Machine Learning—Basic Unsupervised Methods (Cluster Analysis Methods, t-SNE)

M. Espadoto, S. B. Martins, W. Branderhorst and A. Telea

Abstract

Understanding how trained deep neural networks achieve their inferred results is challenging but important for relating how patterns in the input data affect other patterns in the output results. We present a visual analytics approach to this problem that consists of two mappings. The so-called forward mapping shows the relative impact of user-selected input patterns to all elements of the output. The backward mapping shows the relative impact of all input elements to user-selected patterns in the output. Our approach is generically applicable to any regressor mapping between two multidimensional real-valued spaces (input to output), is simple to implement, and requires no specific knowledge of the regressor's internals. We demonstrate our method for two applications using image data—a MRI T1-to-T2 generator and a MRI-to-pseudo-CT generator.

Keywords

Explainable AI · Sensitivity analysis · Medical image synthesis · Image-to-image transformation · Deep learning regression · Visual analytics

M. Espadoto (✉)
Institute of Mathematics and Statistics, University of São Paulo, Sao Paulo, Brazil
e-mail: mespadot@ime.usp.br

S. B. Martins
Federal Institute of São Paulo, Sao Paulo, Brazil
e-mail: samuel.martins@ifsp.edu.br

W. Branderhorst
University Medical Center Utrecht, Utrecht, The Netherlands
e-mail: w.j.branderhorst@umcutrecht.nl

A. Telea
Department of Information and Computing Science, Utrecht University, Utrecht, The Netherlands
e-mail: a.c.telea@uu.nl

1 Introduction

In recent years, machine learning and in particular deep learning methods have been used in increasingly many applications. However, understanding how such trained models work is challenging, especially for the case of deep learning architectures [1, 2]. In certain domains, such as medical science, it is particularly important to gain such understanding, both for increasing the confidence and interpretability of the inferred results and also for increasing their acceptance by a wider public [3, 4].

Visual analytics (VA) tools and techniques have emerged as one of the approaches of choice in the field of Explainable Artificial Intelligence (XAI) [5–7]. However, while such methods have proven to be effective in improving training and explaining how deep learning architectures work, they have addressed comparatively far less the task of explaining how trained models achieve their inference. Moreover, such VA tools have mainly focused on explaining classifiers rather than the more general regressor models.

In this paper, we aim to fill the above gaps by proposing Instance-Based Inference Explainers (IBIX). In contrast to other VA techniques, which aim to explain how a trained model treats an entire dataset, our method focuses on explaining individual instances in such a dataset, and even user-selected parts of such instances, such as parts of images. To do this, IBIX offers two operation modes that explain (a) which parts of the inferred result (output) are most strongly affected by a user-specified part of the input; and (b) which parts of the input most strongly affect a user-selected part of the output. IBIX operates generically, requiring no knowledge of the architecture, hyperparameters, or training of a deep learned model, can be applied to any n-dimensional to m-dimensional data regressor, is simple to implement and use, and is computationally scalable. We demonstrate the use of IBIX for two deep learned regressors—a MR T1-to-T2 image synthesizer and an MRI-to-CT image synthesizer.

The structure of this paper is as follows. Section 2 discusses related work in VA techniques for deep learning engineering explanation and positions our contribution in this domain. Section 3 explains our method. Section 4 presents two applications of our method related to medical image synthesis. Section 5 discusses our contributions. Finally, Sect. 6 concludes the paper.

2 Related Work

Consider a dataset $\mathcal{D} = \{(\mathbf{x}_1, \mathbf{y}_1), \ldots, (\mathbf{x}_n, \mathbf{y}_n)\}$, where $\mathbf{x}_i \in \mathbb{R}^n$ is a sample of some high-dimensional data, e.g., an image, text document, or row in a data table; and $\mathbf{y}_i \in \mathbb{R}^m$ is a value associated with $\mathbf{x}_i$. In supervised machine learning, one typically wants to construct a function $f : \mathbb{R}^n \to \mathbb{R}^m$ so that, for a training or test set $\mathcal{D}$, $f(\mathbf{x}_i) \simeq \mathbf{y}_i, \forall(\mathbf{x}_i, \mathbf{y}_i) \in \mathcal{D}$. If we replace $\mathbb{R}^m$ by a set C of categorical labels, f becomes a *classifier*. In the general case, when the codomain f is a subset of $\mathbb{R}^m$, we speak of a *regressor*.

Deep learning (DL) is one of the (supervised) methods aiming to build models f following the above pattern. While deep neural networks (DNNs) has been advancing the state-of-the-art in a variety of domains [8–10], their nature of being *black-boxes* results in a lack of interpretability concerning their learned representations and predictions (outputs) [11]. While our methodology for explaining inference, next presented in Sect. 3, can be applied equally well to any regressor f, we limit its discussion in this paper—and thus the discussion of related work next—to DL applications.

Several visual analytics (VA) solutions have been proposed [11–15] to help practitioners understand, interpret, and improve, the working of such a model. Following a recent survey on VA methods for deep learning model engineering [6], visual explanations aim to explain one of the following parts of the common deep learning pipeline: *training*, *model*, or *inference*. We review methods in all these classes next, observing already that most such methods have been designed to help with classification models [12, 14]. Using VA tools to interpret deep generative models—the proposal of this paper—has attracted only limited attention.

2.1 Explaining Training

Using visualization during the *training process* aims to explore the training data and their learned representations, to answer questions such as which classes did train suboptimally, how are classes separable in the learned feature space, and which are hard-to-process observations. We also note that most VA work we are aware of for explaining training focuses on *classifier* models.

A common visual approach to investigate a dataset is to project its learned deep representations (feature vectors) onto two dimensions [12, 15] by a dimensionality reduction technique (e.g., t-SNE [16]). One can then plot all projected data instances as points in a scatter plot and assign a different color for each class [17–19]. This approach is also used to explain the trained model (Sect. 2.2).

Rauber et al. [17] show that the visual separability of classes in a t-SNE projection is highly correlated with the ability of a classifier to separate classes in the original feature space. Consequently, the visual inspection supports understanding poor predictions in two ways: (i) a pair of classes grouped in the 2D space can indicate class imbalance or the need for more data; and (ii) all classes mixed can indicate that the learned representations are not good enough for the addressed problem. In this sense, some methods also provide visual tools to assign labels to new data examples [19–21], especially in applications in which high-quality annotated data is absent, such as medical image analysis.

Some methods investigate the examples that the model is most uncertain or unsure about [20, 22, 23]. When analyzing these so-called *hard examples*, one can have insights on the model's inference (e.g., misprediction). For example, a hard example may have been incorrectly labeled, or it may have different patterns than others in its class, or it may be an outlier. To improve the model's accuracy, one can then retrain the model, for example, by assigning a different weight for each training example [24].

One common approach to visually investigate hard examples is to retrieve the original data (e.g., images) associated to specific projected data points in a 2D scatter plot [12, 15, 18, 25]. The user may then visually inspect the original data of points from different classes which are grouped in the projected space. On the other hand, *active learning* (AL) strategies automatically search for hard examples by selecting those near the model's decision boundaries and asking the user for feedback (e.g., labels) to improve the learning model [22, 23, 26]. Bernard et al. [20] proposes a visual-interactive labeling (VIAL) that unifies both approaches to make labeling more efficient. VIAL uses AL-based methods to leverage visual interactive interfaces for the analysis, for example, presenting hard examples to the user.

2.2 Explaining the Model

This class of visual methods enables users to explore intrinsic characteristics of the learned model such as its learned parameters [13, 27, 28] and architecture [29–31]. Visualizing such information helps model developers troubleshoot and further improve their models [11, 13, 29, 32]. Explaining the model consists of answering questions such as: How do the weight patterns correlate with specific architecture layers? How do activations (for each class) look? What types of latent features are learned by specific model layers?

VA solutions visualize model architectures commonly using a *computational graph* in which nodes represent neurons and weighted edges represent connections between a pair of neurons [29–31, 33]. One can also encode the weight magnitude using color or link thickness [12]. This design is taken by TensorBoard [31], a popular VA tool that visualizes learning curves during training and displays images generated by the trained model. Wongsuphasawat et al. [30] present a design study of the network architecture visualization from TensorBoard. Drawing computational graphs does not scale well for production-size architectures having millions of links. To address this, Liu et al. [29] use a bi-clustering-based edge bundling technique to reduce visual clutter caused by too many links.

Visualizing the learned filters (weights) allows investigating what a deep model has learned for a given problem—e.g., which filters are responsible for separating a class from others [27, 33]. SUMMIT [34] analyzes activation patterns by visualizing the interaction between the learned features and the model's predictions.

Other methods aim to investigate how *neuron activations* respond to particular classes throughout the network [13, 33, 35]. ActiVis [36] represents the model architecture as a graph (nodes are operations) from which users can visualize activation patterns at each layer and for each class by an interactive table view (columns are neurons and rows are activation instances). The tool displays also a 2D projection of instance activations colored according to their classes. Rauber et al. [13] also project activations to investigate the relationships between neurons. Other techniques map neuron activations to the input pixel space to display patterns recognized by the deep model [33, 35, 37].

Several techniques aim to explain what is the role of each network layer in the model inference [14]. Using such techniques, one could find that, in deep learning images, lower layers create representations of simple features (e.g., edges) while higher layers contain specific information about classes [38–40]. Other VA tools support finding stable layers—that learned a stable set of patterns—and layers that do not contribute to solving a given classification problem [28].

A few VA tools have aimed to explain generative adversarial networks (GANs) by exploring their internal structures. Gan Lab [41] is an interactive tool designed for non-experts to learn and experiment with GAN models. DGMTracker [42] and GANViz [43] aim to explain the training dynamics of GANs, e.g., by visualizing their neural activations, to help developers better train the models.

2.3 Explaining the Inference

The third and final class of VA methods, to which our proposal also belongs, aims to explain how outputs $f(\mathbf{x})$ of a trained model f depend on the input instances $\mathbf{x}$.

Saliency maps [27, 39, 44–47] are likely the most used and best known visual tool for inference explanation. For models whose inputs $\mathbf{x}$ are images, they mark each pixel $\mathbf{p} \in \mathbf{x}$ with a value indicating $\mathbf{p}$'s contribution, or influence, to the decision $f(\mathbf{x})$.

Also for image-processing networks, Zeiler and Fergus [27] used *deconvolutional networks*, as proposed in [35], to project learned feature activations to the input image space. This allows users to debug the deep model by visualizing the learned features from specific layers, with multiple variations of the technique being proposed afterwards [12]. Zhou et al. [44] propose Class Activation Mapping (CAM), a technique that shows the discriminative active region in an image for a given label. Selvaraju et al. [45] presented its relaxed generalization, Grad-CAM, which uses label-specific gradients to calculate the importance of spatial locations in convolutional layers.

All methods so far presented generate visual explanations based on components of the learned DNNs, such as their architectures and activations. Despite presenting impressive results for many problems [12], these visual methods are designed for a restricted class of DNNs. In contrast, a different approach, referred as *reverse engineering* [48], only uses the input $\mathbf{x}$ and inference $f(\mathbf{x})$ of the deep model without exploiting any model internals. For a learned deep model, this approach applies a *random perturbation* to the input and compares its inference with the unperturbed one [48]. One can then create visual explanations, e.g., a heatmap, from this comparison. Note that this approach is independent of the kind of DNN.

Bazzani et al. [49] use the reverse engineering approach for weakly supervised object detection. Given a pre-trained deep model designed for image classification, their method analyzes the degeneration in classification scores when artificially perturbing different regions of the image by masking them out. The masked regions that significantly drop the classification scores are considered as including the target objects. Other object detection methods use a similar strategy [50, 51].

Our proposed framework follows the reverse engineering approach when creating a heatmap from comparing the original inference and the perturbed one. This heatmap shows which parts of the inference—e.g., a reconstructed image by a generative neural network—have been influenced by input variables selected by the user and vice versa. This allows for a fine-grained study of how specific sets of output variables are affected by perturbations to input variables.

A distinctive attribute of our framework is that it enables the study of black-box models having continuous multivariate *inputs and outputs*, such as autoencoders and GANs. This is in stark contrast with most existing techniques described earlier, that either seek to explain single-output classification models, or require the use of internal structures of the network to derive an explanation of the model. This makes our framework particularly suitable for understanding models created for image transformation, for example, but not limited to those. Our framework can work with different types of models, regardless of their internal structure, as long as they have n inputs and m outputs, both real-valued, as detailed next.

3 IBIX Method

3.1 Definitions

Let $\mathbf{x} \in \mathbb{R}^n$ be an input sample, such as a n-dimensional feature vector or a grayscale image having n pixels. Let $f : \mathbb{R}^n \to \mathbb{R}^m$ be the learned model by a deep neural network. Note that the output space (and its dimensionality m) need not be identical to the input space (and its dimensionality n). We next denote $\mathbf{x} = (x_1, \ldots, x_n)$, i.e., $x_i \in \mathbb{R}$ is the i^{th} component of the n-dimensional vector $\mathbf{x}$. Similarly, if $f(\mathbf{x}) = (y_1, \ldots, y_m)$, then let $f_i : \mathbb{R}^n \to \mathbb{R}$, $f_i(\mathbf{x}) = y_i$, be the i^{th} component of the function f. Note that f_i is a real-valued function with n variables.

Let $M^{\mathbf{x}}$ be a region in $\mathbf{x}$, i.e., a subset of components of $\mathbf{x}$ that we are next interested to analyze. For example, if $\mathbf{x}$ is a 2D image, then $M^{\mathbf{x}}$ is a mask that we draw on the image to select some of its pixels. Formally, $M^{\mathbf{x}}$ can be modeled as an indicator with ones for the selected variables (pixels) x_i and zero elsewhere. That is, $M^{\mathbf{x}} \in \{0, 1\}^n$. Hence, $M = (M_1^{\mathbf{x}}, \ldots M_n^{\mathbf{x}})$ so that $M_i^{\mathbf{x}} = 1$ if variable x_i is selected and zero otherwise. Similarly, for the output, let M^f be a region in $f(I)$. Intuitively, M^f allows us to mark components of $f(\mathbf{x})$ that we want to 'trace back' to the input $\mathbf{x}$. Just as $M^{\mathbf{x}}$, $M^f \in \{0, 1\}^m$ can be modeled as an indicator. That is, $M^f = (M_1^f, \ldots M_m^f)$ so that M_i^f is one if output component $f_i(\mathbf{x})$ is selected for analysis and zero otherwise.

3.2 Forward Mapping

As outlined in Sect. 1, the first goal of out IBIX method is to visually explain how much specific parts of the input $\mathbf{x}$—more precisely, those marked by the user in a mask $M^{\mathbf{x}}$—affect the output $f(\mathbf{x})$. We call this a *forward mapping* and denote it as $F(M^{\mathbf{x}})$. Formally put, $F(M^{\mathbf{x}}) = (F_1, \ldots, F_m)$ with $F_j \in [0, 1]$, $1 \leq j \leq m$. That is, $F(M^{\mathbf{x}})$ is a weight vector, with one value F_j per output dimension.

We compute $F(M^{\mathbf{x}})$ by perturbing, or jittering, the marked part $M^{\mathbf{x}}$ of the input sample $\mathbf{x}$, passing the perturbed data through f, and seeing how much $f(\mathbf{x})$ has changed. The intuition behind this idea is simple: If changing the marked area of $\mathbf{x}$ does not affect the inferred value $f(\mathbf{x})$, then the respective input part can be seen as neglected by the regressor. Conversely, if a small change to the marked area strongly affects $f(\mathbf{x})$, then the regressor has somehow learned to be very sensitive to the respective input part. When the two above situations occur, it is the user who has to decide if neglect or high-sensitivity are desirable behavior or not for the regressor, depending on the actual location of $M^{\mathbf{x}}$ and variation of $F(M^{\mathbf{x}})$.

Computing $F(M^{\mathbf{x}})$ consists of two steps, as follows.

Single perturbation: Consider a (small) jitter value $h \in \mathbb{R}$. Let $\Delta\mathbf{x} = hM^{\mathbf{x}}$, that is, a vector which is zero outside the region $M^{\mathbf{x}}$ and equal to h inside $M^{\mathbf{x}}$, respectively. With it, we compute $f(\mathbf{x} + \Delta\mathbf{x})$, i.e., the model's response to the input $\mathbf{x}$ jittered by $\Delta\mathbf{x}$, normalized by the change size. We denote this normalized change by a vector $F^h(M^{\mathbf{x}}) = (F_1^h, \ldots, F_m^h)$, where

$$F_j^h = \frac{f_j(\mathbf{x} + \Delta\mathbf{x}) - f_j(\mathbf{x})}{h}, \quad 1 \leq j \leq m. \tag{1}$$

If h is small, F_j^h is the sum of the components of the forward finite-difference-approximated gradient of f_j that considers only the variables selected by $M^{\mathbf{x}}$. This is analogous to taking the derivative of f_j in the direction given by the n-dimensional unit vector corresponding to the ones in $M^{\mathbf{x}}$, i.e.

$$F_j^h \simeq \frac{\partial f_j}{\partial M^{\mathbf{x}}}. \tag{2}$$

As the directional derivative is linked to the gradient of a function by the dot product

$$\frac{\partial f_j}{\partial M^{\mathbf{x}}} = \nabla f_j \cdot M^{\mathbf{x}}, \tag{3}$$

it follows that

$$F_j^h \simeq \sum_{1 \le i \le n | M_i^{\mathbf{x}} = 1} \frac{\partial f_j}{\partial x_i}. \tag{4}$$

where $\frac{\partial f_j}{\partial x_i}$ is the partial derivative of f_j with respect to the variable x_i.

Multiscale perturbation: To eliminate the effect of the choice of the jitter size h, we evaluate Eq. 1 for a N zero-centered, uniformly-spaced, jitters $h_k = kH/N$, with $-N \le k \le N$, where H is an application-dependent parameter specifying the maximum jitter, set typically to 10 to 20% of the norm of the input signal $\mathbf{x}$. The final forward mapping is then computed as

$$F(M^{\mathbf{x}}) = \frac{1}{2N} \sum_{-N \le k \le N} F^{h_k}(M^{\mathbf{x}}), \tag{5}$$

that is, the average of the responses for all perturbations h_k. Note that, conceptually, Eq. 5 is equivalent to computing a scale-space version of the directional derivative in Eq. 2. Intuitively, $F(M^{\mathbf{x}})$ will be large for output components of f which are strongly affected by changes in input variables selected in $M^{\mathbf{x}}$, and conversely.

3.3 Backward Mapping

The second goal of our IBIX method is to visually explain how much all variables in $\mathbf{x}$ affect a part of $f(\mathbf{x})$ that is selected by some mask M^f. By analogy to the forward mapping $F(M^{\mathbf{x}})$ in Sect. 3.2, we call this the *backward mapping* and denote it by $B(M^f)$. Formally put, $B(M^f) = (B_1, \ldots, B_n)$ with $B_j \in [0, 1]$, $1 \le j \le n$. That is, $B(M^f)$ is a weight vector, with one value B_j per input variable.

Unlike $F(M^{\mathbf{x}})$, we cannot compute $B(M^f)$ directly since we do not have the inverse function f^{-1} of our deep learned model. Hence, we proceed differently: We partition the input space of n variables into a set of K block regions D_k, $1 \le k \le K$. Intuitively, if $\mathbf{x}$ is an image, the blocks D_k can be seen as a tessellation of $\mathbf{x}$ into so-called superpixels. Each block D_k acts as a region mask $M^{\mathbf{x}}$ for the input $\mathbf{x}$. Next, we compute for each block D_k the forward mapping $F(D_k)$ using Eq. 5. Subsequently, we define the backward mapping from the mask M^f in the output space to block D_k in the input space, denoted as B_{D_k}, as the fraction of the integral of the forward mapping $F(B_k)$ that falls within M_f, i.e.,

$$B_{D_k} = \frac{\sum_{1 \le i \le m | M_i^f = 1} F(D_k)_i}{\sum_{1 \le i \le m} F(D_k)_i}, \quad 1 \le k \le K. \tag{6}$$

Note that, if the blocks D_k are of unit size, i.e., the input space is partitioned into $K = n$ blocks, one per input variable x_k, and we consider a single scale h in Eq. 5, then $F(D_k)_i = \frac{\partial f_i}{\partial x_k}$. Then, for x_k, we get the backward mapping expression as

$$B_k = \sum_{1 \le i \le m | M_i^f = 1} \frac{\partial f_i}{\partial x_k}. \tag{7}$$

That is, the value of the inverse mapping B at input variable k is the sum of all partial derivatives of f with respect to x_k for all components that are marked one in the mask M^f.

The forward mapping (Eq. 4) and the backward mapping (Eq. 7) have similar expressions—both are sums of partial derivatives, the difference being the indices that vary and the ones that are fixed. However, evaluating the backward mapping is more costly, since, in Eqs. 6 and 7, we sum over all dimensions i selected in the output-mask M^f. For each such dimension, we need to evaluate the full forward mapping F (Eq. 6) or, if we use the notation in Eq. 7, a partial derivative. The problem is that a typical DL model implementation does not let one 'selectively' evaluate a single output component f_i; we need to evaluate *all* the m output components even if some fall outside the mask M^f. In contrast, for the forward mapping (Eq. 4), we sum over all input variables marked as one in the input mask $M^{\mathbf{x}}$. This can be done very efficiently simply by changing the respective inputs of the neural network.

Following the above, computing the backward mapping is K times more expensive than computing the forward mapping, where K is the number of blocks used to represent the input space. Using fewer blocks (low K) accelerates computing this mapping but creates a low resolution understanding of how input variables affect the output region M^f—all variables in a block are seen as 'acting together' to influence the output. Conversely, using more blocks is slower, but gives a fine-grained understanding of how output dimensions in M^f depend on input variables—in the limit, for $K = n$, we see how how every single variable of $\mathbf{x}$ contributes to outputs in M^f. We discuss efficient ways to trade off computational speed *vs* insight resolution further in Sect. 4.1.2.

4 Explainer Applications

Our IBIX framework (Sect. 3) can be used to explain any $\mathbb{R}^n$ to $\mathbb{R}^m$ regressor, whether implemented by deep learning or not. The required adaptations for this are (1) defining ways to select the regions of interest $M^{\mathbf{x}}$ and M^f; (2) defining the jitter range H (Sect. 3.2); and (3) suitably visualizing the direct and inverse mappings F and B. We next illustrate IBIX on different deep learning applications: two image-to-image regressors for medical data (Sects. 4.1 and 4.2).

4.1 Explaining Autoencoders

We considered the generation of MR-T2 brain images (Fig. 1b) from MR-T1 brain images (Fig. 1a) using convolutional autoencoders (CAEs) [52]. This use-case is of interest when one wants to simulate the effect of a T2 scan but only avails of T1 scans as input data.

Figure 1a presents the CAE architecture we used, having three 2D convolutional layers with 16, 8, and 8 filters of 3×3 weights each, followed by ReLU activation [53] and 2D max-pooling in the encoder. The decoder contains the corresponding reconstruction operations. The model is trained to minimize mean squared error (MSE) between the generated and target T2 images using the *nadam* gradient optimizer [54].

We trained the CAE using the CamCan public dataset [55], which has 653 pairs of 3D MR-T1 brain images of 3 Tesla from healthy men and women between 18 and 88 years. For each 3D MR-T1 image, CamCan also has a corresponding 3D MR-T2 image. To our knowledge, CamCan is the largest public dataset with 3D images of healthy subjects acquired from different scanners.

We applied typical MRI noise reduction and bias field correction to all MR-T1 and MR-T2 images. Next, we registered the images to the same MNI template [56]. Since the considered CAE only supports 2D images, we extracted the central 2D axial slice from all 3D images to build our final training set (Fig. 1b, c). Each training instance is therefore an 8-bit grayscale 2D image: pixels' intensities within $[0, 255]$. Training the CAE reached mean squared errors around 0.0052 in the training set after 500 epochs with a batch size of 32. The trained model and preprocessed data are available online for replication purposes (https://github.com/hisamuka/IBIX-CAE).

4.1.1 Visual Explanation of CAE

We next used IBIX to explain the CAE MR-T1 to MR-T2 autoencoder. In this case, both inputs and outputs of the CAE function f are grayscale images, both of $m = n = 232 \times 200$ pixels. Hence, the masks $M^{\mathbf{x}}$ and M^f are binary images of the same size. To view and manipulate such images, we designed the user interface (Fig. 2) which is based on the *napari* image viewer [57]. The tool allows users to select an input MR-T1 image $\mathbf{x}$, run it through the trained CAE f, display the output MR-T2 $f(\mathbf{x})$, and, most importantly, paint regions $M^{\mathbf{x}}$ (in the input), respectively M^f (in the output), and next compute and visualize the forward and backward mappings F and B as heatmaps.

Figure 3 shows how IBIX works for the CAE problem. Images (a1) and (a2) show an MR-T1 input $\mathbf{x}$ and its CAE-synthesized MR-T2 output $f(\mathbf{x})$, respectively. In (b1), the user selected a single pixel region $M^{\mathbf{x}}$ in the input (marked red, see also inset). Image (b2) shows the *forward mapping* F of this single pixel using a heat colormap: Warm regions are output pixels which strongly change upon small changes of the (red) input pixel.

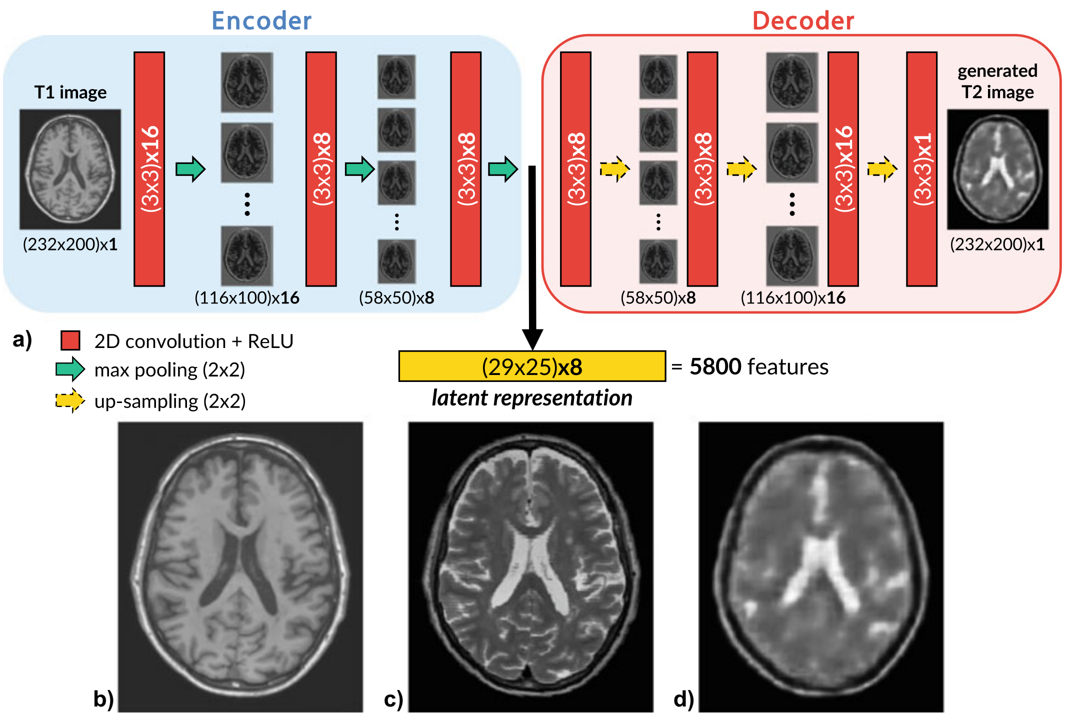
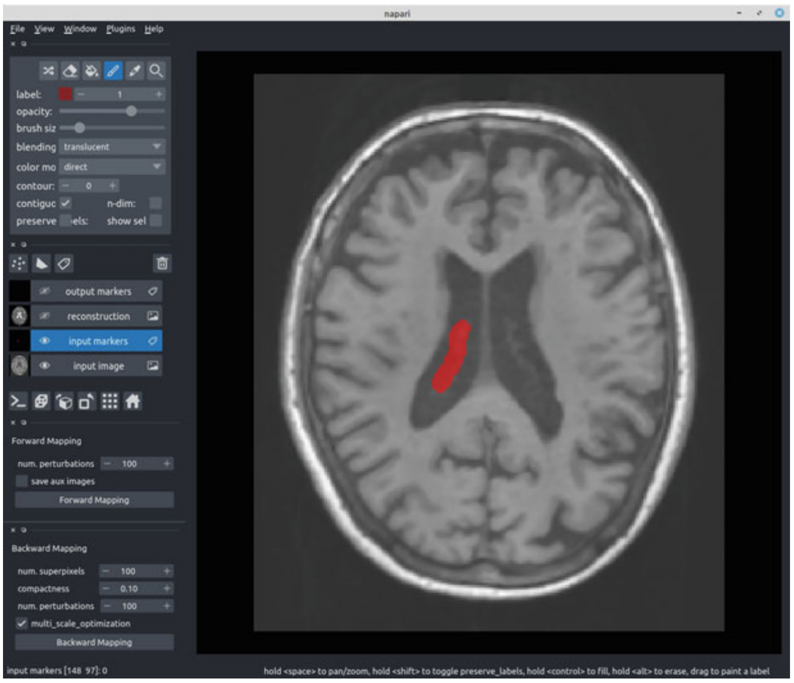


Fig. 1 **a** Architecture of the MR-T1 to MR-T2 convolutional autoencoder. The autoencoder is trained to generate the target MR-T2 brain image (**b**) from the input MR-T1 brain image (**a**). Output generated MR-T2 image shown in (**c**). See Sect. 4.1

Fig. 2 User interface for the IBIX explainer with user-marked region in red



label: 1
opacity:
brush siz
blending translucent
color mo direct
contour: 0
contiguc ✓ n-dim:
preserve labels: show sel

Forward Mapping
num. perturbations 100
save aux images

Backward Mapping
num. superpixels 100
compactness 0.10
num. perturbations 100
✓ multi_scale_optimization

input markers [148 97]: 0

hold <space> to pan/zoom, hold <shift> to toggle preserve_labels, hold <control> to fill, hold <alt> to erase, drag to paint a label

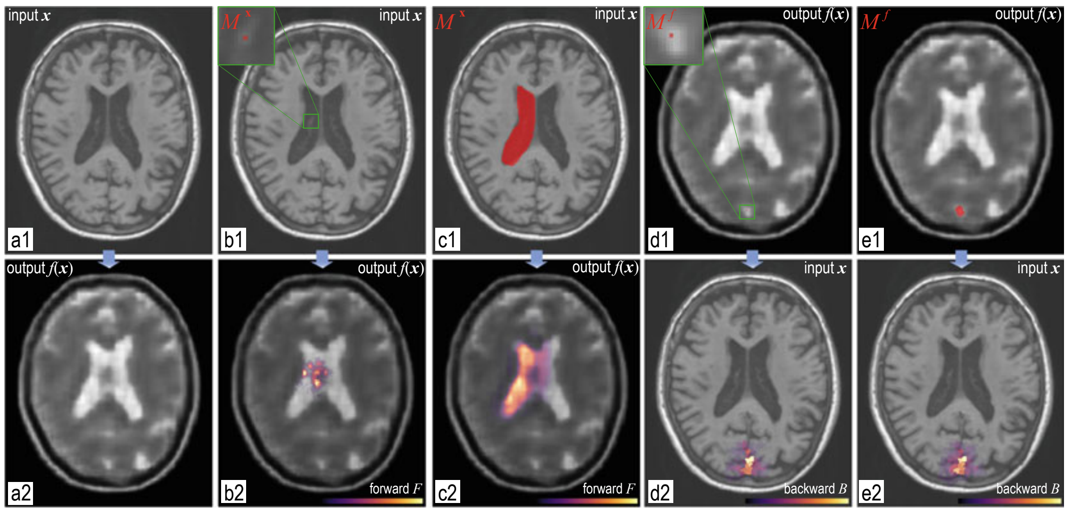


Fig. 3 Images (a1) and (a2) show an MR-T1 input and its CAE-synthesized MR-T2 output image, respectively. Images (b–c) and (d–e) show next the CAE forward, respectively backward, mappings (Sect. 4.1.1)

We see that these are close to the location of the red input—which is desired, since the MR-T1 to MR-T2 mapping should be *spatially coherent*. That is, a region $\mathbf{x}$ in the MR-T1 input is supposed to influence only *close* regions in the MR-T2 output. However, the forward mapping F (image b2) shows a non-linear 'response' shape to the single-selected input pixel in (b1) consisting of roughly six closely-packed peaks. This indicates some potential problems of the CAE training. Image (c1) shows a more complex input selection $M^{\mathbf{x}}$ consisting of the left ventricle. Image (c2) shows that this input region affects mostly left-ventricle pixels in the output, albeit with a limited 'leak' to the right ventricle. This is definitely desirable, since large-scale structures such as the ventricle are not supposed to appear fundamentally differently in MR-T1 and MR-T2 images. For both forward mapping examples, we considered 100 zero-centered, uniformly-spaced, jitters within $[-100, 100]$; that is, $N = H = 100$ for multiscale perturbation (Sect. 3.2).

Image (d1) shows the *backward mapping*: Here, we selected a single pixel (red, M^f) in the MR-T2 output. Image (d2) shows the regions in the corresponding MR-T1 input, as defined by *superpixels*, that strongly influenced the selected output region. As desired, these regions are located close to and around the selected pixel. Image (e1) extends this test by selecting a larger output region. In image (e2) we see that the backward mapping highlights input pixels close to and around the selected structure, which is desirable. In both examples, we used the popular SLIC algorithm [58] for superpixel segmentation due to its robustness and simplicity. We extracted $K = 500$ superpixels for evaluation with compactness value of 0.1. These numbers guarantee reasonable small-scale superpixels—which are desirable for a fine-grained understanding (Sect. 3.3)—but demands considerable processing times. We considered the same 100 jitters used for forwarding mapping.

Summarizing the use of IBIX for this example: Ideally, we want both the forward (F) and backward (B) mappings to be *localized*, i.e., when selecting a region in one of the (input or output) spaces, we see that a similar-location-and-shape region is responsible for that. If not, the CAE would have learned to 'couple' anatomically unrelated regions, which is clearly undesirable. Still, images (b1-b2) show that the CAE exhibits a certain amount of *diffusion*—small-scale structures can have a relatively strong effect at a certain distance from them in the output.

We tested the speed of IBIX on an AMD Ryzen 7 3700X 8-Core PC with 16 GB RAM with an NVIDIA Titan XP 12 GB GPU. Performing a for-

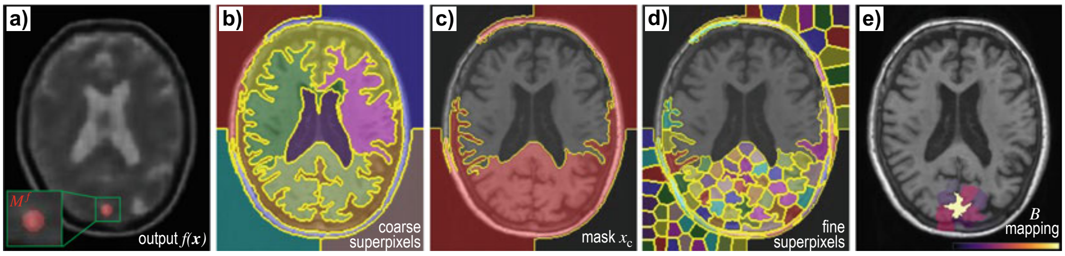


Fig. 4 Multiscale CAE optimization. From markers (M^f) drawn on the output (**a**), we first perform backward mapping on a coarse scale using just a few superpixels (**b**). We next locate superpixels having high B values (**c**) and refine the computation by segmenting only these on a finer-scale (**d**). Figure (**e**) shows the final backward mapping

ward mapping is *fast*, taking about 0.33 sec regardless the input selection size (i.e., the number of painted pixels in $M^{\mathbf{x}}$). Using $K = 500$ superpixels, performing a backward mapping takes about 174 seconds, roughly 500 times more than forward mapping (see also Sect. 3.3). This high processing time makes an interactive user experience unfeasible. When parallelizing the backward mapping—i.e., running its K forward mappings (500 in our case) in parallel (see Sect. 3.3)—computing is nearly halved: 97 s. Section 4.1.2 presents another optimization strategy to further speed up backward mapping.

4.1.2 Optimizing Backward Mapping

The standard superpixel segmentation method we use [58] allows one to control the size of superpixels but typically produces similar-size superpixels for an entire image. Hence, to get a high resolution of the backward mapping, we need to segment the input image $\mathbf{x}$ in a high number of superpixels, e.g., $K = 500$, each of which is next forward-mapped, yielding an overall slow method, as mentioned in Sect. 4.1.1. We observe that several of these superpixels are far from the markers M^f or even out of the brain. We also observe that, in general, the backward mapping is *localized*, i.e., B has high values over $\mathbf{x}$ only over *small* image extents, which are also typically close to M^f.

We use the above observations to accelerate the backward mapping B computation by a multiscale strategy, as follows. Consider Fig. 4, where image (a) shows the region M^f marked in the output. We first compute the backward mapping B using a coarse segmentation of the input $\mathbf{x}$ into a few superpixels $K_c \ll K$, where K is the number of fine-scale superpixels deemed small enough by the user for a good resolution. In our example, we use $K_c = 10$ coarse-scale superpixels, shown in Fig. 4b. We next compute B on these K_s superpixels as outlined in Sect. 3.3. Let $\mathbf{x}^c$ be the subset of the image $\mathbf{x}$ covered by coarse-scale superpixels having a B value over a user-specified threshold (Fig. 4c, red area). We next segment $\mathbf{x}_c$ into K_f fine-scale superpixels (Fig. 4d) and use these to compute the final backward mapping (Fig. 4e).

Several remarks are due, as follows. The total processing time of this multiscale strategy depends on the total superpixel count $K_c + K_f$ used for the coarse, respective fine, scales. Note that $K_f < K$ where K would be the number of superpixels used by the single-scale strategy (Sect. 3.3), since only a *subset* of the input $\mathbf{x}$ is segmented on the fine scale—red area in Fig. 4. In the example in Fig. 4, the fine-scale superpixels are roughly of the same size as the $K = 500$ superpixels needed to cover the entire image with the single-scale strategy. However, $K_f = 100$ and $K_c = 10$, so we have only 110 superpixels to treat instead of the 500 ones in the single-scale strategy. Using parallelization of the forward mapping (Sect. 3.3), the multiscale computation scheme needs only 24 seconds instead of 174 seconds (single-scale, sequential) or 94 seconds (single-scale, parallelized).

4.2 Explaining MRI-to-CT Generators

Besides MR-to-MR image generators (Sect. 4.1), medical imaging scientists have also been concerned with generating synthetic CT images from

MRI scans [59, 60]. This is useful e.g. in the context of MR-guided radiotherapy where one needs to examine the anatomy (typically best seen in a CT scan) for online position verification and dose planning of the radiotherapy [61]. Such applications are an important beneficiary of explainable AI (XAI) methods such as ours [7].

For MRI-to-CT generation for pelvis scans, Maspero et al. [62] have recently shown good results using Generative Adversarial Networks (GANs). GANs are a class of generative models that train by framing the problem as a supervised learning one with two sub-models: A *generator* model is trained to generate new examples; a *discriminator* model tries to classify examples as either real (from the domain) or fake (generated) ones. The two models are trained together in a zero-sum (adversarial) game until the discriminator model is fooled about half the time, meaning that the generator model can create plausible examples. For their task, Maspero et al. have used the Pix2Pix model [63], which is a GAN originally proposed for transferring image styles between two different domains, e.g., real picture to cartoons or satellite maps to blueprint maps.

In our work, we used a similar model to Maspero et al. to synthesize CT images from the head-and-neck and pelvis regions, as follows. The input image $\mathbf{x}$ is a set of 3 transaxial 2D image slices (480^2 pixels) obtained by taking the water, fat and in-phase images from a T2 TSE mDixon MRI sequence. Similar to [62], we did not use the fourth channel (out-phase) in this study. From each slice, a 256^2 pixel sub-image was extracted at a random location, clipped to the range between 0 and the 95% percentile, and then normalized to $[-1, 1]$. These images are further used for training our network. Figure 5a, c show two examples of such images. The scans are registered using Elastix [64, 65] to the ground-truth (GT), which are CT scans of the same patients. CT values are clipped to the $[-1024, 1250]$ HU range and then normalized to $[-1, 1]$. Figure 5b, d show two examples corresponding to the MRI scans in images (a,c). Two separate models are trained for the head-and-neck (60 scans) and pelvis (13 scans) regions, respectively. The *generator* model uses a U-NET architecture [66] using, for the encoder, 8 convolutional layers with 64, 128, 256, and 512 (last 5 layers) filters, each being a 4×4 filter applied with stride 2, and downsampling factor of 2; and for the decoder 8 convolutional layers with 512 (first 5 layers), 256, 128, and 64 filters and corresponding upsampling parameters to the encoder. The model is trained with L1 loss. The *discriminator* uses the Markov PatchGAN [63] that only penalizes structure at the scale of $N \times N$ pixel patches, with $N = 70$ pixels. As in Pix2Pix, the discriminator is run convolutionally across patches over the entire image, averaging all local responses to provide the final output, i.e., whether the generator creates real or fake images. Convolutions are 4×4 spatial filters applied with a stride of 2 and downsample factor of 2.

The above GAN achieves good results—a mean average error (MAE) between the predicted and ground-truth CT of 271.22 HU for air (< -200 HU), 56.67 HU for soft tissue ($-200 \ldots +200$ HU) and 311.74 HU for bone ($> +200$ HU) and a mean structural similarity index (SSIM [67]) between the two images of 0.89. However, subtle errors occur in the prediction. Figure 5e, h show two ground-truth CT scans of the head-and-neck region, with corresponding predicted images in (f, i) and ground-truth-*vs*-prediction errors color-coded in images (g, j)—white indicates no difference; blue indicates predicted value lower than GT value; and red indicates predicted value higher than GT value, respectively. Soft-tissue regions are, overall, predicted well. Yet, we see some ‘bone loss’ (blue arrows). We also see some ‘fake bone’ tissues being created by the prediction (red arrow A, image (i)) as well as small-scale cavities being filled up with tissue (other three red arrows, image (i)). Apart from that, we see a more general smoothing (or loss) of small-scale details.

Although we experimented with various ways of tuning of the GAN to decrease such artifacts, including hyperparameter grid search, we could not consistently eliminate them. As such, obtaining insights how the output (CT) structures depend on the input ones and, more importantly, on the

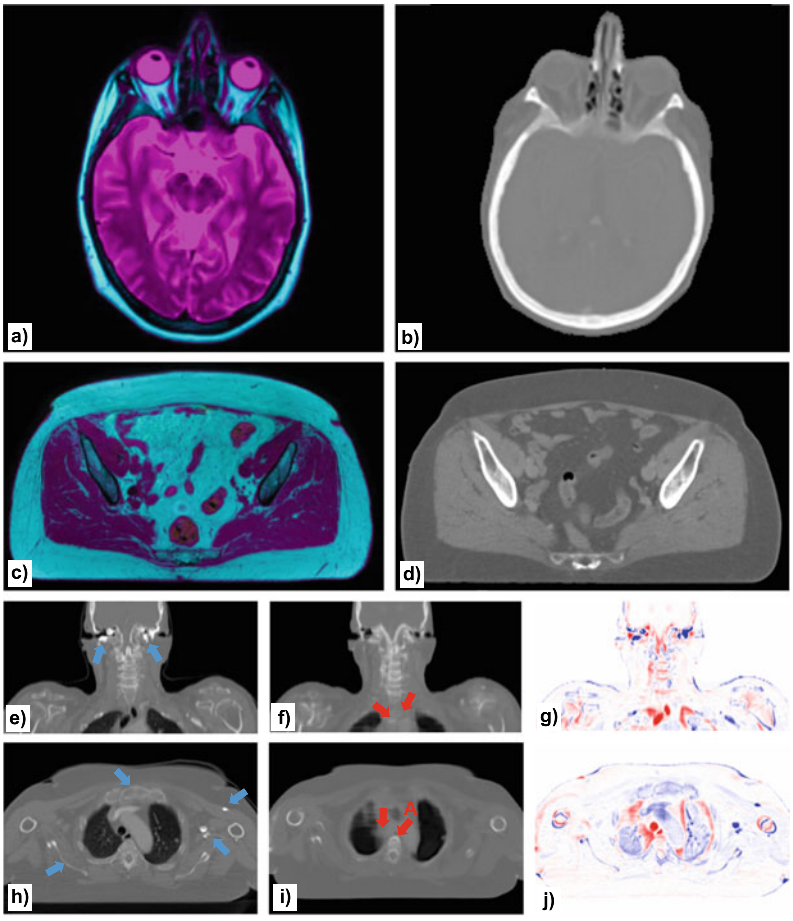


Fig. 5 Training data for the MRI to CT generation. **a**, **c** MRI 3-channel scans (water, fat, in phase) coded as RGB images; **b**, **d** CT scans of the same patients. **e**, **h** True CT scans with **f**, **i** synthetic CT reconstructions and **g**, **j** differences between the two (white = no difference; blue = pseudo-CT lower than true CT, see also blue arrows in (**e**, **h**); red = pseudo-CT higher than true CT, see also red arrows in (**f**, **i**). See Sect. 4.2

actual underlying anatomical details, is an important step to further tuning the prediction. For this, we use IBIX (see next Fig. 6).

We proceed as follows. Since the prediction errors are *small-scale* structures, we only select a few pixels in $M^{\mathbf{x}}$, respectively M^{f}. Also, we repeat the selection for close spatial locations in the input, respectively output, e.g., images (a–c) and (d–f). By comparing the obtained mappings F and B, we can better understand how the model learned the inference for such structures. For the forward mapping F, e.g., images (a–c), we show the region in the MRI input around the selection $M^{\mathbf{x}}$ as a small inset top-right in the respective images. The main image shows the output CT scan, overlaid by F, color-coded by a heatmap. For clarity, we also show F in the top-left inset. For the backward mapping, e.g., images (d–f), we use the same selection as in the corresponding forward mapping, i.e., $M^{f} = M^{\mathbf{x}}$, so we do not need to show this selection again. The main image shows the input MRI scan, overlaid by B, color-coded by a blue-to-yellow colormap. For clarity, we also show B in the top-left inset.

We next examine four different situations observed during the CT prediction, as follows.

Well-predicted bone: For this case, we want to understand how the model proceeded when achieving good prediction. Images (a-c) show three closely located selected pixel areas (yellow in the top-right insets) inside a vertebra structure, the latter seen as dark blue in the MR images in the insets. This structure is quite well predicted visible as the V-shaped light-gray area in the predicted images. The first (a) and last (c) selected areas are smaller than the middle one (b). We see that the forward mappings F match very well the expected shape of the bone—the hot-colored areas do not 'leak' out of the light-gray area, meaning that the selected bone pixels are used, indeed, only to predict bone in the same structure. Also, we see that the middle mapping F (image (b)) has a larger hot-spot than the other two. This is expected, since its selection—yellow in image (b)—is larger and more intense. If we look at the inverse mappings B for the same selections, we see a few bright-colored (yellow) superpixels in images (d–f). These are also quite closely located to the selected pixels. Hence, the predicted bone pixels are caused mainly by bone pixels in the same structure in the input MRI. In other words, the prediction is *localized* and follows the expected bone anatomy.

Poorly-predicted bone: As shown in Fig. 5e, h, some small-scale bone structures in the GT are missed by the model. To explain why this is the case, we select three pixel zones close to such a bone structure, visible as the dark ring in the MRI insets in Fig. 6g–i. Again, the middle selection (h) is larger than the other two. The forward mappings in images (g–i) show heatmaps that are located close to the ring structure, but do not closely follow its shape, being rather blurry. In all three maps, the region inside the ring is also marked by the heatmaps as being predicted by the small (yellow) selected areas which are on the bone proper. Hence, the model 'blurs out' the small-scale bone information. As a result, the bone itself is not visible in the output CTs. Again, the mapping for the larger selection (h) is stronger than the other two. This is an expected effect, since a larger selected input zone will affect a larger zone in the output. The backward mappings (images j–l) show a similar effect—the selected output pixels are affected by the entire area around the selected zone—that is, both by the elements marked dark in the MRI insets in (g–i) but also surrounding, brighter, pixels. Since the bone structure there is very thin, blurring occurs, i.e., the model 'averages' the bone with the surrounding softer tissues in its prediction. In other words, both the forward and backward mappings show that the trained model apparently understands that the pixels inside the ring pattern belong to the same structure, but it does not apply the same intensity value as in the well predicted bone. We conclude that the network looks at local structure, and could be improved if it would be trained to use information from other similar bones in a different and/or more distant location.

Well-predicted cavities: As shown in Fig. 5e–i, air-filled cavities inside the tissue—black in those images—are well predicted. It is interesting to examine this further. Images (m–o) show such a cavity in the MRI input (insets) in which we, again, selected three pixel areas with the middle one (n) larger than the other two. The forward mappings show heatmaps which are very high close to the selected pixels (central pink dot in the respective heatmaps) but also contain a 'ring' of high F values close to the air-tissue interface, i.e., where the black hole touches the surrounding gray pixels. This means that the model used the selected air pixels to predict both air pixels but also the *borders* of the entire air cavity. Interestingly, the heatmaps are black (zero) in the cavity outside the selected pixels themselves. By definition of F (Sect. 3.2), this means that small changes in the air values in the input MRI will not affect the prediction of air in the output CT. This is a desirable result as it shows that the model is resistant to noise present in the input in low-HU areas. In other words, if the network had been sensitive to small-scale variations of the acquired intensity in low-HU areas, it would have had a hard time predicting the air cavity as all being the same tissue type—air, that is. However, our forward mapping show that this was not the case since the perturbations IBIX applies only affect a subset of the local pixels *inside* the cavity and the homogeneous HU value of air was apparently not due to deviating noisy pixels being constrained by the prediction of other pixels in the cavity. The backward mappings (p,r) show a similar insight: In the insets, we see a value slightly higher than the surroundings in for the cavity, visible as the whitish-light-blue color surrounded by dark blue. This shows (1) that predicted CT cavity correctly only depends on the actual cavity recorded in the MRI data and (2) this prediction is robust to noise. Indeed, by definition of the backward mapping (Sect. 3.3), a low value of B indicates that the output will not change much when the input changes slightly.

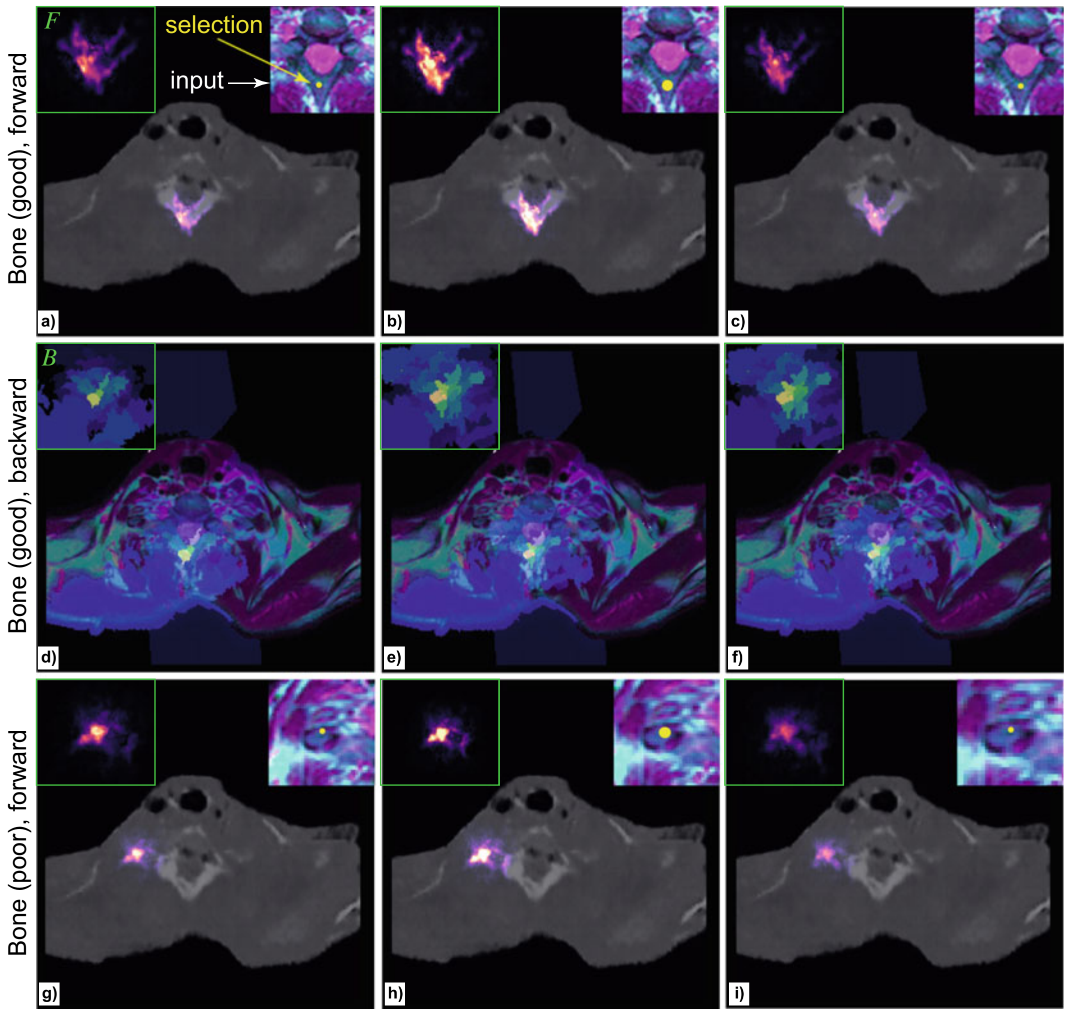


Fig. 6 Explaining MRI-to-CT generation. Forward (**a**–**c**, **g**–**i**, **m**–**o**) and backward (**d**–**f**, **j**–**l**, **p**–**r**) mappings for a well-predicted bone (**a**–**f**), poorly predicted bone (**g**–**l**), and well-predicted air cavity (**m**–**r**). Mappings are overlaid over the respective input or output images. Top-right insets show the area in the input MRI with selected pixels in yellow. Top-left insets show the mappings without overlay. See Sect. 4.2

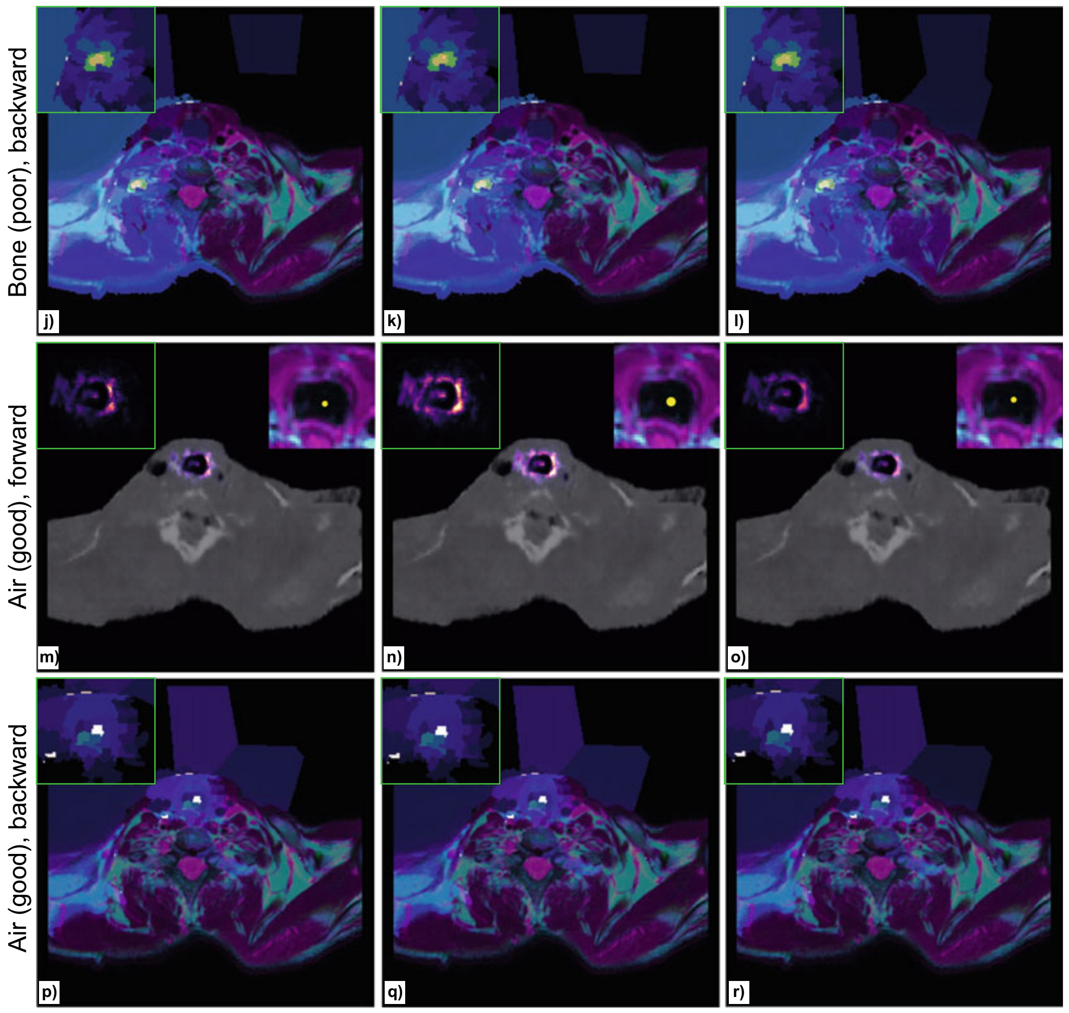


Fig. 6 (continued)

5 Discussion

We discuss next several aspects of our proposed IBIX framework.

Genericity: By construction, IBIX can handle any types of mappings f, as long as these input and output real-valued quantities. While we demonstrated IBIX only for *regressors* (which, as explained in Sect. 2, are the more complex and less covered case in the literature), our framework can handle any mapping, provided that one defines (1) ranges for the input perturbations and (2) suitable visualizations for the induced output changes. This is in stark contrast to most VA methods for XAI which work only for specific input and/or output data types [6, 48]. In the same time, IBIX is *fully* black box-compatible, needing only the ability to evaluate the model f for some given input $\mathbf{x}$, in stark contrast with many XAI techniques that need more knowledge over f [2, 48].

Ease of use: IBIX is fully automatic, requiring the use only to select a region of interest in the input ($M^{\mathbf{x}}$) or output (M^f) to explain these. The actual selection mechanism, of course, depends on the kind of input (and/or output) data.

Speed: IBIX's speed is fundamentally determined by the speed of evaluating the underlying model f.

For the forward mapping F, IBIX's cost equals the inference cost of f times the number H of jitters (Sect. 3.2). For the backward mapping B, this cost increases by a factor of K, equal to the number of blocks used to discretize the input domain. This cost can be however spread over multiple scales (Sect. 4.1.2) to generate high-resolution mappings in areas where the signal is high, thus, of interest to the user. All in all, for typical DL pipelines, F runs at interactive rates for inputs (and outputs) of dimensionality (n, respectively m) of up to one million. Computing B takes over 20 seconds for such input sizes using two scales. Using multiple scales could further reduce such costs, an investigation which is subject to future work. Note also that we currently compute our two scales using superpixels (Sect. 4.1.2), which only works for *image* inputs. However, our multiscale idea is generic—one can use any subdivision of the input domain, e.g., quadtrees, octrees or any similar multiresolution scheme.

Limitations: The arguably largest limitation of IBIX is its parameterization. That is, one should decide how many jitter levels N and jitter range size H to use (Sect. 3.2). Too conservative bounds hereof will inevitably only expose the working of the regressor f for a small part of its dynamic range. Setting N and H is, for now, application dependent, based on the expected range and dynamics of f. Separately, IBIX is designed, for now, to explain *single* input samples $\mathbf{x}$. This is on purpose, since existing VA methods do not handle this use-case well (Sect. 2). Extending IBIX to aggregate its findings for entire *datasets*, while maintaining its attractive speed, ease-of-use, and genericity, is a key direction to explore next.

IBIX can explain how the input of a regressor influences certain parts of its output, and conversely. This is aimed to help model engineers to spot problematic inference pertaining to certain input and/or output structures, such as demonstrated in Sect. 4.2. However, IBIX does not (aim to) solve such inference problems—it only exposes their presence. It is, still, the task of the model engineer to detect patterns in such problems and, based on that, devise changes to the model's training data, hyperparameters, or architecture to correct these.

6 Conclusion

We have presented Instance-Based Inference Explainers (IBIX), a framework for building visual explanations of the way multidimensional regressors infer their results for particular instances of interest. IBIX has a simple underlying operation, essentially measuring the rate of change of dimensions of an output (inferred) sample as function of change of the dimensions of the corresponding input. By relating the two changes, IBIX proposes a forward mapping explanation that highlights the output dimensions strongest affected by user-selected dimensions in an input sample; and a backward mapping explanation that, given user-selected dimensions in an output sample, highlights the input dimensions which strongest affect that selection. IBIX is simple to implement, works generically for any multidimensional regressor working on quantitative data, needs no knowledge of the regressor's internals, and is easy to use.

Several extension directions are possible. We envisage extending IBIX to explain groups of samples rather than individual ones, thereby lifting insights on the regressor's operation to a higher, more general, level. Alternatively, we consider designing specialized classes perturbations—generic but also application-specific—that users can select to 'probe' a given regressor's response to obtain finer-grained understanding of its functioning, similar to impulse-response testing in dynamical systems analysis. Separately, we aim to extend the bi-level acceleration scheme for backward mapping computation to a multilevel one, thereby bringing its operation to (near) real time without resolution trade-offs. Finally, as IBIX is fully generic in terms of the explored model, we aim to apply it to a larger class of multidimensional regressors beyond image-to-image ones or deep-learning models.

Acknowledgements We acknowledge the help of Mathijs de Boer with the implementation and evaluation of the MRI-to-pseudo-CT application. We also acknowledge the advice of dr. Matteo Maspero in developing the Pix2Pix generating network used for the same application and prof. Nico van den Berg for stimulating discussions.

References

1. Das A, Rad P. Opportunities and challenges in explainable artificial intelligence (XAI): a survey. 2020. arXiv:2006.11371
2. Ribeiro M, Singh S, Guestrin C. Why should I trust you?: explaining the predictions of any classifier. In: Proceedings of ACM SIGMOD KDD; 2016, p. 1135–44.
3. Adadi A, Berrada M, Bhateja V, Satapathy S, Satori H. Explainable AI for healthcare: from black box to interpretable models. Embedded Syst Artif Intell. 2020;1076:327–37.
4. Yang G, Ye O, Xia J. Unbox the black-box for the medical explainable AI via multi-modal and multi-centre data fusion: a mini-review, two showcases and beyond. 2021. arXiv:2102.01998
5. Rodrigues F, Espadoto M, Hirata R, Telea AC. Constructing and visualizing high-quality classifier decision boundary maps. Information. 2019;10(9):280.
6. Garcia R, Telea A, da Silva B, Torresen J, Comba J. A task-and-technique centered survey on visual analytics for deep learning model engineering. Comput Graph. 2018;77:30–49.
7. van der Velden BH, Kuijf HJ, Gilhuijs KG, Viergever MA. Explainable artificial intelligence (XAI) in deep learning-based medical image analysis. 2021. arXiv:2107.10912 [eess.IV]
8. Chen L-C, Papandreou G, Kokkinos I, Murphy K, Yuille AL. Deeplab: semantic image segmentation with deep convolutional nets, atrous convolution, and fully connected crfs. IEEE Trans Pattern Anal Mach Intell. 2017;40(4):834–48.
9. Ren S, He K, Girshick R, Sun J. Faster R-CNN: towards real-time object detection with region proposal networks. 2015. arXiv:1506.01497
10. Akkus Z, Galimzianova A, Hoogi A, Rubin DL, Erickson BJ. Deep learning for brain MRI segmentation: state of the art and future directions. J Digit Imaging. 2017;30(4):449–59.
11. Spinner T, Schlegel U, Schäfer H, El-Assady M. explAiner: a visual analytics framework for interactive and explainable machine learning. IEEE Trans Vis Comput Graph. 2020;26(1):1064–74.
12. Hohman F, Kahng M, Pienta R, Chau DH. Visual analytics in deep learning: an interrogative survey for the next frontiers. IEEE Trans Vis Comput Graph. 2018;25(8):2674–93.
13. Rauber PE, Fadel SG, Falcão AX, Telea AC. Visualizing the hidden activity of artificial neural networks. IEEE Trans Vis Comput Graph. 2016;23(1):101–10.
14. Seifert C, Aamir A, Balagopalan A, Jain D, Sharma A, Grottel S, Gumhold S. Visualizations of deep neural networks in computer vision: a survey. In: Transparent data mining for big and small data. Springer; 2017, p. 123–44.
15. Ma Y, Fan A, He J, Nelakurthi AR, Maciejewski R. A visual analytics framework for explaining and diagnosing transfer learning processes. 2020. arXiv:2009.06876
16. Maaten LVD, Hinton G. Visualizing data using t-SNE. J Mach Learn Res. 2008;9:2579–605.
17. Rauber PE, Falcão AX, Telea AC. Projections as visual aids for classification system design. Inf Vis. 2018;17(4):282–305.
18. Benato BC, Telea AC, Falcão AX. Semi-supervised learning with interactive label propagation guided by feature space projections. In: Conference on graphics, patterns and images (SIBGRAPI); 2018. p. 392–99.
19. Sedlmair M, Aupetit M. Data-driven evaluation of visual quality measures. Comput Graph Forum. 2015;34(3):201–10.
20. Bernard J, Zeppelzauer M, Sedlmair M, Aigner W. VIAL: a unified process for visual interactive labeling. Vis Comput. 2018;34(9):1189–207.
21. Behrisch M, Korkmaz F, Shao L, Schreck T. Feedback-driven interactive exploration of large multidimensional data supported by visual classifier. In: IEEE conference on visual analytics science and technology (VAST); 2014. p. 43–52.
22. Tuia D, Volpi M, Copa L, Kanevski M, Munoz-Mari J. A survey of active learning algorithms for supervised remote sensing image classification. IEEE J Sel Top Signal Process. 2011;5(3):606–17.
23. Saito PTM, Suzuki CTN, Gomes JF, Rezende PJ, Falcão AX. Robust active learning for the diagnosis of parasites. Pattern Recogn. 2015;48(11):3572–83.
24. Ren M, Zeng W, Yang B, Urtasun R. Learning to reweight examples for robust deep learning. In: International conference on machine learning; 2018, p. 4334–343.
25. Harley AW, An interactive node-link visualization of convolutional neural networks. In: International symposium on visual computing; 2015, p. 867–77.
26. Bernard J, Zeppelzauer M, Lehmann M, Müller M, Sedlmair M. Towards user-centered active learning algorithms. Comput Graph Forum. 2018;37(3):121–32.
27. Zeiler MD Fergus R. Visualizing and understanding convolutional networks. In: European conference on computer vision; 2014. p. 818–33.
28. Pezzotti N, Höllt T, Van Gemert J, Lelieveldt BPF, Eisemann E, Vilanova A. Deepeyes: progressive visual analytics for designing deep neural networks. IEEE Trans Vis Comput Graph. 2017;24(1):98–108.
29. Liu M, Shi J, Li Z, Li C, Zhu J, Liu S. Towards better analysis of deep convolutional neural networks. IEEE Trans Vis Comput Graph. 2016;23(1):91–100.
30. Wongsuphasawat K, Smilkov D, Wexler J, Wilson J, Mane D, Fritz D, Krishnan D, Viégas FB, Wattenberg M. Visualizing dataflow graphs of deep learning models in tensorflow. IEEE Trans Vis Comput Graph 2017;24(1):1–12.

31. Abadi M et al. TensorFlow: large-scale machine learning on heterogeneous systems. 2015. Software available from tensorflow.org. https://www.tensorflow.org/
32. Choo J, Liu S. Visual analytics for explainable deep learning. IEEE Comput Graph Appl. 2018;38(4):84–92.
33. Yosinski J, Clune J, Nguyen A, Fuchs T, Lipson H. Understanding neural networks through deep visualization. In: Deep learning workshop, international conference on machine learning (ICML); 2015.
34. Hohman F, Park H, Robinson C, Chau DH. SUMMIT: scaling deep learning interpretability by visualizing activation and attribution summarizations. IEEE Trans Vis Comput Graph. 2019;26(1):1096–106.
35. Zeiler MD, Krishnan D, Taylor GW, Fergus R. Deconvolutional networks. In: IEEE conference on computer vision and pattern recognition; 2010. p. 2528–35.
36. Kahng M, Andrews PY, Kalro A, Chau DH. ActiVis: visual exploration of industry-scale deep neural network models. IEEE Trans Vis Comput Graph. 2017;24(1):88–97.
37. Nguyen A, Yosinski J, Clune J. Multifaceted feature visualization: Uncovering the different types of features learned by each neuron in deep neural networks. In: Visualization for deep learning workshop, international conference in machine learning; 2016. arXiv:1602.03616
38. Dosovitskiy A, Brox T. Inverting visual representations with convolutional networks. In: IEEE conference on computer vision and pattern recognition; 2016, p. 4829–37.
39. Simonyan K, Vedaldi A, Zisserman A. Deep inside convolutional networks: visualising image classification models and saliency maps. 2013. arXiv:1312.6034
40. Mahendran A, Vedaldi A. Visualizing deep convolutional neural networks using natural pre-images. Int J Comput Vis. 2016;120(3):233–55.
41. Kahng M, Thorat N, Chau DH, Viégas FB, Wattenberg M. Gan lab: understanding complex deep generative models using interactive visual experimentation. IEEE Trans Vis Comput Graph. 2018;25(1):310–20.
42. Liu M, Shi J, Cao K, Zhu J, Liu S. Analyzing the training processes of deep generative models. IEEE Trans Vis Comput Graph. 2017;24(1):77–87.
43. Wang J, Gou I, Yang H, Shen H-W. Ganviz: a visual analytics approach to understand the adversarial game. IEEE Trans Vis Comput Graph. 2018;24(6):1905–17.
44. Zhou B, Khosla A, Lapedriza A, Oliva A, Torralba A. Learning deep features for discriminative localization,. In: IEEE conference on computer vision and pattern recognition; 2016, p. 2921–29.
45. Selvaraju RR, Cogswell M, Das A, Vedantam R, Parikh D, Batra D. Grad-cam: visual explanations from deep networks via gradient-based localization. In: IEEE international conference on computer vision; 2017. p. 618–26.
46. Li H, Tian Y, Mueller K, Chen X. Beyond saliency: understanding convolutional neural networks from saliency prediction on layer-wise relevance propagation. Image Vis Comput. 2019;83:70–86.
47. Mahendran A, Vedaldi A. Salient deconvolutional networks. In: European conference on computer vision; 2016. p. 120–35.
48. Guidotti R, Monreale A, Ruggieri S, Turini F, Giannotti F, Pedreschi D. A survey of methods for explaining black box models. ACM Comput Surveys (CSUR). 2018;51(5):1–42.
49. Bazzani L, Bergamo A, Anguelov D, Torresani L. Self-taught object localization with deep networks. In: IEEE winter conference on applications of computer vision (WACV), 2016; p. 1–9.
50. Li D, Huang J-B, Li Y, Wang S, Yang M-H. Weakly supervised object localization with progressive domain adaptation. In: IEEE conference on computer vision and pattern recognition; 2016. p. 3512–520.
51. Zhang D, Han J, Cheng G, Yang M-H. Weakly supervised object localization and detection: a survey. IEEE Trans Pattern Anal Mach Intell. 2021.
52. Masci J, Meier U, Cireşan D, Schmidhuber J. Stacked convolutional auto-encoders for hierarchical feature extraction. In: International conference on artificial neural networks; 2011. p. 52–9.
53. Nair, V., Hinton GE. Rectified linear units improve restricted Boltzmann machines. In: International conference on machine learning (ICML); 2010. p. 807–14.
54. Sutskever I, Martens J, Dahl G, Hinton G. On the importance of initialization and momentum in deep learning. In: International conference on machine learning (ICML); 2013, p. 1139–147.
55. Taylor JR, Williams N, Cusack R, Auer T, Shafto MA, Dixon M, Tyler LK, Henson RN, et al. The Cambridge Centre for ageing and neuroscience (Cam-CAN) data repository: structural and functional MRI, MEG, and cognitive data from a cross-sectional adult lifespan sample. Neuroimage. 2017;144:262–9.
56. Fonov VS, Evans AC, McKinstry RC, Almli CR, Collins DL. Unbiased nonlinear average age-appropriate brain templates from birth to adulthood. Neuroimage. 2009;47:S102.
57. Sofroniew N et al.. napari/napari: 0.4.12rc2,' Oct 2021. Available https://doi.org/10.5281/zenodo.5587893
58. Achanta R, Shaji A, Smith K, Lucchi A, Fua P, Süsstrunk S. SLIC superpixels compared to state-of-the-art superpixel methods. In: IEEE Trans on Pattern Anal Mach Intell. 2012;34(11):2274–282.
59. Owrangi A, Greer P, Glide-Hurst C. MRI-only treatment planning: benefits and challenges. Phys Med Biol. 2018;63(5).
60. Spadea MF, Maspero M, Zaffino P, Seco J. Deep learning based synthetic-CT generation in radiotherapy and PET: a review. Int J Med Phys Res Pract. 2021;48(11):6537–66.
61. Low D. MRI guided radiotherapy. In: Cancer treatment and research. Springer; 2017. p. 41–67.

62. Maspero M, Savelije M, Dinkla A, Seevinck P, Intven M, Jurgenliemk-Schulz I, Kerkmeijer L, van den Berg C. Dose evaluation of fast synthetic-CT generation using a generative adversarial network for general pelvis MR-only radiotherapy. Phys Med Biol. 2018;10(63).
63. Isola P, Zhu I-Y, Zhou T, Efros AA. Image-to-image translation with conditional adversarial networks. In: IEEE conference on computer vision and pattern recognition; 2017, p. 1125–134.
64. Klein S, Staring M, Murphy K, Viergever MA, Pluim JP. Elastix: a toolbox for intensity-based medical image registration. IEEE Trans Med Imaging. 2009;29(1):196–205.
65. Shamonin DP, Bron EE, Lelieveldt BP, Smits M, Klein S, Staring M. Fast parallel image registration on CPU and GPU for diagnostic classification of Alzheimer's disease. Front Neuroinf. 2014;7:50.
66. Ronneberger O, Fischer P, Brox T. U-net: convolutional networks for biomedical image segmentation. In: Medical image computing and computer-assisted intervention (MICCAI). Springer; 2015. p. 234–41.
67. Wang Z, Bovik A, Sheikh H, Simoncelli E. Image quality assessment: from error visibility to structural similarity. IEEE Trans Imag Process. 2004;13(4):600–12.