Permutation-Based Visualisation of Input and Encoded Space in Autoencoders

Alan Inglis*, Andrew Parnell*

***Abstract*—Autoencoders, known for their capability to compress and reconstruct data, play a pivotal role in unsupervised learning tasks. Deciphering the importance of input features and encoded dimensions can enhance the interpretability of the black-box nature of autoencoders. This paper introduces a permutation importance method tailored for evaluating the importance of raw pixel values and encoded dimensions in image data processed by autoencoders. We apply permutation importance in two stages: first, on the original image data to assess the impact of each pixel on the encoded representations; and second, on the encoded space to determine the importance of each dimension in the reconstruction process for different image classes. Our approach reveals how variations in input feature importance affect the encoded representations, shedding light on the encoder's focus and potential biases. Experimental results on benchmark image datasets and on a larger case study, concerning audio samples, demonstrate the efficacy of our method, providing a novel perspective on evaluating feature importance in unsupervised learning scenarios and offering greater interpretability of the inner workings of autoencoders.**

***Index Terms*—Autoencoders, Visualisation, Feature Importance.**

I. Introduction

AUTOENCODERS (AE) are a type of neural network (NN) which have gained popularity due to their strong predictive power and flexibility across different types of data [1]. They are widely used in fields such as financial modelling for detecting anomalies and assessing risks [2], healthcare for analysing medical images and predicting diseases [3], natural language processing (NLP) [4], image recognition [5], and data denoising [6]. AEs have proven particularly effective in image compression/dimension reduction and reconstruction [7], which are the central topics of this study. Dimension reduction helps machines process lower-dimensional data efficiently, which is crucial as handling high-dimensional data can be resource-intensive [8].

Fundamentally, AEs consist of an encoder that compresses the input into a lower-dimensional latent-space representation and a decoder that reconstructs the input from this encoded form. In the context of image compression and reconstruction, the goal is to closely match the original input by capturing essential data characteristics and ignoring less significant features. Activation functions introduce non-linearity, enabling the model to learn complex patterns. For image tasks, common choices are the Rectified Linear Unit (ReLU) and sigmoid functions, with ReLU outputting the input if positive and zero otherwise, while the sigmoid function maps inputs to values between 0 and 1. Performance is measured using loss functions that quantify the difference between the original input and the reconstructed output. Common choices include Mean Squared Error (MSE) and Binary Cross-Entropy (BCE). Optimisers, such as Adam (Adaptive moment estimation) [9], adjust the model's weights to minimise the loss function, improving training efficiency and effectiveness. An example of the basic architecture is shown in Figure 1; for more details on AE architecture, see [10].

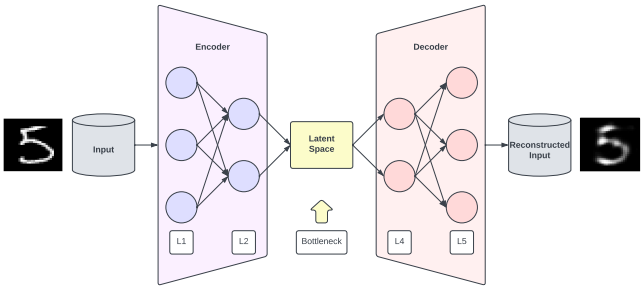


Fig. 1: Basic autoencoder architecture. The process starts with an input image, which undergoes compression by the encoder through layers L1 and L2 into a lower-dimensional latent space representation, called the bottleneck. The decoder then reconstructs the image from the latent space through layers L4 and L5, resulting in the output that closely approximates the original input.

Given the critical applications of autoencoders in sectors such as healthcare and finance, ensuring their decisions are trustworthy is a key concern [11]. However, the interpretability of these models presents a significant challenge. AEs, like many deep learning models, operate as *black-box* systems where the internal workings and the logic behind decisions remain largely opaque. This opacity is primarily due to their reliance on complex non-linear transformations and the presence of numerous parameters, which complicate the tracing and understanding of how inputs are transformed into outputs. Furthermore, the difficulty in interpreting these models also hinders the ability to diagnose and rectify flaws in their

Andrew Parnell's work was supported by the SFI Centre for Research Training in Foundations of Data Science 18/CRT/6049 and the SFI Research Centre award 12/RC/2289_P2. For the purpose of open access, the author has applied a CC BY public copyright licence to any author-accepted manuscript version arising from this submission.

*Alan Inglis: Hamilton Institute, Maynooth University. Email: alan.inglis@mu,ie

*Andrew Parnell: Hamilton Institute, Maynooth University. Email: andrew.parnell@mu.ie

Manuscript received April 19, 2021; revised August 16, 2021.

architecture or training data [12]. For example, biases embedded in the training data can inadvertently be learned by the model, propagating these biases in its outputs. Without a clear understanding of the model's decision-making process, identifying and correcting these biases becomes a complex challenge [13].

Addressing the interpretability challenges of complex models requires the development of methods that simplify their understanding or provide deeper insights into their decision-making processes. To this end, several approaches have been introduced. For example, [14] designed an autoencoder based on decision trees specifically for categorical data, eliminating the need for data transformation. Additionally, [15] modified a technique known as local interpretable model-agnostic explanations [LIME; 16]. Traditionally, LIME generates explanations for individual instances by creating a synthetic dataset through random sampling and perturbations around a specific instance, followed by training a local, interpretable linear model. In their adaptation, an AE is used to improve the weighting function of the local model. These methods aim to make the models more transparent, allowing users and developers to gain a better understanding of a model's behaviour.

Two popular methods for explaining the operation of AEs, especially in image recognition, are feature visualisation and pixel attribution. Feature visualisation allows us to understand which features, such as textures, edges, and patterns, are detected by the neurons, layers, or feature maps of an AE [17]. On the other hand, pixel attribution assesses the importance of each input pixel for tasks like classification or image reconstruction. Typically, there are two types of pixel attribution, gradient based and perturbation based [12]. The gradient based attribution methods typically measures the influence of each input feature on the network's predictions using the gradients of the outputs with respect to the input features, which presupposes that the model behaves nearly linearly around the input values [18]. Perturbation methods include techniques such as the previously mentioned LIME method. These typically involve altering sections of an image to produce explanations that are model-agnostic. These attributions are often shown as heatmaps or contour maps highlighting which pixels are most critical to the model's decisions, and are called saliency maps [18]. The methods outlined later in our work fall into the perturbation category.

In this article, we present our permutation importance method for AEs to improve interpretability by shedding a light on the underlying structure of the data and how the network learns. Our approach is divided into three distinct parts, which we summarise here, but for an in-depth explanation, see Section II:

1) **Input Pixel Importance Method:** We measure the importance of input pixels on the latent space by permuting the values of each pixel in the input data and observing the impact on the AE's performance. This allows us to gauge the relative importance of each pixel within the learned representation. Additionally, we fit a regression model for each pixel against the encoded dimension to determine the directional importance by examining the sign of the slope. Figure 2 shows the basic workflow process.
2) **Latent Space Importance Method:** We apply permutation importance to the latent space data to identify which latent dimensions are important for reconstructing images of specific classes. Figure 3 shows the basic workflow of this process.
3) **Analysis and Visualisation:** To enable a coherent picture of what the AE is doing, we combine the findings from both input pixel importance and latent space importance methods into a single visualisation. This combination allows us to comprehensively understand how individual input pixels and latent dimensions contribute to the AE's performance and reconstruction accuracy.

In addition, we have developed an `R` package, called `aim` (Autoencoder Importance Mapping) that includes our permutation methods and various plotting functions and can be found at https://github.com/AlanInglis/AIM. This package also features a Shiny [19] app that showcases both directional pixel importance and encoded dimension importance jointly. All plots in Sections III and IV were generated using the `aim` package. A demonstration of the Shiny app is shown in Figure 4, with a more detailed explanation of the app available in Appendix B.

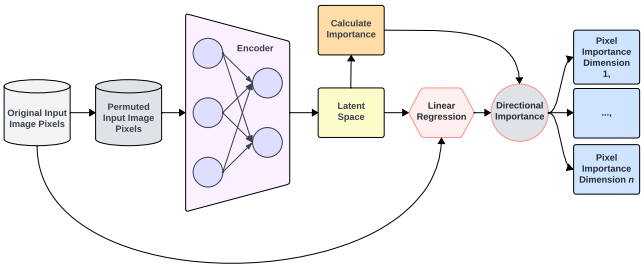


Fig. 2: Workflow demonstrating stage 1 of our approach; the pixel importance method. A permuted input is passed through the encoder part of an AE. From this the importance value is calculated using the mean square error. Using the encoded space as the response, a linear model is fit to each input pixel to determine the direction of the importance before visualising each output dimension.

The layout of our paper is as follows. In Section II we describe the permutation importance algorithm and our methods and visualisations. In Section III, we examine two benchmark image datasets and demonstrate our new techniques. In Section IV, we apply our methods in a case study using more complex data. Finally in Section V, we provide some concluding remarks and discuss limitations of our proposed methods. This study was performed using the `keras` [20] and `tensorflow` [21] packages in the `R` [22] language environment. To reproduce the examples in this text, all the code has been made available at https://github.com/AlanInglis/AutoEncoder_Paper.

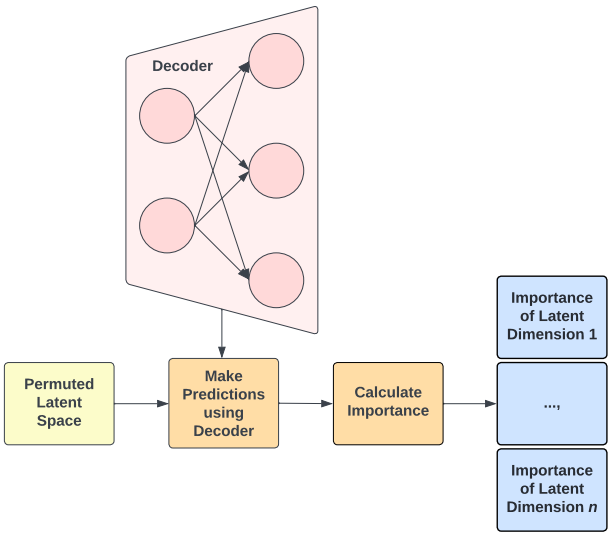


Fig. 3: Workflow demonstrating stage 2 of our latent space importance method. Each encoded dimension is permuted before making predictions using the decoder part of the AE. The importance is calculated using the mean square error before being visualised.

II. Visualising Feature Importance in Autoencoders

In our work, we apply the permutation feature importance technique at both the input and encoded parts of the AE and visualise the results. These visualisations not only enhance understanding of the model's performance across various processing stages but also provide insights into the influence of specific features and dimensions on image reconstruction, enabling a deeper exploration of the model's behaviour. In this section, we describe our methods to obtain the importance scores and demonstrate our visualisations using an AE specifically fit on the digits zero and one from the MNIST data [23] for the purpose of illustration and simplicity, with the encoding dimensions set to a size of 12. This approach was chosen to highlight the methods under discussion. The results are shown in Figure 4 via the use of our Shiny app (see Appendix B for a demonstration of the apps features) and are discussed in more detail in the subsequent subsections. To start, we provide a brief overview of the permutation importance algorithm, as understanding this will help the reader grasp our methods.

A. Permutation Importance

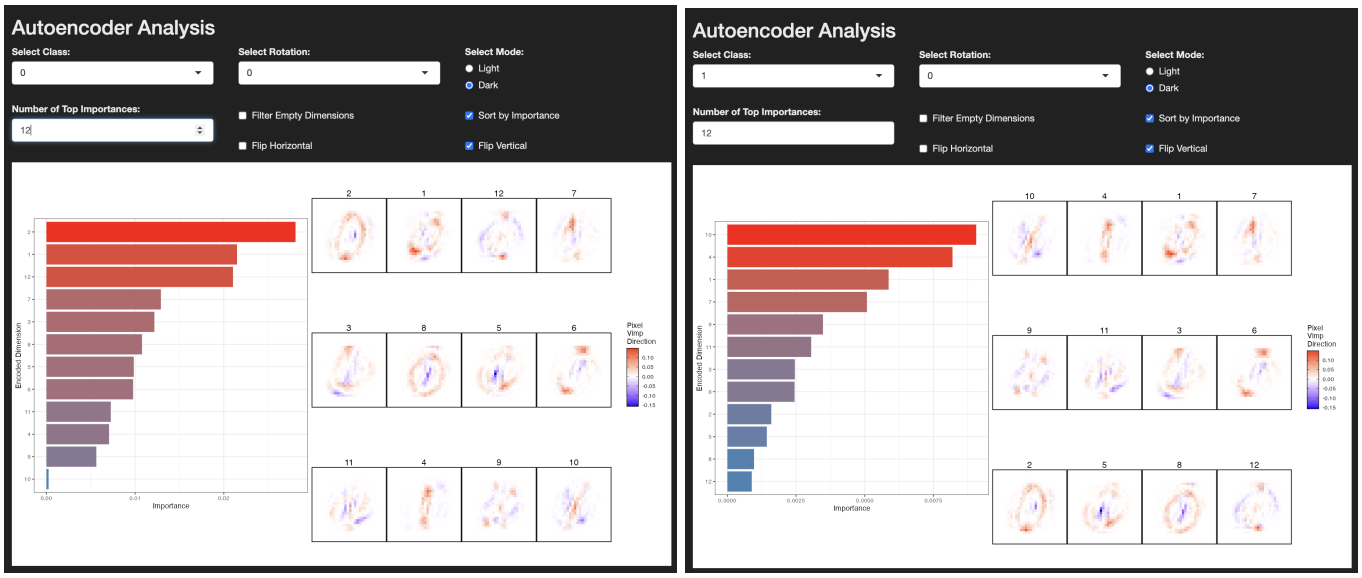
Permutation importance, first proposed by [24], is a technique used to determine which of the input variables has the greatest effect on the predicted outcome of a machine learning model. The permutation algorithm (shown in Appendix A) begins with measuring the model's predictive accuracy, then, for every variable, shuffling the variable and updating the predictive accuracy using the revised data set. This shuffling of the variables values, breaks any relationship between it and the response variable, thus providing insight into how much the model depends on the feature. The difference between the performance of the permuted model and the baseline model when a single feature value is randomly shuffled is considered the permutation-variable importance score.

B. Importance in Encoded Representations

To assess the importance of the encoded representations, we apply the permutation feature importance method to the encoder component of an AE by permuting the pixels of the raw input image (see Figure 2). This approach allows us to identify which pixels in the image data significantly influence the learned encoded representations and, consequently, the quality of the reconstructed images. To achieve this, we initially encode the original, unaltered test images to create a baseline of encoded outputs. We then permute each pixel's values throughout the dataset and re-encode the altered images. The difference in the encoded outputs caused by the permutation is measured using the MSE. The results are then averaged across multiple permutations to assign an importance score to each pixel. This importance score illustrates the degree to which changes in pixel values can affect the encoded representations and, by extension, the quality of the image reconstruction and represents which pixels are most important for the encoder when compressing the data into a lower-dimensional space. To further refine our understanding of each pixel's importance, we introduce a method that uncovers the directional importance of individual pixels within the encoding process. By fitting linear regression models to each pixel against encoded dimensions and simplifying the coefficients, we classify their influence as negative, neutral, or positive. These adjusted coefficients are then combined with permutation importance scores and visualised as heatmaps, revealing both the importance and direction of each pixel's impact. The algorithm for this process can be found in Appendix A and an example visualisation can be seen as the heatmaps in Figures 4a and 4b.

C. Encoded Dimension Importance in Image Reconstruction

Finally, we explore the importance of individual dimensions within the encoded space on the reconstruction quality of images (see Figure 3). In this method, we begin by isolating the decoder component from a broader AE model. Using this isolated decoder, we reconstruct images from their encoded states, establishing a baseline of reconstruction quality for images from a targeted category. We then permute each dimension in the encoded representations of the category-specific images a set number of times. After each permutation, the newly altered encoded data is processed by the decoder to produce reconstructed images. The quality of these reconstructions is then compared to the baseline using the MSE. The importance scores are then obtained by averaging over all permutations. This process provides insight into which encoded dimensions are essential for effectively reconstructing different categories. The algorithm describing this process can be found in Appendix A and an example visualisation can be seen as the barplots in Figures 4a and 4b.



(a) Digit 0 (b) Digit 1

Fig. 4: Permutation importance plot. Panel (a) shows the sorted importance of each encoded dimension in image reconstruction in the left-hand barplot and the directional importance of the input pixels in the right-hand heatmaps, sorted by the importance of the encoded dimensions, for the digit 0 from the MNIST data. Panel (b) shows the importance for the same metrics for digit 1 from the MNIST data.

D. Visualising Feature and Dimension Importance

To visualise the encoded dimension importance in image reconstruction, we use a bar chart to depict the importance scores across different encoded dimensions. The barplots in Figures 4a and 4b show examples for the digits zero and one, respectively, with the x-axis displaying the importance scores and the y-axis displaying the encoded dimensions, sorted by importance.

The importance of encoded dimension representations is further visualised in Figures 4a and 4b as heatmaps for each of the 12 encoding dimensions of our AE, also for the digits zero and one, respectively. These heatmaps are constructed by reshaping the importance scores of each pixel into the original image dimensions. Each panel of the heatmaps represent one dimension of the encoded space. For a direct comparison, the heatmaps are sorted according to the most important encoded dimensions. Each pixel is coloured on a diverging colour scale where blue represents negative values, red represents positive values, and the intensity of each colour indicates the importance.

In Figure 4, we observe that the top three most important encoded dimensions for the digit zero are dimensions 2, 1, and 12, while for the digit one, they are dimensions 10, 4, and 1. This suggests that these specific encoded dimensions capture critical features necessary for accurate reconstruction. Further analysis of the heatmaps support this, showing varied patterns of importance across different dimensions. Specifically, in encoding dimension 2, the red-coloured pixels distinctly outline the shape of the digit zero, indicating its crucial role in recognising and reconstructing zeros. In contrast, in dimensions 10 and 4, the red pixels form the straight line characteristic of the digit one, highlighting their importance for identifying and reconstructing ones.

It is noteworthy that dimension 1 is among the top three most important encoded dimensions for both digits zero and one. This suggests that dimension 1 captures fundamental features that are important for the reconstruction of both digits. The shared importance of this dimension indicates that it may be encoding common structural elements or characteristics essential for distinguishing and reconstructing these digits accurately. Some dimensions, like 9 and 11, display seemingly random clusters of red or blue pixels that do not form any specific shape, likely encoding more abstract or less specific features. These abstract features contribute to the overall variability of the AE's representations and might capture more general characteristics of the digits or variations in handwriting styles.

III. Method Demonstration with Example Datasets

Here we apply our methods on two benchmark data sets, both of which are variants of the the well known MNIST (Modified National Institute of Standards and Technology) dataset [23]. For each dataset, we fit an AE model following a consistent workflow. This included splitting the dataset into training and test (85-15), normalising and reshaping images, defining the model with 32 latent dimensions, compiling with the Adam optimiser and binary crossentropy loss, and training with a batch size of 256. The only parameter varied was the number of epochs to examine the effect of the permutation importance approach under different settings.

The first data set used in this Section is the Extended-MNIST dataset (EMNIST) [25] and consists of 28x28 pixel

grayscale images of handwritten letters. In our work we select only the uppercase vowels from the dataset. This results in 12,000 training images and 2,000 test images. The second data set is the Fashion-MNIST dataset (FMNIST) [26] and consists of 60,000 training and 10,000 testing 28x28 pixel grayscale images, each representing one of ten fashion categories and are described in Table I.

A. EMNIST Data

For the EMNIST data, we build an AE by following the workflow previously outlined. Here we only train the model using capitalised vowels and set number of epochs as 100. Figure 5 shows a selection of the original input images against their reconstructed output counterparts.

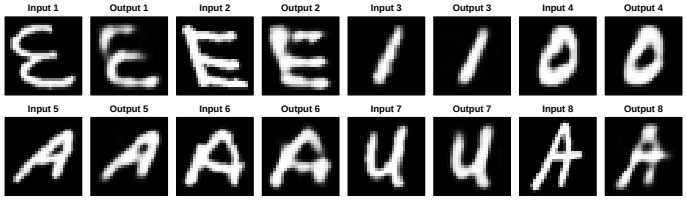
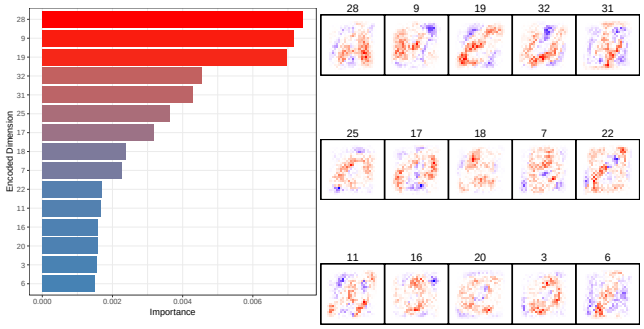
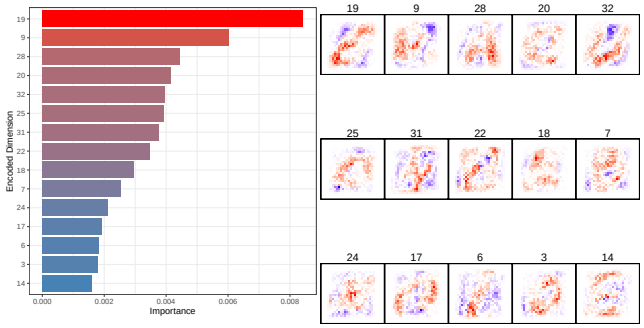


Fig. 5: A selection of original (input) and reconstructed (output) images from at autoencoder using the EMNIST data.

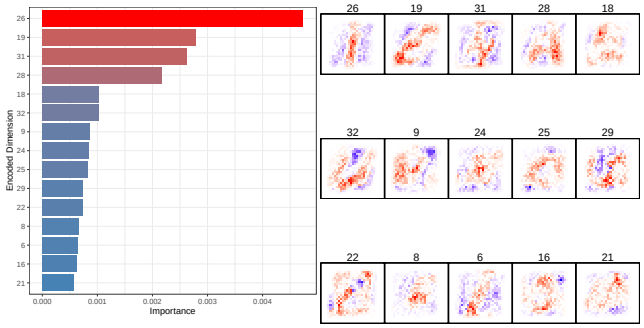
Figure 6 displays the top 15 most important encoded dimensions and the corresponding pixel importance for the classes A, E, and I, shown in 6a, 6b, and 6c, respectively. For clarity and size considerations, the remainder of the plots are not presented via our Shiny app and all legends have been omitted. For plots of all classes, see Appendix C. In Figure 6, the red pixels in the heatmaps for each class's most important encoded dimension typically outline the shape of the respective class, highlighting the regions of the input that are most important for encoding and reconstruction. Dimension 19 appears in the top three most important encoded dimensions for all three classes, suggesting it encodes fundamental features common across different digit structures. However, for class I, dimension 26 is the most important by a considerable margin, highlighting the vertical lines typical of I. Although dimension 19 is the second most important for class I, the reconstruction of I relies heavily on dimension 26, indicating that dimension 19, while still significant, plays a smaller role in this particular case. Dimension 20, although only the fourth most important dimension for class E, appears to capture the distinctive curvature in certain handwritten variations of the letter E (see Figure 5 for example). Notably, this dimension either shows low importance or is absent from the top 15 encoded dimensions for the other classes, suggesting that this specific feature is unique to the representation of E and is not as relevant for distinguishing other characters.



(a) Class A



(b) Class E



(c) Class I

Fig. 6: Top ten most important dimensions for encoded dimension and pixel importance, sorted by encoded dimension importance, across various classes of the EMNIST data. We can see that the most important encoded dimension typically outlines the shape of the respective class.

B. FMNIST Data

The FMNIST data contains 70,0000 images of 10 different types of fashion article classes and is commonly used as a more complex alternative to the traditional MNIST data. Each clothing class label is shown in Table I. An AE was built, using the FMNIST data, as previously outlined, this time with 50 epochs. Figure 7 shows a sample of the original input images compared to the reconstructed images.

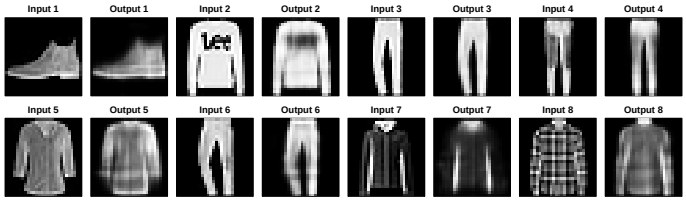
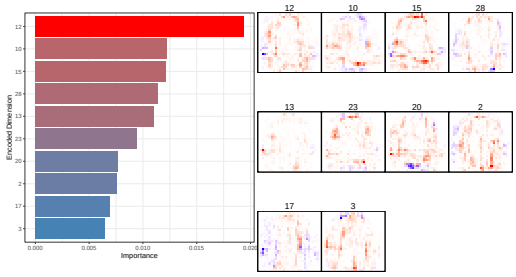


Fig. 7: A selection of original (input) and reconstructed (output) images from at autoencoder using the FMNIST data.

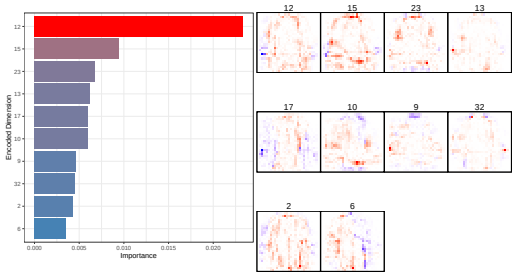
Description	
T-shirt/top	Sandal
Trouser	Shirt
Pullover	Sneaker
Dress	Bag
Coat	Ankle boot

TABLE I: Fashion-MNIST data classes.

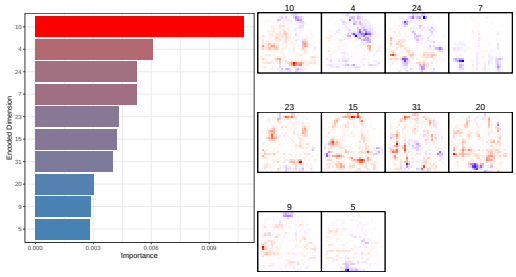
Figure 8 shows the top ten most important dimensions for both the encoded dimension and pixel importance, sorted by the importance of the encoded dimension, across various classes. For brevity, we only show a selection of five classes (see Appendix D for plots of all classes). These are; *Shirt* in panel (a), *T-Shirt* in (b), *Sandal* in (c), *Ankle Boot*, in (d), and *Trouser* in (e). We can observe that items of clothing with similar physical characteristics also share the same most important encoded dimension. For example, *Shirt* and *T-Shirt* both share dimension 12 as their most important encoded dimension. Also, *Sandal* and *Ankle Boot* (both being footwear) both share dimension 10 as their most important dimension. This suggests that these dimensions play a crucial role in determining the basic shape of these classes. Examining the heatmaps, we can observe distinct patterns of pixel importance for each encoded dimension. Specifically, the heatmaps for *Shirt* and *T-Shirt* highlight the central region, which corresponds to the torso area of the clothing, resembling their respective items.



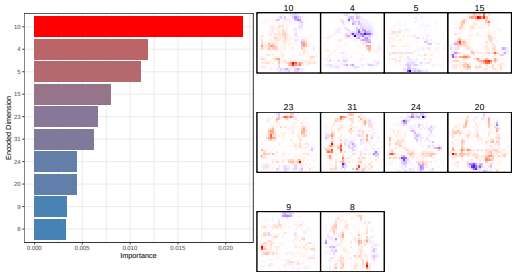
(a) Class $Shirt$



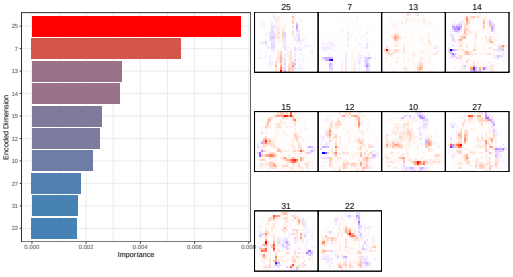
(b) Class $T-Shirt$



(c) Class $Sandal$



(d) Class $Ankle\ Boot$



(e) Class $Trouser$

Fig. 8: Top ten most important dimensions for encoded dimension and pixel importance, sorted by encoded dimension importance, across various clothing classes from the FMNIST data. Notably, dimension 15 is consistently important across all classes, while dimension 25 is unique to trousers, indicating sensitivity to linear patterns.

For *Trouser*, which uniquely has dimension 25 as its most important, we can see a series of straight lines in the red pixels, suggesting that this dimension is particularly sensitive to linear patterns commonly found in trousers. Interestingly, dimension 15 appears in the top 6 of all shown classes and dimension 10 in all but *Trouser*, suggesting that these dimensions capture a fundamental feature common across various types of clothing.

IV. Case Study: Analysing Audio MNIST Data

In this section we apply ou methods on a case study with more complex data. The dataset used in this study consists of 30,000 audio recordings from 60 individuals from various countries, each vocalising the digits from 0 to 9. It was originally studied by [27] and can be found at https://github.com/soerenab/AudioMNIST. These audio samples are transformed into visual representations called spectrograms, an example of which is shown in Figure 9. The axes of a spectrogram represent frequency versus time, while the amplitude of the audio signal at different frequencies and times is depicted through varying colors in these images. In our case, the size of each image is set to 32x32 pixels. The following subsections detail the methodology employed in preparing the audio samples and the specific architecture of the autoencoder.

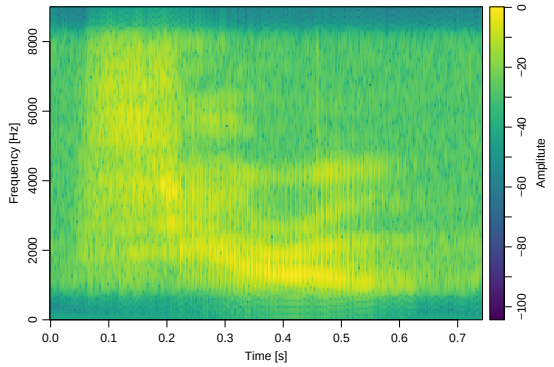


Fig. 9: Example spectrogram of a person saying the number zero.

A. Data Preparation

The raw audio data, visualised as spectrograms, are first pre-processed by smoothing the RGB color channels to enhance the signal and reduce noise. The raw images have dimensions of 32x32x4, where the four channels represent the red, green, and blue color channels and an alpha channel for transparency. Since the alpha channel is not needed for our analysis, it is discarded. Subsequently, using a kernel smoother for irregular 2D data, the RGB channels are merged into a single channel to represent the amplitude. Following this, the data is normalised to the range [0,1].

B. Autoencoder Architecture

The encoder of the autoencoder compresses the input, which consists of 1024 features (flattened 32x32 images), into a compact representation of 32 features through eight dense layers. The number of units in these layers progressively decreases, before reaching the bottleneck layer of 32 units. All layers employ ReLU activation functions. The decoder reconstructs the original image from this compressed representation by mirroring the encoding process, expanding the 32 features back up to 1024 features through a reverse sequence of dense layers, also using ReLU activation functions. The final output layer uses a sigmoid activation function to reconstruct the image. The autoencoder is compiled with the Adam optimiser and a binary cross-entropy loss function, reflecting the binary nature of the normalised pixel values. It is trained over 100 epochs with a batch size of 256, including shuffling and validation with test images. Figure 10 illustrates the original versus reconstructed spectrogram images for a selection of classes. As we can see, the autoencoder had the effect of smoothing and de-noising the original images.

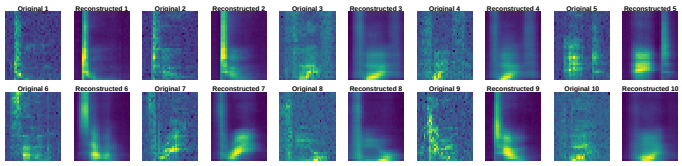


Fig. 10: Original input spectrogram images verses the reconstructed spectrogram images.

C. Analysis of Results

1) UMAP: Here, we used Uniform Manifold Approximation and Projection (UMAP) [28] to visualise the encoded dimensions of the audio MNIST data. This helps us assess how effectively the AE separates the different classes in the encoded space. Using UMAP serves as a sanity check, ensuring that our subsequent results are consistent with the visual separation observed in the UMAP plot. The UMAP is a dimensionality reduction technique that is particularly well-suited for visualising high-dimensional data. It reduces the dimensions of the data to two or three, making it possible to plot the data points in a scatterplot. The technique aims to preserve both the local and global structure of the data, meaning that points that are close together in the high-dimensional space should remain close together in the low-dimensional space, while also maintaining the overall shape and distribution of the data. The UMAP plot in Figure 11 shows the resulting 2D visualisation of the AE applied to the audio MNIST data.

In Figure 11 we can see a moderate separation of the classes, with some noticeable clusters corresponding to specific digits. For instance, digits such as $\{3, 4, 6, 8\}$ show well-defined clusters, indicating that the AE has learned to encode these digits in a way that makes them distinct in the latent space. However, there are also regions with significant overlap, such as between digits $\{0, 2, 7\}$, or $\{1, 5, 9\}$, suggesting that the AE may struggle to distinguish between these digits as effectively. This pattern can be observed in the following importance plots. An interesting note here is that some classes appear as outliers. For example, in between the 6 and 8 cluster we can see a single instance of a 3. This may be due to mis-labelling or it could indicate that the feature representation of this particular

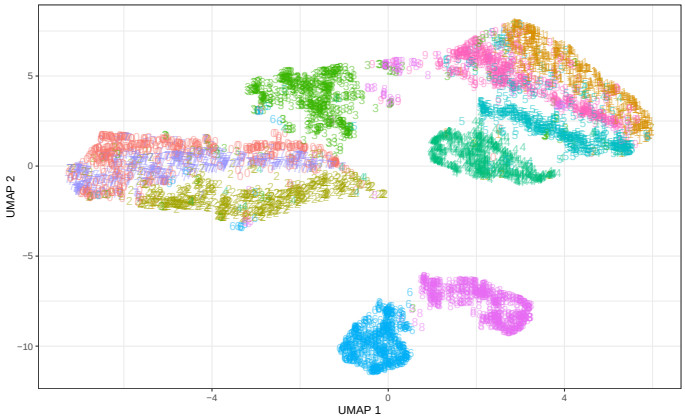


Fig. 11: UMAP plot of the encoded dimensions showing moderate class separation and noticeable clusters, particularly for digits 6 and 8.

3 shares similarities with the 6 and 8 clusters, suggesting ambiguity in the digit's shape or style. A similar effect can be seen where the digit 6 can be found in several of the clusters.

2) Encoded Dimensions and Pixel Importance: In Figure 12, we show the top ten most important encoded dimensions and their corresponding pixel importance plots for the classes 0, 4, 5, 6, 7, and 8 (see Appendix E for plots of all classes). Here we can establish many interesting observations. For example, we can see that dimension 4 appears as one of the top three dimensions for all digits except digit 6 and 8. This implies that dimension 4 might capture a fundamental feature common across most spoken digits, such as the sharp onset and offset of the spoken digits. Spectrogram analysis of these digits reveals a short, sharp burst of sound over a brief duration, indicating that dimension 4 may be important for encoding these transient acoustic features accurately. We also find that dimension 6 is the most important for digits 6 and 7, and the third most important for digit 0. This importance can be attributed to the unique pronunciation features of these digits, such as the fricative 's' and similar 'z' sound in these digits, which requires distinct frequency bands and temporal patterns to be captured effectively by the AE. Referring back to Figure 11, the overlapping regions of $\{0, 7\}$, can be seen in Figure 12, as they share many of the same important dimensions. Additionally, we can see that both digits 6 and 8 have several dimensions that have a much larger relative importance value (when compared to other classes). This observation aligns with the UMAP plot which shows that these digits have unique characteristics that are well-captured by the identified important dimensions.

Examining the heatmaps in Figure 12, we can see that digits such as 4, and 5, highlight importance in lower frequency bands (dimensions 4 and 29), while digits 0, 6, and 7 highlight importance in higher frequency bands (dimensions 6 and 27), which is consistent with them being associated with the fricative 's' (or 'z') sounds, which are higher pitched sounds. This indicates that different encoded dimensions are sensitive to specific frequency ranges, capturing unique acoustic properties of the digits. Notably, digit 8 has, uniquely, dimension 12 as its most important. Followed by 27 and 6. These dimensions exhibit higher frequency sounds, which, as discussed, is understandable for digits, such as, 6 and 7 due to the fricative 's' sound. However, the prominence of high-pitched dimensions for digit 8 may be attributed to the phonetic characteristics of the 't' sound in 'eight,' which also involves a significant amount of high-frequency energy.

3) Concluding Remarks: This case study demonstrates the effectiveness of using a permutation importance approach to analyse the reconstruction of audio data from spectrograms. By examining the importance of both raw input pixels and encoded dimensions, we have identified key features that are critical for accurate reconstruction. Heatmap visualisations revealed that certain encoded dimensions capture shared acoustic features across multiple digits, such as common phonetic properties, while others highlight unique characteristics specific to certain digits. For example, digits 0, 6, and 7 share many of the same importance profiles and highlight similar frequency bands in their pixel importance heatmaps, suggesting common acoustic properties and distinct phonetic features.

V. Conclusion

In this paper, we presented a novel permutation importance method for evaluating the significance of raw pixel values and encoded dimensions in autoencoders applied to image data. By applying permutation importance at two stages (that is, on the original image data and the encoded space), we provided a detailed analysis of the feature importance in the encoding and reconstruction processes. Our approach reveals how variations in input feature importance affect the encoded representations, shedding light on the encoder's focus and potential biases. Additionally, we identified the key encoded dimensions that significantly impact the reconstruction quality for different image classes. Experimental results on benchmark image datasets, including Fashion-MNIST and EMNIST, demonstrated the efficacy of our method. We observed diverse patterns of pixel importance across different encoded dimensions, indicating that autoencoders learn a variety of features from the data. The case study on Audio MNIST data further validated our approach, showing that specific encoded dimensions capture crucial acoustic features for accurate reconstruction.

Our method enhances the interpretability of autoencoders, providing deeper insights into their inner workings. By understanding which features and dimensions are most important, we can improve model transparency and potentially address biases inherent in the training data. Future work could extend this methodology to other types of data and autoencoder architectures, such as variational or conditional autoencoders, further broadening the scope and applicability of permutation importance in unsupervised learning scenarios. While our approach offers significant insights, it also has limitations. The computational cost of permutation importance can be high, especially for large datasets and complex models. Additionally, the method assumes that the importance of features is independent, which may not always hold true in practice. Addressing these limitations in future research could lead to

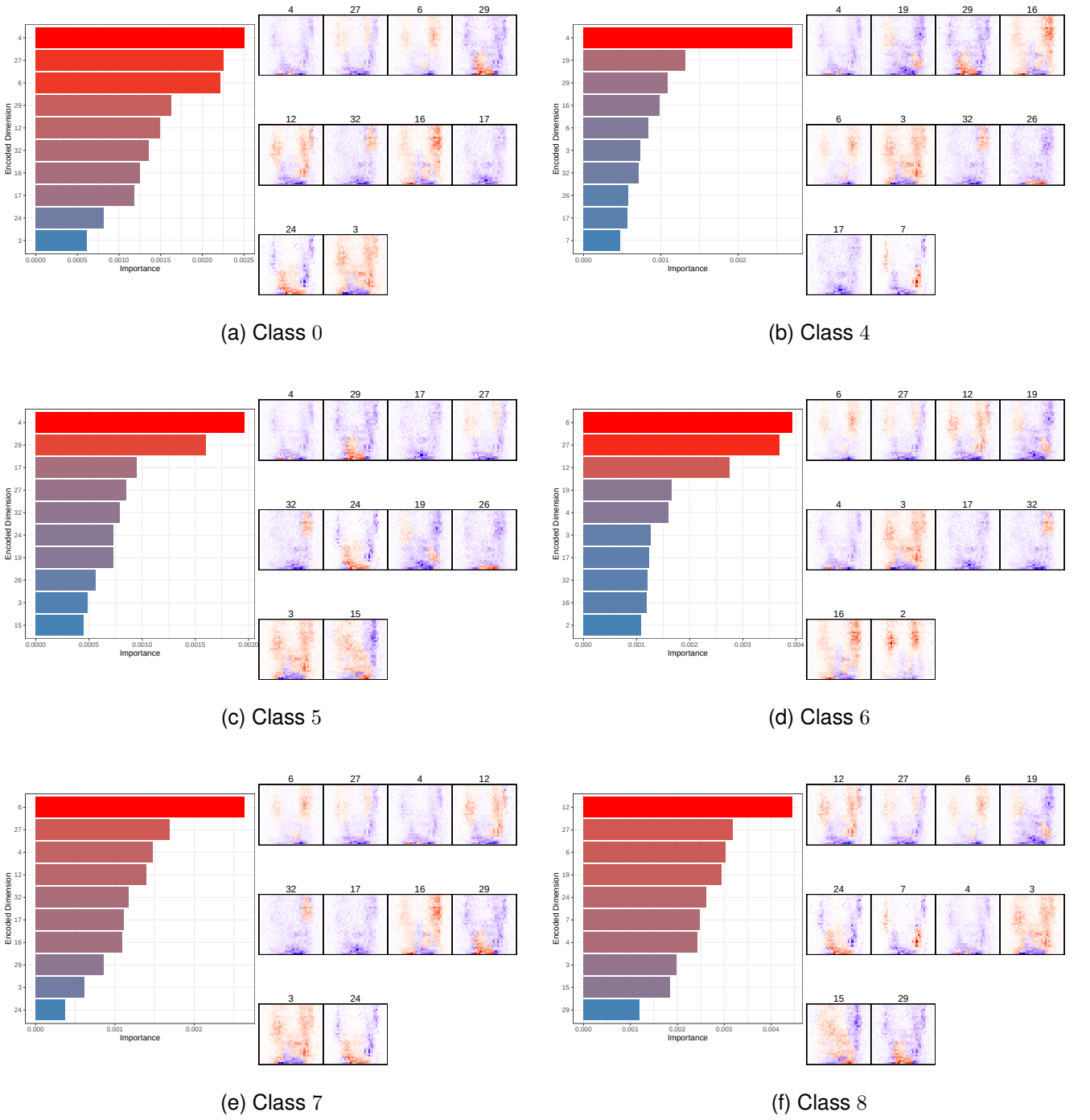


Fig. 12: Top ten most important dimensions for encoded dimension and pixel importance, sorted by encoded dimension importance, across various selected classes of the audio MNIST data.

more efficient and comprehensive interpretability methods for autoencoders and other deep learning models. Moreover, our permutation importance methodology has broader implications for other advanced models such as Variational Autoencoders (VAEs) and Large Language Models (LLMs). For VAEs, understanding the importance of encoded dimensions can aid in improving latent space representations and enhancing the quality of generated data. Similarly, for LLMs, which often rely on intricate encoding mechanisms to understand and generate human language, our approach could be adapted to provide valuable insights into which aspects of the input data (e.g., words, phrases) are most crucial for generating coherent and contextually accurate outputs. By refining and adapting this methodology, we can gain a better understanding and potentially improve performance in various areas of artificial intelligence and machine learning.

ACKNOWLEDGMENTS

This should be a simple paragraph before the References to thank those individuals and institutions who have supported your work on this article.

APPENDIX A
ALGORITHMS

In this section, we present the overall algorithm for our methods. Specifically, the permutation importance for the input pixel and encoded dimension importance.

Algorithm 1: Permutation Importance for Encoded and Decoded Dimensions

Inputs: Encoder model, test data, test labels, autoencoder, list of classes, number of permutations, error metric
Outputs: Feature importances for encoded dimensions and adjusted pixel importance

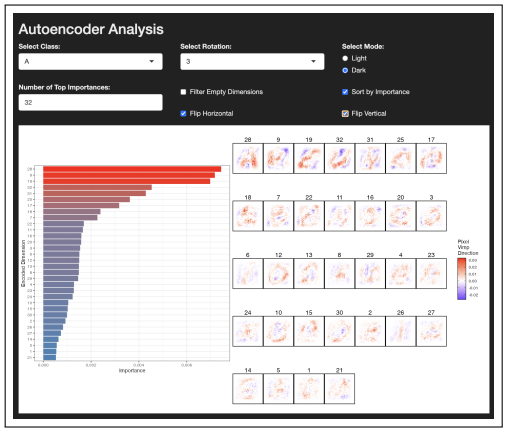
Initialise decoder model from autoencoder
Encode the original test data and compute baseline error

Permutation Pixel Importance:
for *each pixel in the test data* **do**
  for *each permutation* **do**
    Permute the pixel values
    Encode the permuted data
    Calculate importance score for each encoded dimension
  end
end
Average the pixel importance scores over the permutations
Apply linear regression model to adjust pixel importance scores

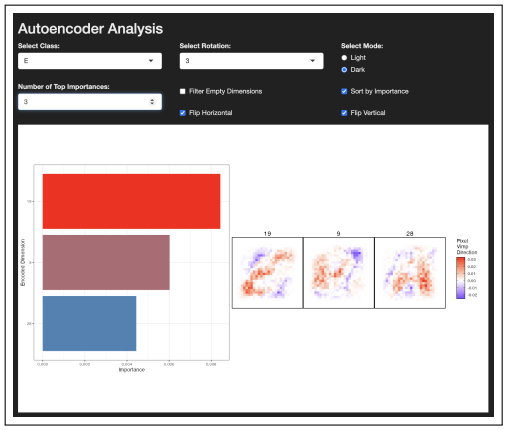
Permutation Encoded Dimension Importance:
for *each class* **do**
  Encode and decode the test data to obtain baseline error
  for *each encoded dimension* **do**
    for *each permutation* **do**
      Permute the encoded dimension values
      Decode the permuted encoded data
      Calculate importance score using the error metric
    end
  end
  Average the importance scores over the permutations
end
return Importance scores for encoded dimensions and adjusted pixel importance scores

APPENDIX B
SHINY APP

In this section, we show a brief demonstration of the Shiny app developed in our `R` package `aim` (Autoencoder Importance Mapping), using the EMNIST vowel data. In Figure B.1, we can see the layout of the app.



(a)



(b)

Fig. B.1: Demonstrations of the Autoencoder analysis Shiny app using EMNIST vowel data. Panel (a) shows all the dimensions for class A. Panel (b) filters the important dimensions from 32 to 3 for the selected class E.

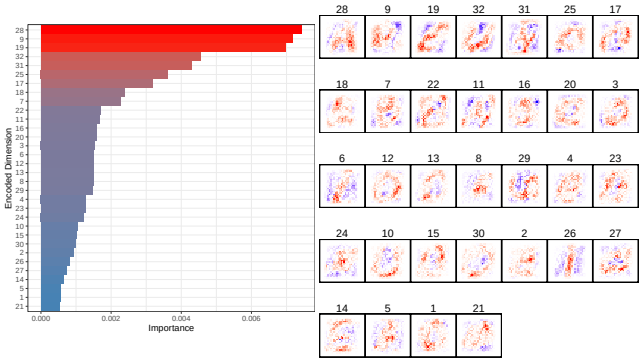
In the main plot window we show both the encoded dimension importance and the pixel importance side-by-side. Additionally, we provide multiple user interfaces, starting from the top left, these are:

- **Select Class:** This allows the user to select and display a specific class from the data.
- **Number of Top Importances:** Allows the user to filter the plot to display the top x most important dimensions.
- **Select Rotation** Specifies the rotation of the heatmaps, where 0 = no rotation, 1 = 90°, 2 = 180°, 3 = 270° .
- **Filter Empty Dimensions:** This filters any dimensions that are not used by the autoencoder.
- **Sort by Importance:** When checked, this sorts the encoded dimension importance from high to low and additionally sorts the pixel importance plots to correspond with the encoded dimensions.
- **Flip Horizontal:** Flips the heatmaps horizontally.
- **Flip Vertical:** Flips the heatmaps vertically.
- **Select Mode:** Allows the user to switch between light and dark modes on the app.

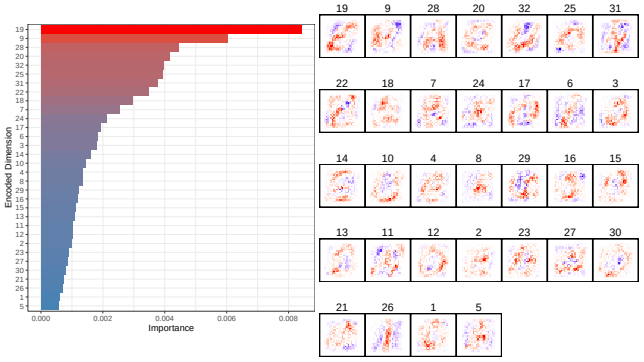
In Figure B.1 (a), we can see that the selected class is A, and we are displaying all the 32 dimensions used

in building the AE model from Section III. As there are no unused dimensions, the `Filter Empty Dimensions` option is unchecked. We are additionally sorting the plots by the encoded dimension importance values. We have also selected the third rotation (that is 270°) and have flipped the image both horizontally and vertically. The inclusion of rotation and flipping options for heatmaps in the Shiny app addresses the need for correct image orientation and enhanced visual clarity. Sometimes the initial orientation of an image is incorrect, and these features allow users to easily adjust the heatmaps to the correct orientation by rotating and flipping. This ensures that the visualisations are displayed in the most readable and interpretable manner. The final option is a mode button which allows a user to select either a 'Dark' mode (with darker surrounding colours) or a 'Light' mode (which has lighter surrounding colours). In this case 'Dark' mode is selected. In panel (b) the selected class has been changed to E and the number of top importances has been set to three. This filters the plot to display the top 3 most important encoded dimensions. All other input remain the same as in panel (a).

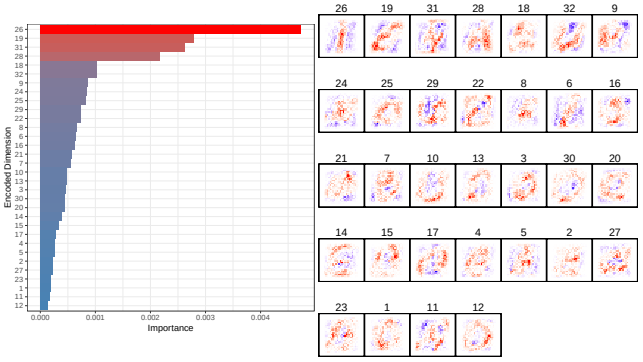
APPENDIX C
EMNIST IMPORTANCE PLOTS FOR ALL CLASSES



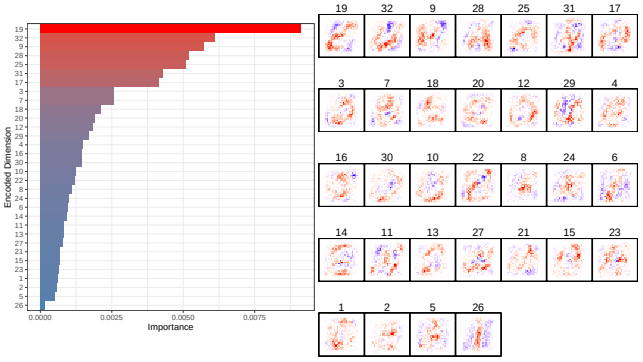
(a) Class A



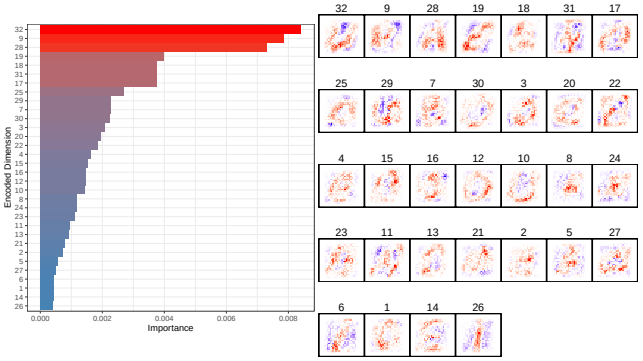
(b) Class E



(c) Class I



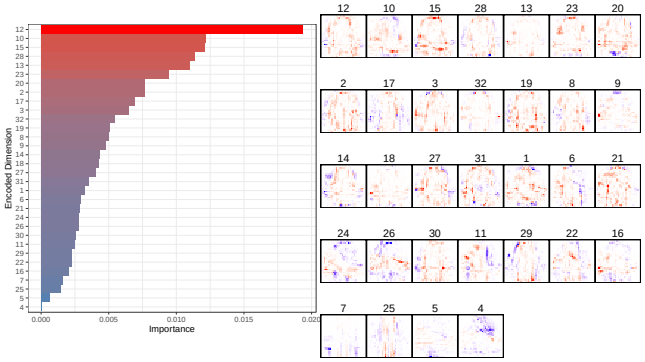
(d) Class O



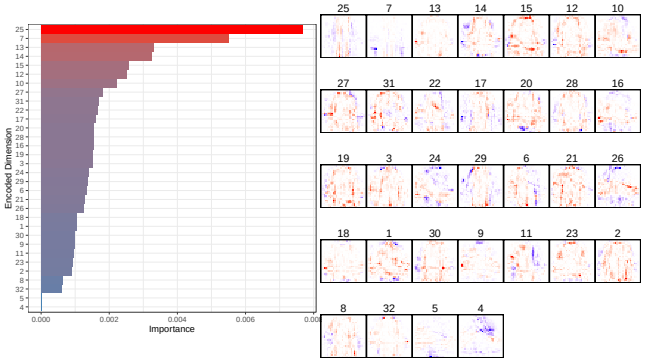
(e) Class U

Fig. C.1: The most important dimensions for encoded dimension and pixel importance, sorted by encoded dimension importance, across all classes of the EMNIST data.

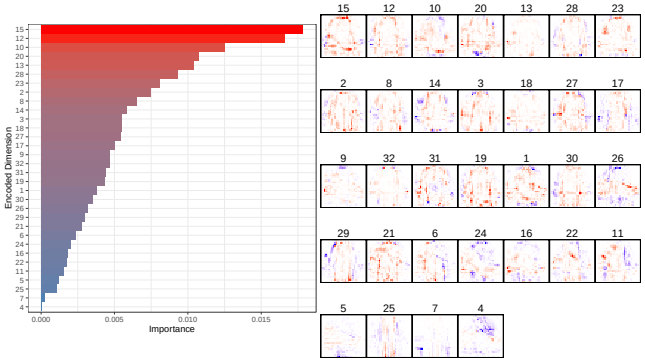
APPENDIX D
FMNIST IMPORTANCE PLOTS FOR ALL CLASSES



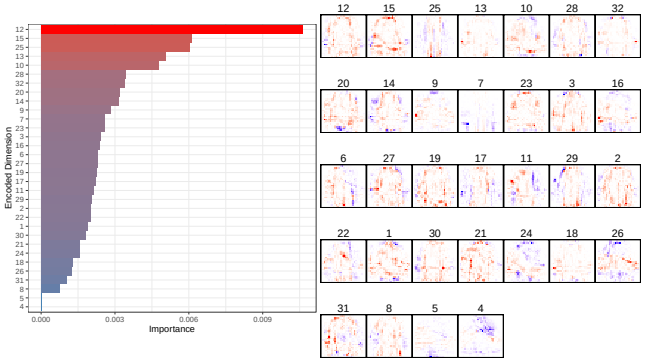
(a) Shirt



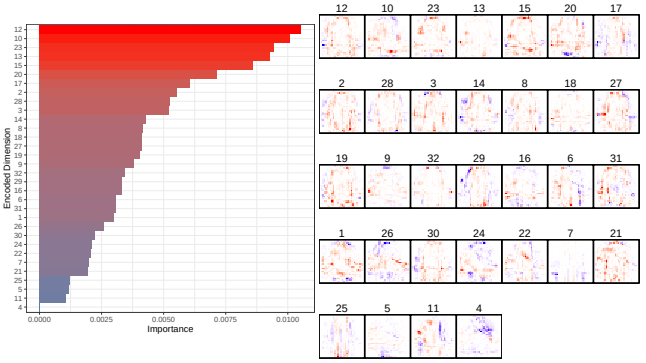
(b) Trouser



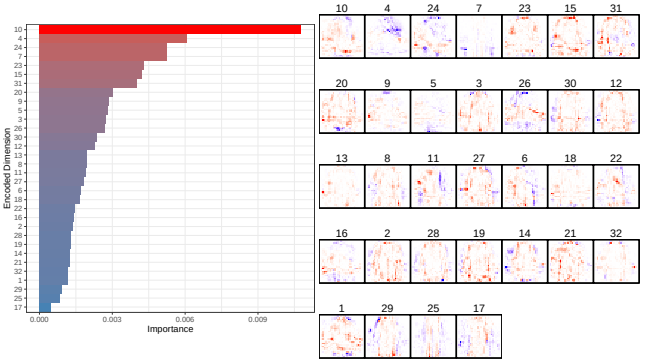
(c) Pullover



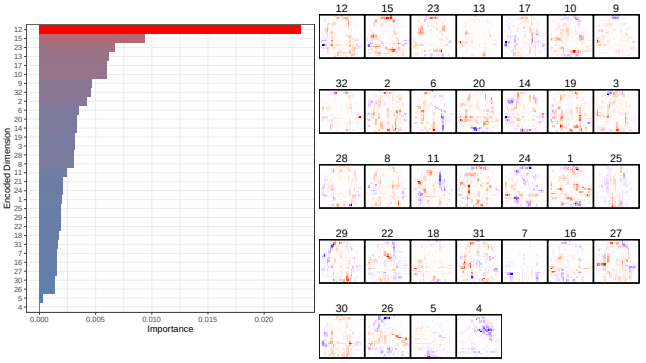
(d) Dress



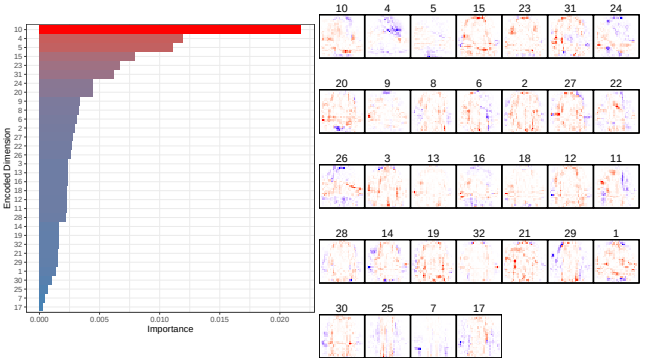
(e) Coat



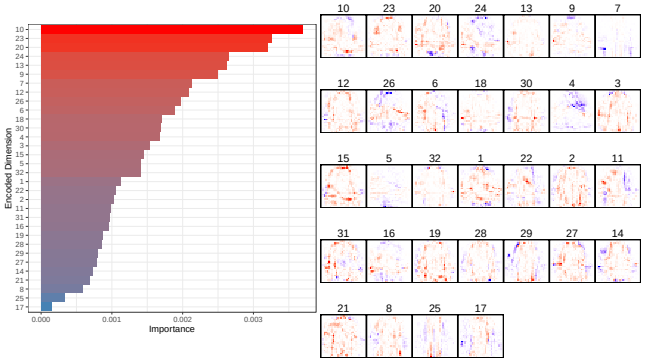
(f) Sandal



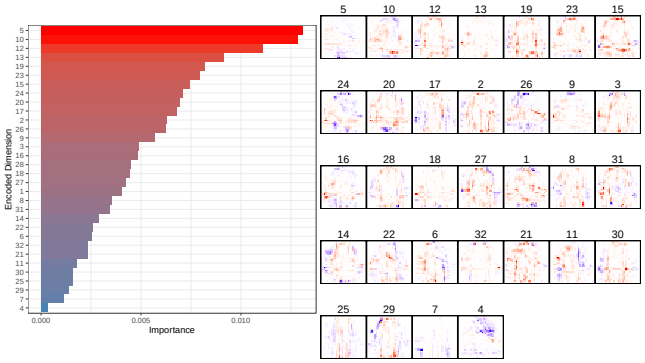
(g) T-Shirt



(h) Ankle Boot



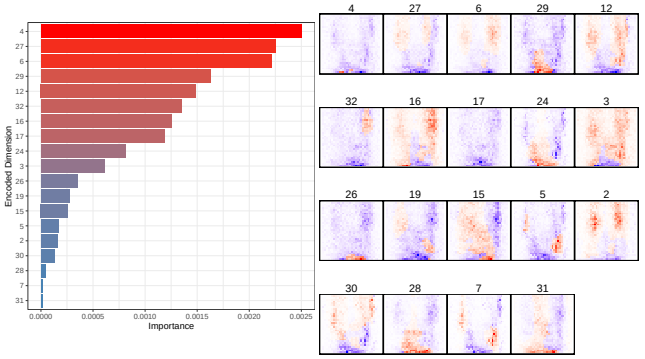
(i) Sneaker



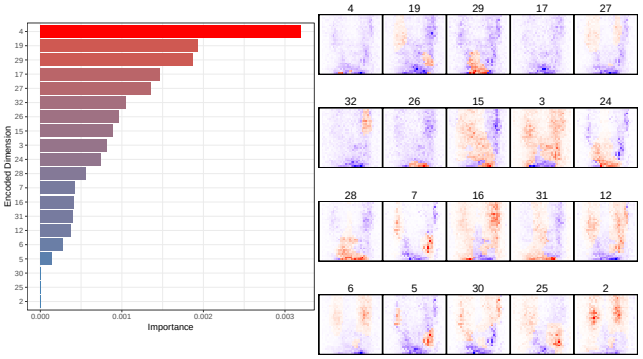
(j) Bag

Fig. D.1: The most important dimensions for encoded dimension and pixel importance, sorted by encoded dimension importance, across all classes of the FMNIST data.

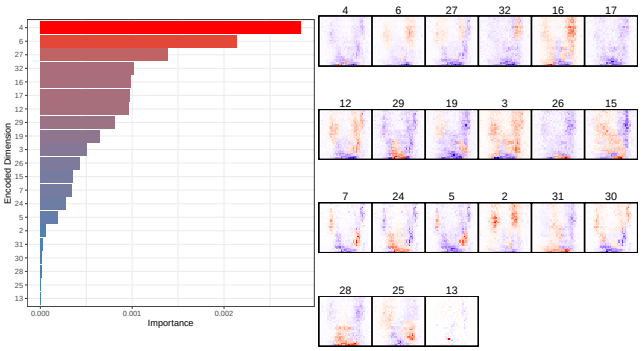
APPENDIX E
AUDIO MNIST IMPORTANCE PLOTS FOR ALL CLASSES



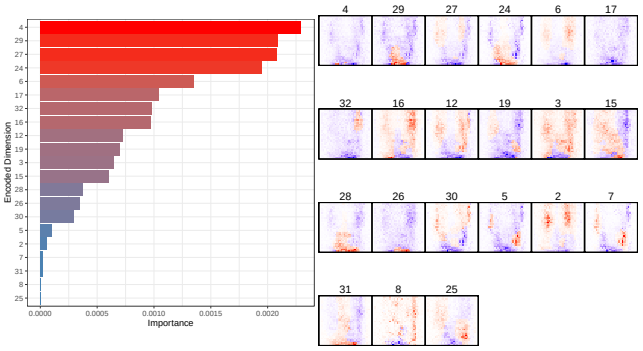
(a) Class 0



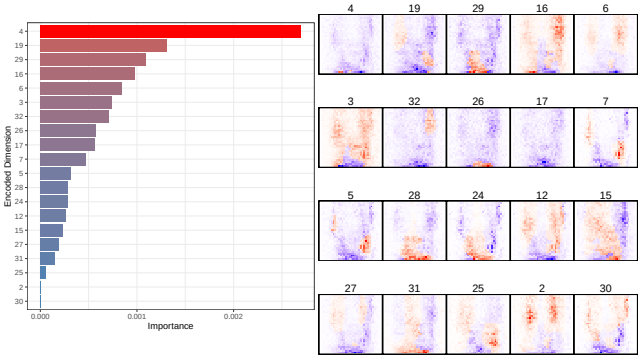
(b) Class 1



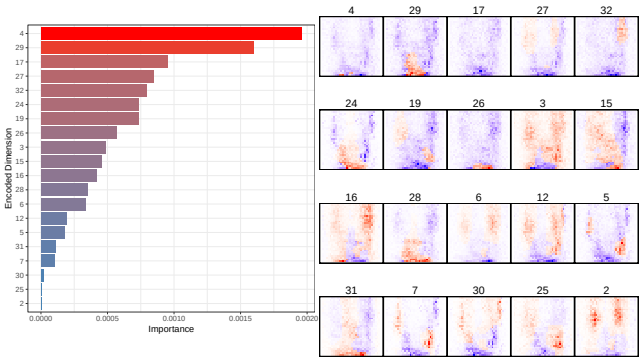
(c) Class 2



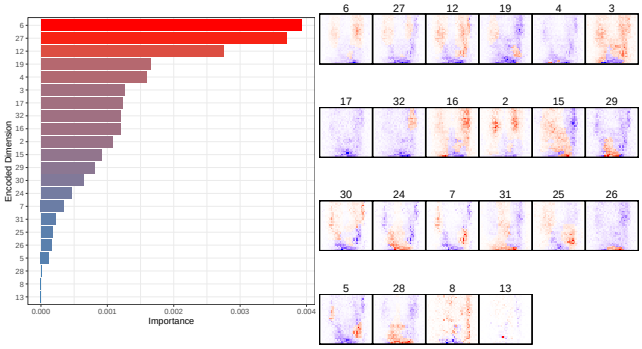
(d) Class 3



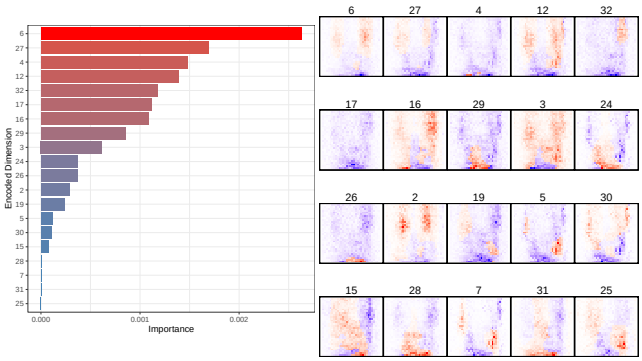
(e) Class 4



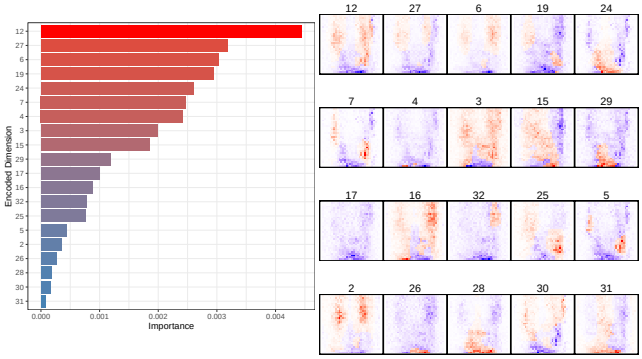
(f) Class 5



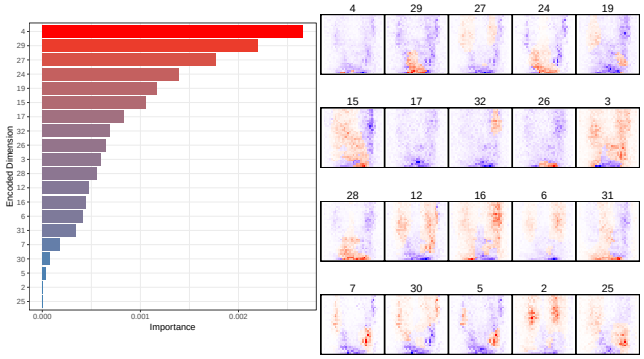
(g) Class 6



(h) Class 7



(i) Class 8



(j) Class 9

Fig. E.1: Top most important dimensions for encoded dimension and pixel importance, sorted by encoded dimension importance, across all classes of the audio MNIST data.

REFERENCES

[1] S. Chen and W. Guo, "Auto-encoders in deep learning—a review with new perspectives," *Mathematics*, vol. 11, no. 8, p. 1777, 2023.

[2] H. D. Nguyen, K. P. Tran, S. Thomassey, and M. Hamad, "Forecasting and anomaly detection approaches using lstm and lstm autoencoder techniques with the applications in supply chain management," *International Journal of Information Management*, vol. 57, p. 102282, 2021.

[3] M. Chen, X. Shi, Y. Zhang, D. Wu, and M. Guizani, "Deep feature learning for medical image analysis with convolutional autoencoder neural network," *IEEE Transactions on Big Data*, vol. 7, no. 4, pp. 750–758, 2017.

[4] A. M. Dai and Q. V. Le, "Semi-supervised sequence learning," *Advances in neural information processing systems*, vol. 28, 2015.

[5] Y. Pu, Z. Gan, R. Henao, X. Yuan, C. Li, A. Stevens, and L. Carin, "Variational autoencoder for deep learning of images, labels and captions," *Advances in neural information processing systems*, vol. 29, 2016.

[6] K. Zhang, W. Zuo, Y. Chen, D. Meng, and L. Zhang, "Beyond a gaussian denoiser: Residual learning of deep cnn for image denoising," *IEEE transactions on image processing*, vol. 26, no. 7, pp. 3142–3155, 2017.

[7] L. Theis, W. Shi, A. Cunningham, and F. Huszár, "Lossy image compression with compressive autoencoders," in *International Conference on Learning Representations*, 2016.

[8] Y. Yang, Q. J. Wu, and Y. Wang, "Autoencoder with invertible functions for dimension reduction and image reconstruction," *IEEE Transactions on Systems, Man, and Cybernetics: Systems*, vol. 48, no. 7, pp. 1065–1079, 2016.

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

[10] P. Baldi, "Autoencoders, unsupervised learning, and deep architectures," in *Proceedings of ICML workshop on unsupervised and transfer learning*. JMLR Workshop and Conference Proceedings, 2012, pp. 37–49.

[11] H. Liu, Y. Wang, W. Fan, X. Liu, Y. Li, S. Jain, Y. Liu, A. Jain, and J. Tang, "Trustworthy ai: A computational perspective," *ACM Transactions on Intelligent Systems and Technology*, vol. 14, no. 1, pp. 1–59, 2022.

[12] C. Molnar, *Interpretable machine learning*. Lulu. com, 2020.

[13] S. Geman, E. Bienenstock, and R. Doursat, "Neural networks and the bias/variance dilemma," *Neural computation*, vol. 4, no. 1, pp. 1–58, 1992.

[14] D. L. Aguilar, M. A. Medina-Pérez, O. Loyola-Gonzalez, K.-K. R. Choo, and E. Bucheli-Susarrey, "Towards an interpretable autoencoder: A decision-tree-based autoencoder and its application in anomaly detection," *IEEE transactions on dependable and secure computing*, vol. 20, no. 2, pp. 1048–1059, 2022.

[15] S. M. Shankaranarayana and D. Runje, "Alime: Autoencoder based approach for local interpretability," in *Intelligent Data Engineering and Automated Learning–IDEAL 2019: 20th International Conference, Manchester, UK, November 14–16, 2019, Proceedings, Part I 20*. Springer, 2019, pp. 454–463.

[16] M. T. Ribeiro, S. Singh, and 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.

[17] C. Olah, A. Mordvintsev, and L. Schubert, "Feature visualization," *Distill*, vol. 2, no. 11, p. e7, 2017.

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

[19] W. Chang, J. Cheng, J. Allaire, C. Sievert, B. Schloerke, Y. Xie, J. Allen, J. McPherson, A. Dipert, and B. Borges, *shiny: Web Application Framework for R*, 2023, r package version 1.7.5. [Online]. Available: https://CRAN.R-project.org/package=shiny

[20] J. Allaire and F. Chollet, *keras: R Interface to 'Keras'*, 2023, r package version 2.13.0. [Online]. Available: https://CRAN.R-project.org/package=keras

[21] J. Allaire and Y. Tang, *tensorflow: R Interface to 'TensorFlow'*, 2023, r package version 2.13.0. [Online]. Available: https://CRAN.R-project.org/package=tensorflow

[22] R Core Team, *R: A Language and Environment for Statistical Computing*, R Foundation for Statistical Computing, Vienna, Austria, 2023. [Online]. Available: https://www.R-project.org/

[23] L. Yann, "The mnist database of handwritten digits," *R*, 1998.

[24] L. Breiman, "Random forests," *Machine learning*, vol. 45, pp. 5–32, 2001.

[25] G. Cohen, S. Afshar, J. Tapson, and A. Van Schaik, "Emnist: Extending mnist to handwritten letters," in *2017 international joint conference on neural networks (IJCNN)*. IEEE, 2017, pp. 2921–2926.

[26] H. Xiao, K. Rasul, and R. Vollgraf, "Fashion-mnist: a novel image dataset for benchmarking machine learning algorithms," *arXiv preprint arXiv:1708.07747*, 2017.

[27] S. Becker, J. Vielhaben, M. Ackermann, K.-R. Müller, S. Lapuschkin, and W. Samek, "Audiomnist: Exploring explainable artificial intelligence for audio analysis on a simple benchmark," *Journal of the Franklin Institute*, 2023. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0016003223007536

[28] L. McInnes, J. Healy, and J. Melville, "Umap: Uniform manifold approximation and projection for dimension reduction," *arXiv preprint arXiv:1802.03426*, 2018.