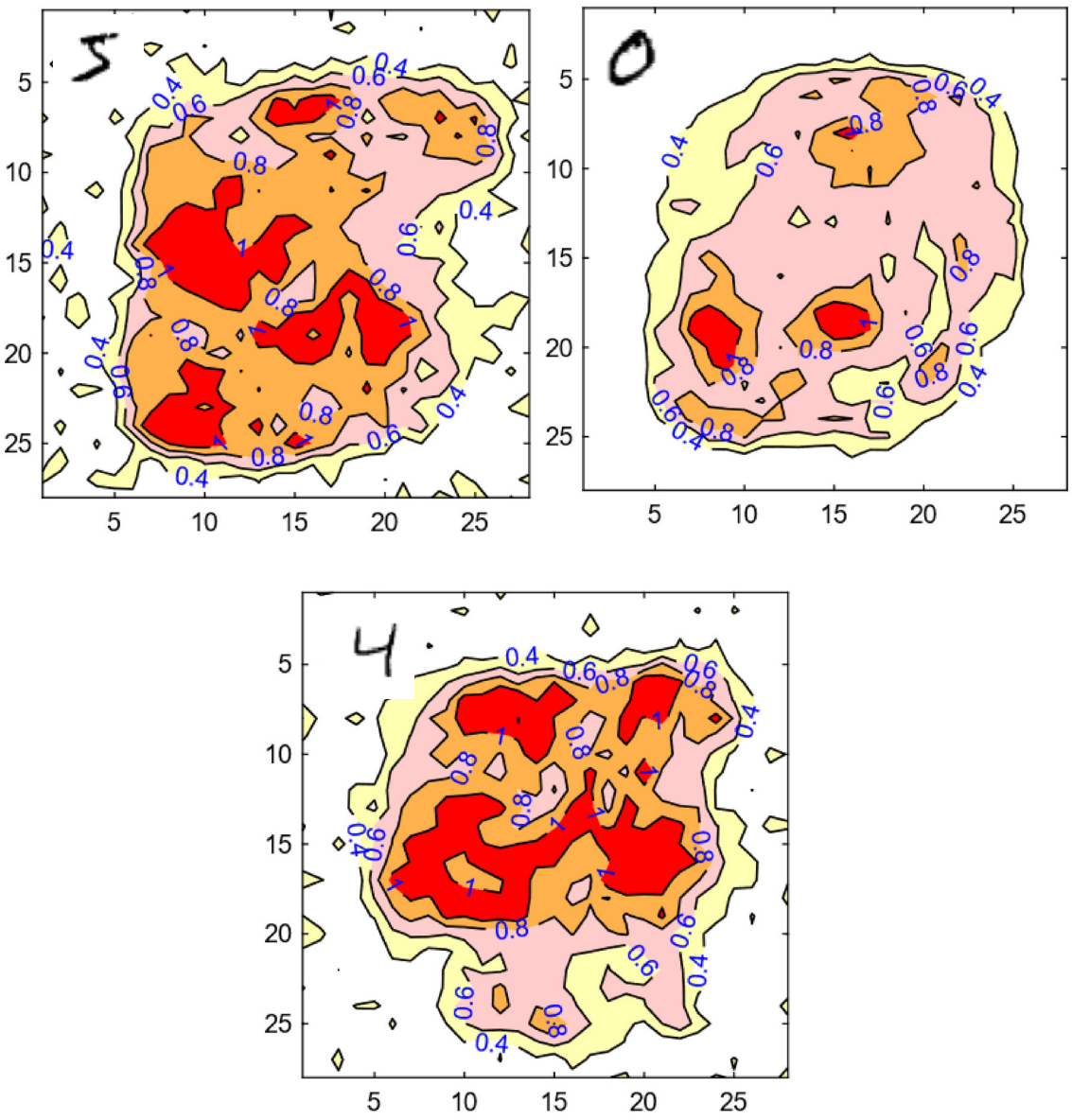


Fig. 8: Saliency maps of digit images used in the denoising autoencoders trained with one class of digits. Red regions of the maps indicate the high attribution values of the pixel features and white regions indicate the almost zero attributions of pixel regions. The saliency maps resemble the shape of the digits class the network is trained with.

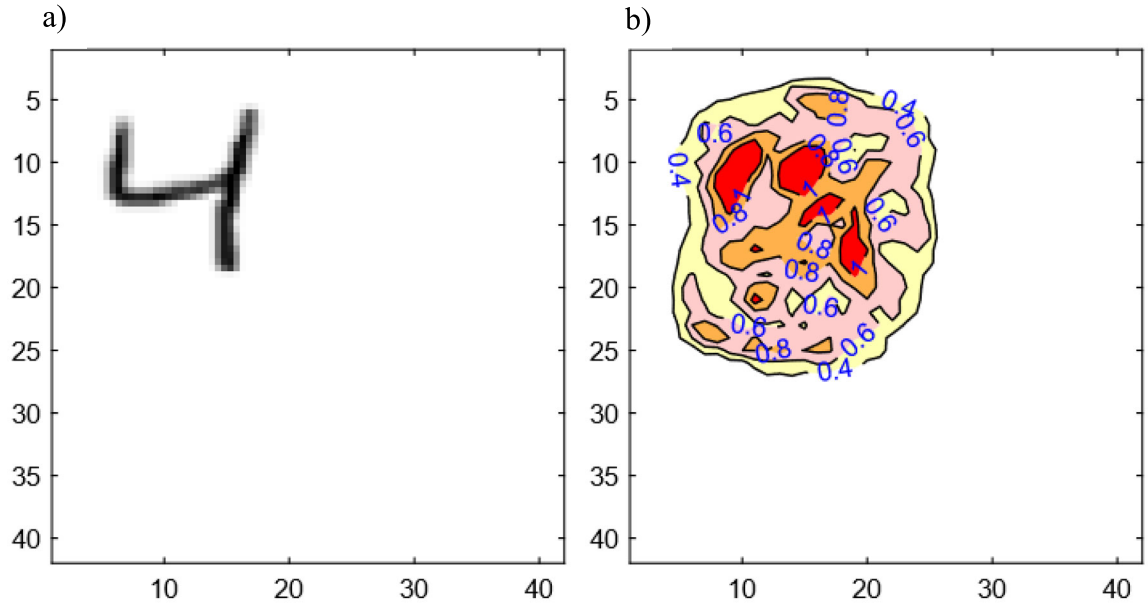


Fig. 9: a) an example of images offset into top left corner b) corresponding saliency map of complex step gradient norms. The contours in the saliency maps shows the same shift as that of the shift introduced in the trained images.

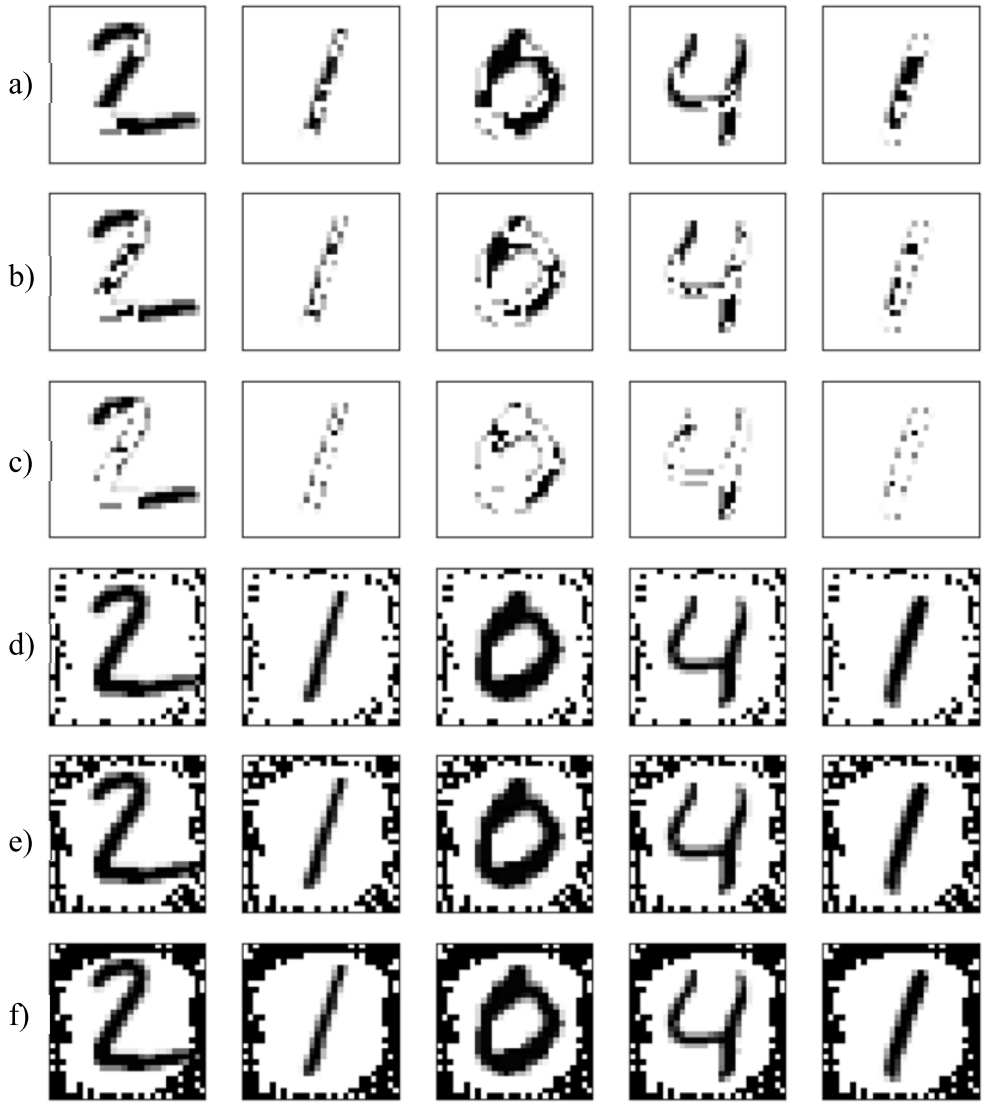


Fig. 10: Examples of testing images used by the classifier. a), b) and c) shows the image examples in which 10%, 20% and 30% of the most important regions are corrupted. d), e) and f) shows the image examples in which 10%, 20% and 30% of the least important regions are corrupted. Most and least important pixel regions of the images are identified through the generated saliency map. The corruption of most important regions shows the significant erasure of strokes of the handwritten digits.

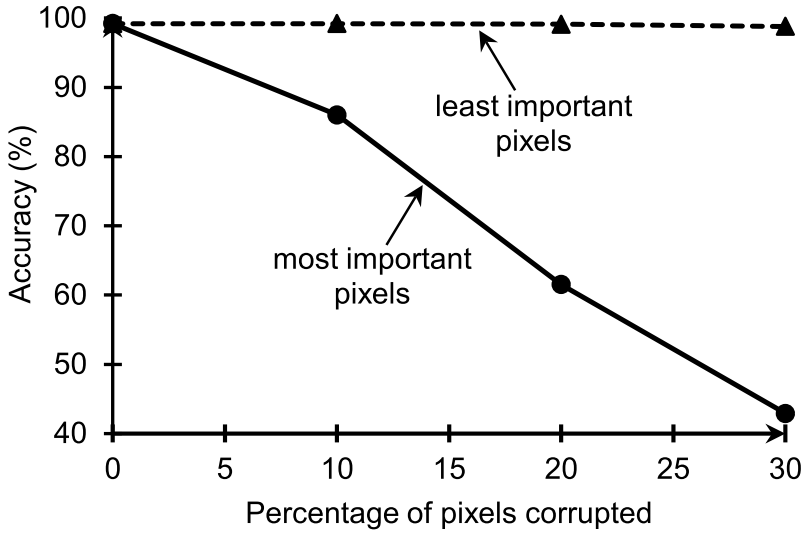


Fig. 11: Plot between classification accuracy (%) and percentage of pixels corrupted. An almost flat dashed line pertains to the corruption of the least important pixels and the solid line with accuracy dropping to 40% pertains to the corruption of most important pixels.