
Analysis of Neural Networks with Linearized Layer Transformations

Lab Report - Frederic Becker
Supervision: Lee Sharkey and Dr. Charley Wu

Abstract

Deep neural networks provide impressive state-of-the-art performance while remaining hardly interpretable [Zhang and Zhu, 2018, Zhang et al., 2021, Erhan et al., 2009]. Providing interpretability by analysing neural network weights is an open research area that is characterised by three main challenges: non-linear (i.e. the computation depends on the input), high-dimensional (i.e. the computation overwhelms our understanding) and context free (i.e. the computation is performed on non-interpretable latent codes). In response to these challenges, we propose, Analysis of Linearized Layer Transformations (ALLT), a novel method for analysing neural network weights. ALLT combines linearization, dimensionality reduction and feature visualisation techniques to make network computations interpretable. We apply our method to a multilayer classifier and validate it by successfully predicting network activity. Finally, we show that ALLT can be used to infer a layerwise classification trace which successfully captures computational differences between correctly and incorrectly classified inputs.

1 Introduction

Deep neural networks provide impressive state-of-the-art performance in a variety of tasks, including language modelling [Vaswani et al., 2017], image classification [He et al., 2016] and game play [Silver et al., 2017]. Even though the network architecture is human-designed, the learned network parameters are fully accessible, and all network activation is observable, we lack a true understanding of why a neural network produces the output it does [Zhang and Zhu, 2018, Zhang et al., 2021, Erhan et al., 2009]. We can explain how a neural network works in all detail, but we fail at explaining why it works how it works. Despite the clear superiority of deep architectures, the lack of interpretability has become their main drawback which increasingly limits their usability.

Interpretability of neural networks is crucial to ensure that intelligent systems operate reliably and safely. For instance, a roadsafe self-driving vehicle must reliably perceive its surrounding. Recent research has shown that perception systems based on deep networks are vulnerable to adversarial attacks [Ranjan et al., 2019]. The presentation of a small manipulated image resulted in a catastrophic failure of the system. Until such failures can be eliminated, the use of safety-critical systems in the real world seems unrealistic. Ranjan et al. [2019] showed that feature visualisation tools can help to understand and study these vulnerabilities. In addition, intelligent systems must meet regulatory requirements, such as the European Commission’s proposed Artificial Intelligence Act¹, which demand more transparent systems. Interpretability tools are needed to provide transparency, for example, by providing insights into how a decision system arrives at a decision [Camarata et al., 2021].

To understand the behavior of a neural network, it is most common to look at its weights [Voss et al., 2021]. While the architecture of a network provides insight into what kind of operations the network can perform, the weights define how inputs are transformed from one layer to the next and can be

¹<https://artificialintelligenceact.eu>

understood as transformation instructions. Gaining meaningful insights from analysing weights is an open challenge because the described computation is non-linear (i.e. the computation depends on the input), high-dimensional (i.e. the computation overwhelms our understanding) and context free (i.e. the computation is performed on non-interpretable latent codes). In direct response to these three challenges, we propose a new method, called Analysis of Linearized Layer Transformations (ALLT), that combines techniques of linearization, dimensionality reduction and feature visualisation to make weights interpretable.

Our work will make three contributions. First we propose a novel method for understanding the computations learned by a neural network. Second, we validate our method by showing that it is capable of predicting network activity. Third, we apply our method to a multilayer classifier and show that our method provides an interpretable trace of the classification process across layers.

2 Challenges in interpreting deep network weights

In this section, we present three aspects for why the interpretation of neural network weights is especially challenging and how our novel interpretation pipeline addresses these challenges.

The first challenge is the non-linearity of layer mappings. Each layer mapping is composed of a linear part (i.e. the weight tensor) and a non-linear part (i.e. the activation function). Due to the non-linear part the layer transformation is dependent of the input (i.e. the layer transformation may not be the same for all inputs). This non-linearity in the transformations significantly complicates the analysis of layer mappings. A simple but incomplete answer to this issue is to assume that the non-linear part is not decisive for the layer interpretation and can thus be neglected [Voss et al., 2021]. Building on Balestriero and Richard Baraniuk [2018], our approach integrates both parts into a single linear transformation that is conditioned on the input. By obtaining many of these linear transformations for a diverse set of inputs, we receive a manifold of linear transformations which captures the non-linear influence of the activation function. As in Balestriero and Richard Baraniuk [2018], our method makes use of this manifold to illustrate and interpret the non-linearity of the neural network.

The second challenge is the curse of dimensionality [Voss et al., 2021]. Layers often implement high-dimensional mappings by making use of thousands of weights. The vast amount of weights simply overloads human understanding. It is therefore essential to reduce or compress the information content of the mapping to a scale that can be interpreted by humans. Recent approaches [Voss et al., 2021, Lee et al., 2007] have addressed this issue by only analysing the strongest connections within a layer. This approach however neglects the majority of connections and is thus not able to provide a full picture. We will apply dimensionality reduction to compress the weights into a few expressive components and demonstrate that there is no significant loss of precision. In accordance with Geva et al. [2020] who found that feed-forward networks can act as key-value memories, our decomposition approach interprets a layer as a battery of feature detectors and their projections to the next layer.

The last challenge is a lack of context [Voss et al., 2021]. The weights of a single layer describe how an input is mapped to an output. To obtain an interpretable description of this mapping, it is important that the input and output spaces are human interpretable. For example, if the layer takes an image and outputs the mirrored image, the calculation is relatively easy to understand. However, if the layer takes an embedding of the image and outputs the mirrored embedding, the calculation is difficult to understand, as we lack an understanding of the context (i.e. the embedding space). In practice, this lack of context is foremost a problem when interpreting weights of hidden layers. Our method provides context by visualising features of the hidden space in the input space.

The most prominent visualisation technique is activation maximization [Erhan et al., 2009, Yosinski et al., 2015]. This technique infers the input pattern that maximally activates a hidden neuron by gradient based optimisation. The obtained pattern thus illustrates what the activation of the hidden unit may represent within the network. Another visualisation technique composes features of deep layers recursively by a linear combination of the features of the previous layers [Erhan et al., 2009, Lee et al., 2007]. The features of the first layer are given by the weights of the first layer. Based on the weights of the next layer, these features are then linearly combined to construct the features of the next layer. Both methods are typically used to visualise features of single neurons, while recent work [Fong and Vedaldi, 2018] suggests that meaningful concepts, which we ideally seek for interpretation, are represented in a distributed manner within a network. Our method leverages the

linear combination approach but improves on this by visualising features of population codes (i.e. components of the decomposition) rather than single neurons.

Having outlined the challenges of interpreting deep network weights and how our method will address these challenges, the next section will introduce the method in detail. In section 4 we apply our method to a multilayer network and validate the method by predicting network activity. In the following section, we provide interpretations for the network’s classification process. Lastly we discuss our findings in Section 6 and give an outlook on future research directions.

3 Analysis of Linearized Layer Transformations

ALLT consists of three central steps each addressing one of the challenges outlined in the previous section. We first briefly summarize the method and then describe each step in detail in the following subsections.

The first step (see Figure 1A) addresses the problem of non-linearity. Therefore, the input to a network is fixed to a specific value and the transformation each layer performs on the fixed input is described by a linear but still high-dimensional transformation matrix. Repeating this procedure for a diverse set of inputs, yields a set of linear transformation matrices for each layer of the network. The second step (see Figure 1B) addresses the problem of complexity by applying truncated Singular Value Decomposition [Golub and Reinsch, 1971] on the linear transformation matrices of a layer. This decomposition allows us to intuitively understand the layer mapping as a small battery of feature detectors over the layer input as well as their projection onto the output. The third step (see Figure 1C) aims at increasing the interpretability of hidden feature detectors by visualising their features in the input space of the network.

In the following, steps 1 and 2 are explained for single layers to set up intuitions, and in step 3 we will extend our approach to multilayer networks. Our method is described with the example of fully connected feed-forward layers, however in general our method can be extended to convolutional layers and recurrent layers [Balestriero and richard baraniuk, 2018].

3.1 Deriving Linear Transformations

Let’s consider a fully connected layer f that maps an input \mathbf{x} to a one dimensional output y ,

$$y = \sigma(\mathbf{w}\mathbf{x} + b),$$

where σ is a non-linear activation function, b is the bias and \mathbf{w} a weight vector. For the sake of simplicity, we absorb bias b in the weight vector \mathbf{w} and include a bias neuron in the layer. When fixing the input \mathbf{x} to a specific value \mathbf{x}_i the non-linear activation function performs a simple scaling operation,

$$y_i = \sigma(\mathbf{w}\mathbf{x}_i) = s_i \cdot \mathbf{w}\mathbf{x}_i,$$

where the scaling factor is simply given by:

$$s_i := \frac{\sigma(\mathbf{w}\mathbf{x}_i)}{\mathbf{w}\mathbf{x}_i}.$$

Both the weight vector \mathbf{w} and the scaling factor s_i perform affine transformations on the input \mathbf{x}_i , hence they can be combined in a single linear transformation vector \mathbf{t}_i :

$$y_i = \sigma(\mathbf{w}\mathbf{x}_i) = \mathbf{t}_i\mathbf{x}_i,$$

where \mathbf{t}_i is obtained by scaling \mathbf{w} :

$$\mathbf{t}_i := s_i \cdot \mathbf{w}.$$

The scaling resembles how the net activation ($\mathbf{w}\mathbf{x}_i$) is scaled to the output (y_i) by the activation function. For a linear activation function the scaling is independent of the input resulting in the same transformation vector for each input \mathbf{x}_i . However, for a non-linear activation function, the scaling depends on the value of the net activation. A piece-wise linear activation function performs a different scaling in each linear region [Balestriero and richard baraniuk, 2018]. For example, the rectified linear unit (ReLU) activation function scales with a factor of 1 for a net activation that falls on the linear region (i.e., larger than zero), whereas the ReLU scales with a factor of 0 for a net activation that

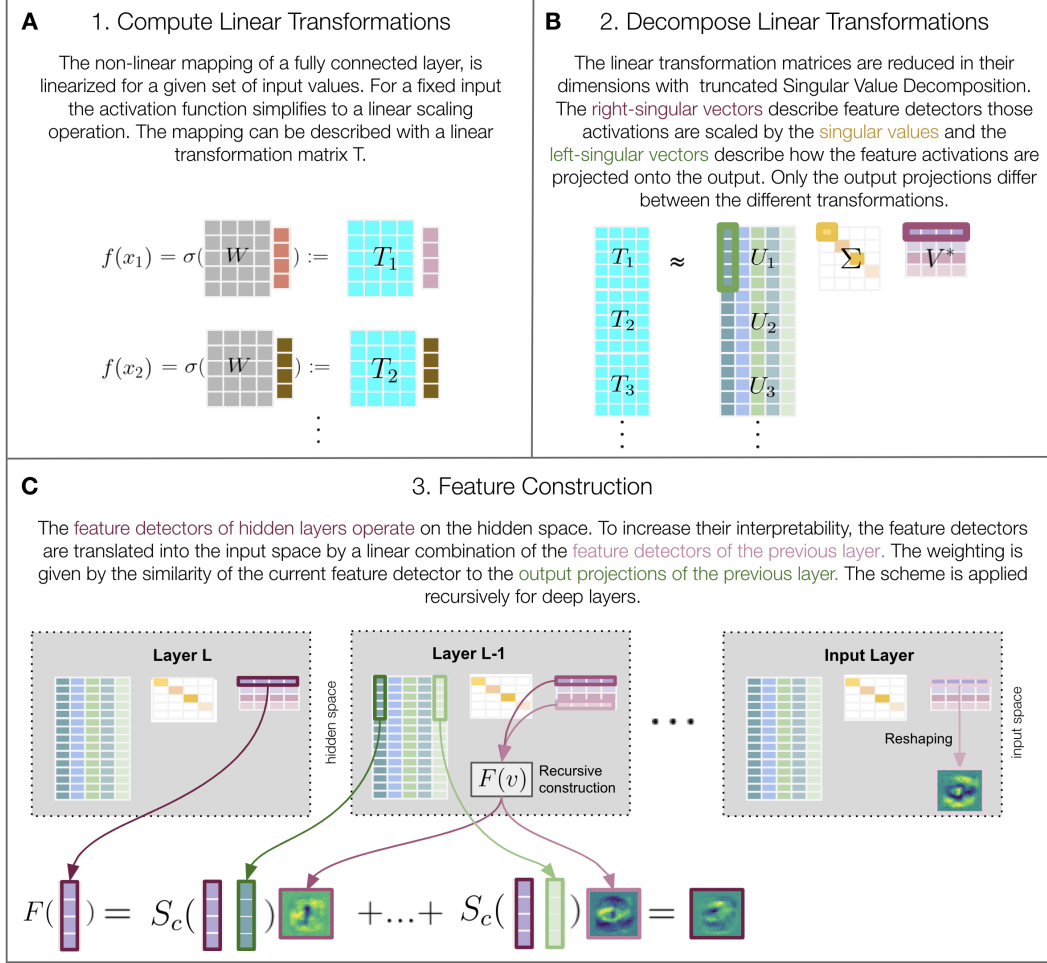


Figure 1: Illustration of the Linearized Layer Transformation presented in three successive stages.

falls on the saturated area (i.e., smaller than zero). A fully connected layer with a ReLU activation function thus performs two different linear transformations that can be described by $1 \cdot \mathbf{w}$ and $0 \cdot \mathbf{w}$. Our computation can be easily extended to fully connected layers with multi-dimensional outputs producing an output vector \mathbf{x}_i . Instead of a single transformation t_i the layer performs a transformation for each output dimension. These transformations can be described in a single transformation matrix:

$$\mathbf{T}_i := \text{diag}(s_i) \mathbf{W},$$

where \mathbf{W} is a weight matrix. Instead of a single scaling factor s_i we now have a scaling factor for each output dimension described by the scaling vector

$$s_i := \frac{\sigma(\mathbf{W} \mathbf{x}_i)}{\mathbf{W} \mathbf{x}_i}.$$

Finally, we can describe the computation that a layer performs on a given set of inputs $X = \{\mathbf{x}_1, \dots, \mathbf{x}_n\}$ by the corresponding set of linear transformations $T = \{\mathbf{T}_1, \dots, \mathbf{T}_n\}$. This procedure provides a linear window to the nonlinear dynamics of the layer.

3.2 Decomposing Linear Transformations

In a second step, we reduce the dimensions of the transformation matrices by applying truncated Singular Value Decomposition (SVD) on the stacked transformation matrices,

$$\begin{bmatrix} \mathbf{T}_1 \\ \vdots \\ \mathbf{T}_n \end{bmatrix}_{(n*d_{out} \times d_{in})} \approx \begin{bmatrix} \mathbf{U}_1 \\ \vdots \\ \mathbf{U}_n \end{bmatrix}_{(n*d_{out} \times k)} \boldsymbol{\Sigma}_{(k \times k)} \mathbf{V}^T_{(k \times d_{in})}.$$

The columns of \mathbf{U}_i are called left-singular vectors, the rows of \mathbf{V}^T are called right-singular vectors and entries of the diagonal matrix $\boldsymbol{\Sigma}$ are called singular values. Truncation is performed by selecting the singular vectors with the largest k singular values. Each transformation matrix \mathbf{T}_i is now approximated by,

$$\mathbf{T}_i \approx \mathbf{U}_i \boldsymbol{\Sigma} \mathbf{V}^T.$$

Besides compression, a major advantage of this decomposition is that it provides an intuitive understanding of the linear transformations. Each right-singular vector is of size d_{in} and can be seen as a feature detector that describes a pattern over the input. For example, if the input to the layer is an image then the right-singular vectors are visual features. The activation of the feature detector is determined by the dot product of the input (e.g. image) and the right-singular vector (e.g. visual feature). The resulting scalar can be understood as the activation of the feature detector and describes how well the input fits the given feature. This activation is then scaled with the associated singular value. The associated left-singular vector is of size d_{out} and explicitly tells us how the activation of the associated feature detector is passed on to the output dimensions. For example, if the output of the layer are scores for different categories, as used in multi-class classification tasks, then the left-singular vector describes how strongly each category is excited by the activation of the corresponding feature detector. Overall, the decomposition can be understood as a battery of k feature detectors and their projections onto the output.

This approach is fundamentally different from the analysis of the weight matrix. The rows of the weight matrices are also of size d_{in} and can also be understood as feature detectors. However, these features correspond to the receptive fields of single neurons, whereas the right-singular vectors correspond to one of many receptive fields of the entire layer. The decomposition thus allows us to analyse feature detectors of an entire layer instead of feature detectors of single neurons.

Moreover, applying SVD on the stacked transformation matrices instead of applying SVD on every transformation matrix individually comes with the advantage that the singular values and the right-singular vectors are constant across all individual decomposition describing the transformations performed on the different input samples. The variation of the transformation matrices are thus solely represented in the left-singular vectors i.e. the variation in the set $\{\mathbf{U}_1, \dots, \mathbf{U}_n\}$ captures the non-linear dynamics of the activation function. Our approach thus separates the constant and varying components of the layer computation. In turn, applying SVD on the stacked transformation matrices limits the flexibility of the factorization and thereby lowers the reconstruction accuracy. In practice, however, this loss is negligible since the variation of the transformation matrices is rather small because they all derive from the same weight matrix and only differ in their scaling factors.

So far we have only considered a single layer. We now extend the procedure to multilayer perceptrons. For this, we consider a feed-forward network with d fully connected layers and widths: w_0, w_1, \dots, w_d , where w_0 corresponds to the input dimension, w_1 to w_{d-1} denote the dimensions of the hidden layers and w_d corresponds to the output dimension. The neural network learns a mapping

$$\mathbf{y} := h(\mathbf{x}),$$

which can be written as a composition of non-linear mappings,

$$f(\mathbf{x}) = (f^{(d)} \circ \dots \circ f^{(2)} \circ f^{(1)})(\mathbf{x}),$$

where \mathbf{x} is first passed to the input layer $f^{(1)}$ and the output is then passed on to the next layer $f^{(2)}$ and so on. When performing a forward pass of a given set of inputs $X^{(0)} = \{\mathbf{x}_1^{(0)}, \dots, \mathbf{x}_n^{(0)}\}$ we obtain layer activations $X^{(1)}$ to $X^{(d)}$. As described above, these can be used to compute a set of linear

transformations matrices for each layer i.e. $T^{(1)}$ to $T^{(d)}$. When decomposing the linear transformation matrices of each layer we obtain the triplets $(U^{(1)}, \Sigma^{(1)}, V^{T(1)})$ to $(U^{(d)}, \Sigma^{(d)}, V^{T(d)})$, where each U is a set of matrices. The triplets describe a series of computations where the dimension of the left-singular vectors of a layer match the dimension of the right-singular vectors of the following layer.

The obtained triplets give us the opportunity to understand the individual layers of the neural network in the intuitive concept of feature detectors and output projections.

3.3 Feature construction

As described above, the right-singular vectors can be interpreted as feature detectors. The feature detectors of the input layer act directly on the input space making them relatively easy to interpret. For example, if the input space describes an image, then the right-singular vectors are spatial features. Thus, the features of the input layer are directly given by the corresponding right-singular vectors,

$$F(\mathbf{v}_i^{(1)}) = \mathbf{v}_i^{(1)}.$$

However, this is not the case for deeper layers where the feature detectors operate in the space of the hidden dimensions. Interpretation of the hidden space is usually difficult as its structure is self-organised by the network in training, which also makes the feature detectors difficult to interpret. To provide an intuitive interpretation of these feature detectors, we relate them to the input space by the means of a recursive matching scheme.

Let's consider the j^{th} right-singular vector of layer l that is $\mathbf{v}_j^{(l)}$. We create its corresponding feature by a linear combination of the features of the previous layer where the weighting is determined by the similarity of the feature detector $\mathbf{v}_j^{(l)}$ to the output projections of the previous layer $U^{(l-1)}$:

$$F(\mathbf{v}_j^{(l)}) = \sum_d^k S_c(\mathbf{v}_j^{(l)}, \mathbf{u}_d^{(l-1)}) * F(\mathbf{v}_d^{(l-1)}),$$

where S_c denotes the cosine similarity $S_c(\mathbf{a}, \mathbf{b}) = \frac{\mathbf{a} \cdot \mathbf{b}}{\|\mathbf{a}\| \|\mathbf{b}\|}$. The function F constructs the feature of the right-singular vector $\mathbf{v}_j^{(l)}$ of layer l by a weighted linear combination of the features of the previous layer (i.e. $F(\mathbf{v}_1^{(l-1)})$, ..., $F(\mathbf{v}_k^{(l-1)})$), where the weights are given by the similarity of the right-singular vector $\mathbf{v}_j^{(l)}$ and the corresponding left-singular vector of the previous layer $\mathbf{u}_d^{(l-1)}$.

Let's assume we want to visualise features of the first hidden layer. Intuitively, we approximate the pattern the feature detector looks out for in the hidden space (e.g. 0101) by the patterns the input layer projects into it (e.g. 0100 and 0001). Luckily, for the input layer we already know which features (e.g. a vertical bar and a horizontal bar) are responsible for the activation of the projection patterns. This allows us to simply combine these features to obtain the feature of the hidden feature detector (e.g. a cross). We can now use the constructed features of the first hidden layer to construct the features of the second hidden layer and so on.

The feature construction process is specific to an input x_i because the left-singular vectors are specific to it. To construct the feature of the j^{th} right-singular vector of the i^{th} input sample we can use a modified version of the previous equation

$$F_i(\mathbf{v}_j^{(l)}) = \sum_d^k S_c(\mathbf{v}_j^{(l)}, \mathbf{u}_{d,i}^{(l-1)}) * F_i(\mathbf{v}_d^{(l-1)}),$$

in which the only difference is the additional subscript i that specifies the input index i .

Finally, the feature construction process yields a set of k features for every layer and every sample, allowing us to visualise the feature detectors in the input space.

4 Method Application and Validation

In this section we apply ALLT on a neural network that was trained on a classification task. We show that the reduced transformations provide a good approximation of the full layer transformations.

In addition, we validate that the feature construction process works accurately by showing that the visual features can be used to reconstruct network activity.

4.1 Network and Training

We analysed a neural network consisting of four fully-connected layers with dimensions 1024, 1024, 512 and 10. The first three layers used a ReLu activation function and the output layer used a Softmax activation function. The network was optimised with stochastic gradient descent and dropout was applied during training. The neural network was trained on the MNIST handwritten digits dataset consisting of 10 classes one for each digit [Deng, 2012]. The training set contained of 60,000 images and the test set 10,000 images. The classification accuracy for the test set after training was 94.7%.

4.2 Choice of Hyperparameters

The query set, that was used to derive the linear transformations, consisted of 1000 samples of the MNIST test set, using 100 samples per class. The decomposition was performed with truncated randomized SVD [Halko et al., 2009]. The number of components k which determine the compression rate was determined by the elbow heuristic. Therefore we plotted the largest 50 singular values of the decomposition and determined the elbow point of the function by visual inspection. The number of components per layer calculated in this way was 20, 10, 10 and 10.

4.3 Validation of Linear Transformation Decomposition

To ensure that the truncated decompositions sufficiently approximate the full transformation matrices, we calculated a SVD reconstruction score for each layer. The decomposition is evaluated by its accuracy of reconstructing the activation of layer l when processing a sample of the query set i.e. $\mathbf{x}^{(l)}$. The reconstruction is calculated by multiplying the layer input with the transformation decomposition of the layer $\hat{\mathbf{x}}^{(l)} = [\mathbf{U}\Sigma\mathbf{V}^T]^l \mathbf{x}^{(l-1)}$. Each sample i of the query set is associated with an individual decomposition (i.e. $\mathbf{U}_i\Sigma\mathbf{V}^T$), which is used for the respective reconstruction calculation.

The Pearson correlation coefficient between the actual and the reconstructed activation was 0.92 for layer 1, 0.99 for layer 2 and 1.00 for layer 3 and 4. The mean absolute error between the actual and the reconstructed activation was 8.0 for layer 1, 4.0 for layer 2, 2.2 for layer 3 and 0 for layer 4. These numbers suggest that the high-dimensional transformations performed by a layer can indeed be described by only a few components without a significant loss in accuracy. This is remarkable as we reduce the dimensions of the transformation matrices by a factor of 50, 100, 50, 1 in the respective layers. The fact that the loss is highest in the first layer, may indicate that the most complex transformations are performed here.

4.4 Validation of Feature Construction

Next we verified that the feature construction process indeed yields accurate feature visualisations. The feature construction process translated feature detectors that described patterns of hidden layer activity into features in the input space. Because the feature detectors are associated with output projections of their layer, the constructed features should directly encode patterns of hidden layer activity. To test this, we calculated a Feature reconstruction score for each layer (see Figure 2). The features are evaluated by their accuracy of reconstructing the activation of layer l when processing a sample of the query set i.e. $\mathbf{x}^{(l)}$. The reconstruction is calculated by the weighted sum of output projections,

$$\hat{\mathbf{x}}^{(l)} = \sum_{i=1}^k \mathbf{u}_i^{(l)} * \sigma_i * (\mathbf{F}(\mathbf{v}_i^{(l)}) \cdot \mathbf{x}^{(0)})$$

where the weighting is given by the similarity of the constructed features $\mathbf{F}(\mathbf{v}_i^{(l)})$ and the input query $\mathbf{x}^{(0)}$ scaled with the respective singular value σ_i . The rationale underlying this equation is that the better the input query matches a feature, the more the activation of the hidden layer should be shaped by the associated activity pattern.

The Pearson correlation coefficient between the actual and the reconstructed activation was 0.92 for layer 1, 0.97 for layer 2 and 3 and 0.98 for layer 4. These results suggest the constructed features can

indeed be interpreting as visual features that encode activity patterns of hidden layers. Further, there seems to only little to none loss in accuracy when translating the features from the hidden space to the input space. The mean absolute error between the actual and the reconstructed activation was 8.0 for layer 1, 7.5 for layer 2, 38.6 for layer 3 and 0.1 for layer 4. This error arises because the reconstructed activation differs from the actual activation by a constant factor in deeper layers. The reason for this difference is yet not fully understood and requires further investigations. However, as we are mainly interested in the coding of activity patterns rather than absolute differences we can still conclude that the feature construction process yields accurate feature visualisations.

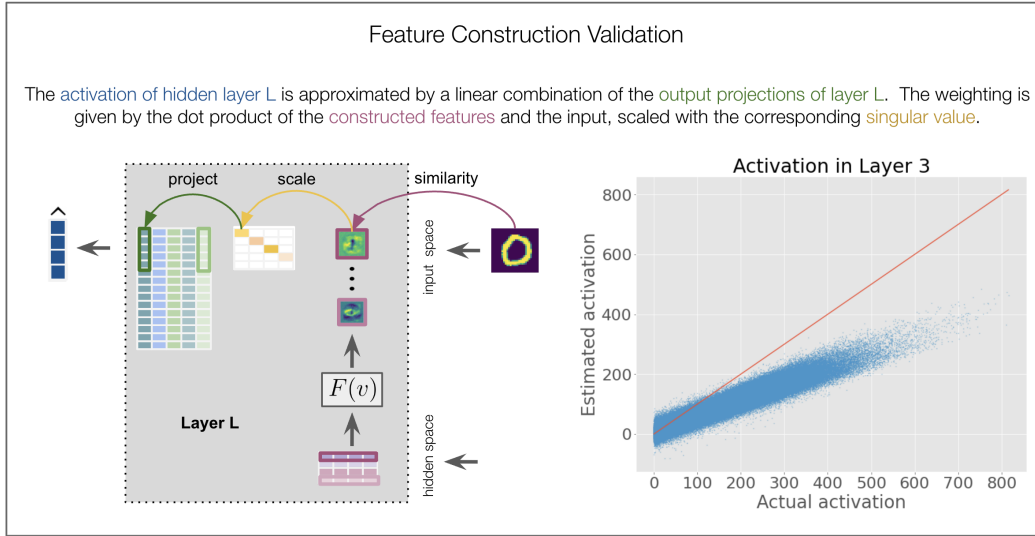


Figure 2: Validation of the feature construction process.

5 Empirical Analysis

Having demonstrated that our method provides both accurate layer transformations and accurate feature visualizations, we now use both to interpret the inner workings of the neural network introduced in the previous section.

5.1 Neural networks learn input-dependent linear transformations

As described in section 3.1, our method captures the non-linearity of the layer mapping in the variation of the output projections i.e., the variation in the set $\{\mathbf{U}_1, \dots, \mathbf{U}_n\}$. This allows us to visualise the non-linear influence by illustrating the variation of the output projections. Figure 3 is doing just this, showing the t-SNE embedding [van der Maaten and Hinton, 2008] of the output projections of each layer. Each datapoint represents the linear transformation of a single query input in the respective layer.

The visualisations suggest that the network has learned input-dependent linear transformations. The embedding shows that the transformations of inputs of the same class are similar and group into clusters. These clusters are more well defined in deeper layers suggesting that the transformations become more class specific in deeper layers i.e. more distinct between classes and more similar within classes. The non-linearity seems to enable the network to learn different modes of transformations for each input class. In addition, the illustration suggests that the transformations applied to misclassified inputs diverge from the class-specific transformations after a certain layer.

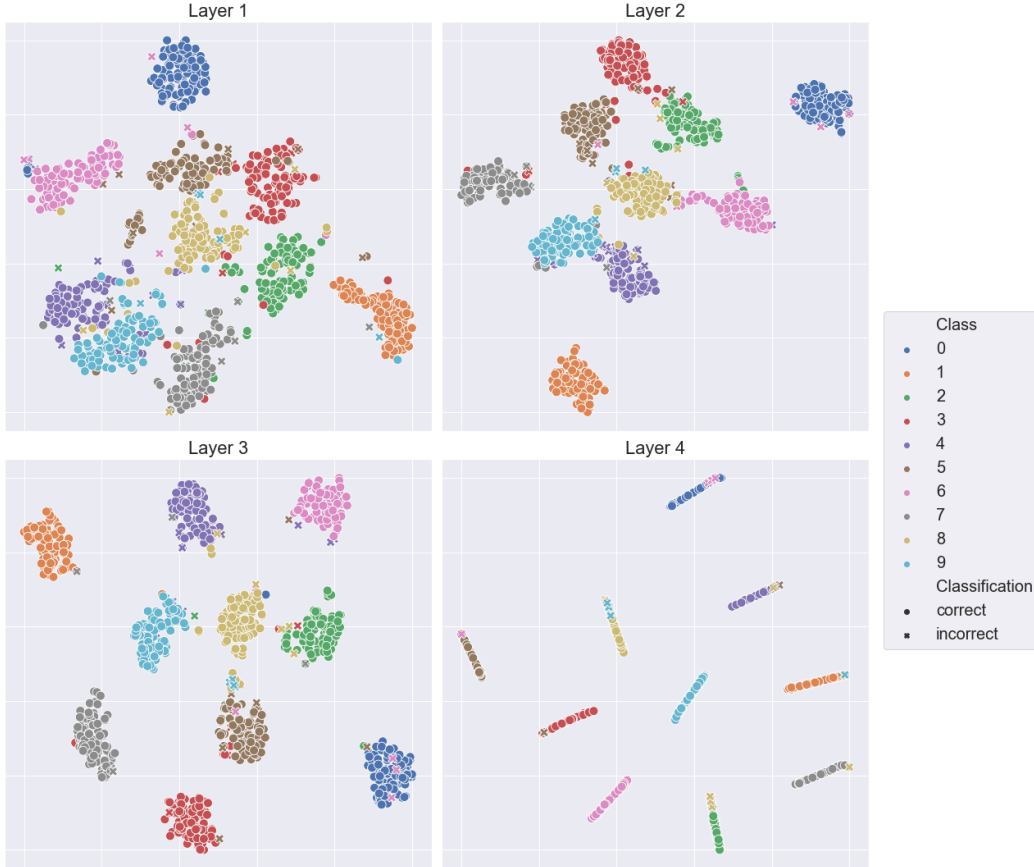


Figure 3: Visualisation of the variation of the output projections depicting the non-linear manifold of the layer mapping. The output projection matrices are embedded in a two dimensional space using t-SNE. Each datapoint depicts the output projection of a transformation of a query input in the corresponding layer. The datapoints are colored based on their ground truth labels.

5.2 Decision Traces

In the last subsection we have seen that the transformation landscape is input dependent and that this landscape is divided into different regions. Furthermore we have seen that these regions are mainly characterized by the input classes. We now use this fact to interpret the network’s layer-wise decision process. To do so, we first derive the average transformation for each input class, by calculating the euclidean center of the transformations belonging to one class. In a second step we assign the transformations of the input queries to its most similar class centroid based on the Euclidean distance. As a result each transformation is assigned to one of ten input classes. This grouping bins the transformation landscape based on classes. Because the transformation landscape is input dependent, we can interpret this grouping as an indirect binning of the input space. Finally, we obtain a preliminary classification for each query input and layer that we will refer to as *decision trace*.

Decision traces are interpretable descriptions of the layerwise classification process of the network. Table 1 shows decision traces of three exemplary query inputs. The first trace indicates that the network switched between two competing classes, and the second trace shows a misclassification without switching.

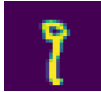
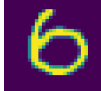

Input	Decision Trace				Output	Label
	Layer 1	Layer 2	Layer 3	Layer 4		
	1	9	9	1	1	9
	0	0	0	0	0	6
	7	7	7	7	7	7

Table 1: Decision traces of query inputs. The decision trace describes in which transformation region the transformation of the corresponding query input is binned in. The final decision layer is marked in bold.

5.3 Layerwise Classification Analysis

Besides the qualitative interpretation of the decision traces, we found that there are systematic differences between decision traces of correctly and incorrectly classified query inputs. Therefore we analysed whether the preliminary decisions classifications were correct or incorrect. In addition we defined a decision layer as the layer after which a decision trace did not change anymore. These two statistics are presented per layer in Figure 4A and B.

For correctly classified inputs, we found that the network used the correct transformation in 91.3% of the time already at the first layer. Complementary, the decision for correctly classified inputs did not change in 90.0% of the time after the first layer. This means that the network identified the correct class and stayed with it in approximately nine out of ten cases. Only in one out of ten cases, the network leveraged deeper layers to correct its initial classification. These results suggest, that mainly the transformations of the first layer are decisive for a correct classification result.

For incorrectly classified inputs, we observe a different pattern. Here, the proportion of correct preliminary classifications ranges between 25% and 41.7% percent in layers 1 to 3. The majority of final decisions was taken in the last layer (45.8%). This indicates, that the network made partially correct judgments throughout all layers and that there were frequent changes of this decision especially in the very last layer.

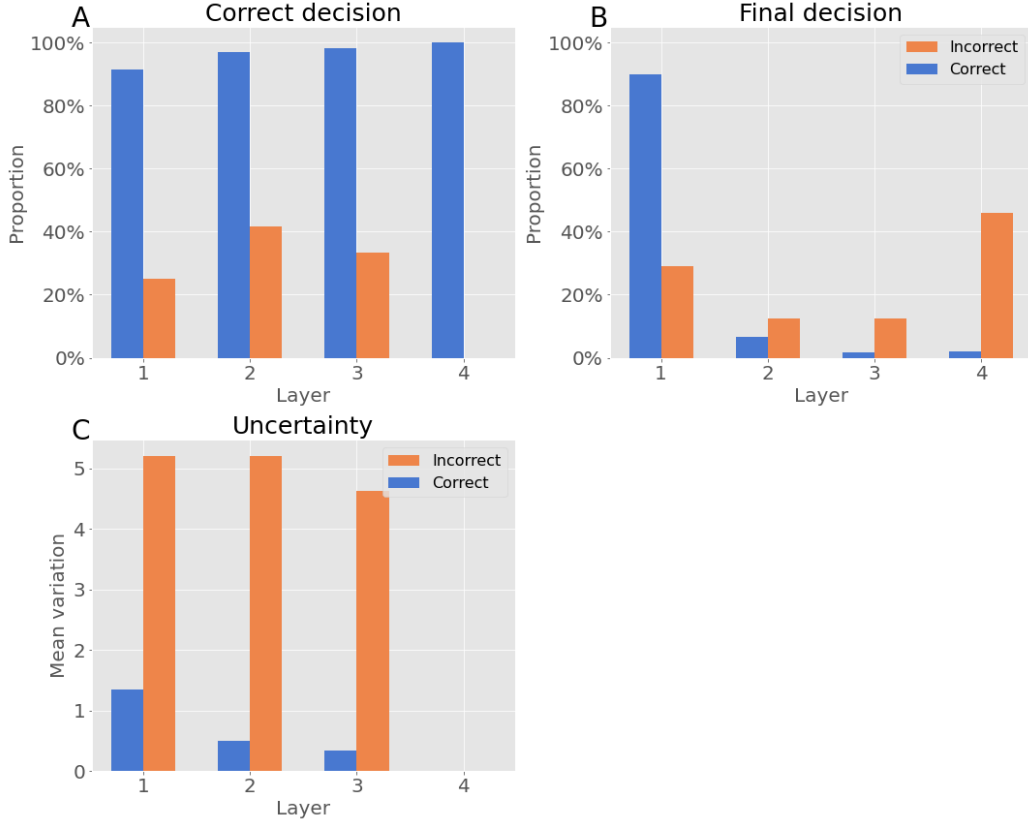


Figure 4: Incorrectly classified inputs have a systematically different decision trace than correctly classified inputs. **A:** The final decision layer presented per classification correctness. **B:** The variation in the extended decision trace presented per classification correctness.

5.4 Layerwise Uncertainty Analysis

Next, we extend the decision traces to quantify the uncertainty of the layerwise classification process. Therefore, we apply the procedure described in the previous subsection (see 5.2) on each of the k dimensions of the output projection matrix separately. This again yields a decision trace, however, with the difference that the grouping is not described by a single class but rather by a profile of k classes e.g. instead of 1 it is the profile $[1,3,1,4,1,1,5]$. Each entry of the profile shows which class center of the respective sub-dimension the transformation is closest to. Finally we quantify the variation in the profiles by simply counting the number of unique entries. We assume that a transformation that is aligned with a class transformation on all sub dimensions marks a certain classification and that a transformation that is aligned with a variety of class transformations in the different sub dimensions is more uncertain.

As shown in Figure 4B, we found that the level of uncertainty is much higher for misclassified inputs across all layers. The mean uncertainty was 5.0 ($SD = 3.7$) for incorrectly classified inputs and 0.7 ($SD = 1.7$) for correctly classified inputs. A possible explanation is that incorrectly classified inputs show properties of different classes. As each sub dimension focuses on a different property via a different feature detectors, such a mixed input may be transformed differently in each sub dimension. In conclusion, we have shown how decision traces can be used to quantify uncertainty in a classification network.

5.5 Qualitative analysis of features visualisations

We now take a detailed look at the produced feature visualisations. Figure 5 shows the averaged feature visualisations of the query samples of classes 0,1 and 2 per layer. We observe three interesting

properties of the constructed features of layer 1. First, unlike features of convolutional neural networks, the features are not sparse and describe more complex patterns. Second, the sensitive area in the center of the features is well matched to the input distribution, since all inputs are centered. Third, the features do not describe noise but coherent patterns such as curve patterns (middle column).

In deeper layers, we see that the features become more differentiated between classes. Although the same hidden feature vector was constructed for all classes, the construction process resulted in different visualizations for each class because it used a separate set of output projections for each query input. Consequently, the differences are due to non-linearity. We note that the same activity pattern appears to encode multiple class-specific input patterns. Interestingly, these patterns adopt the form of the corresponding input classes which can be seen in particular in feature 8 of layer 4.

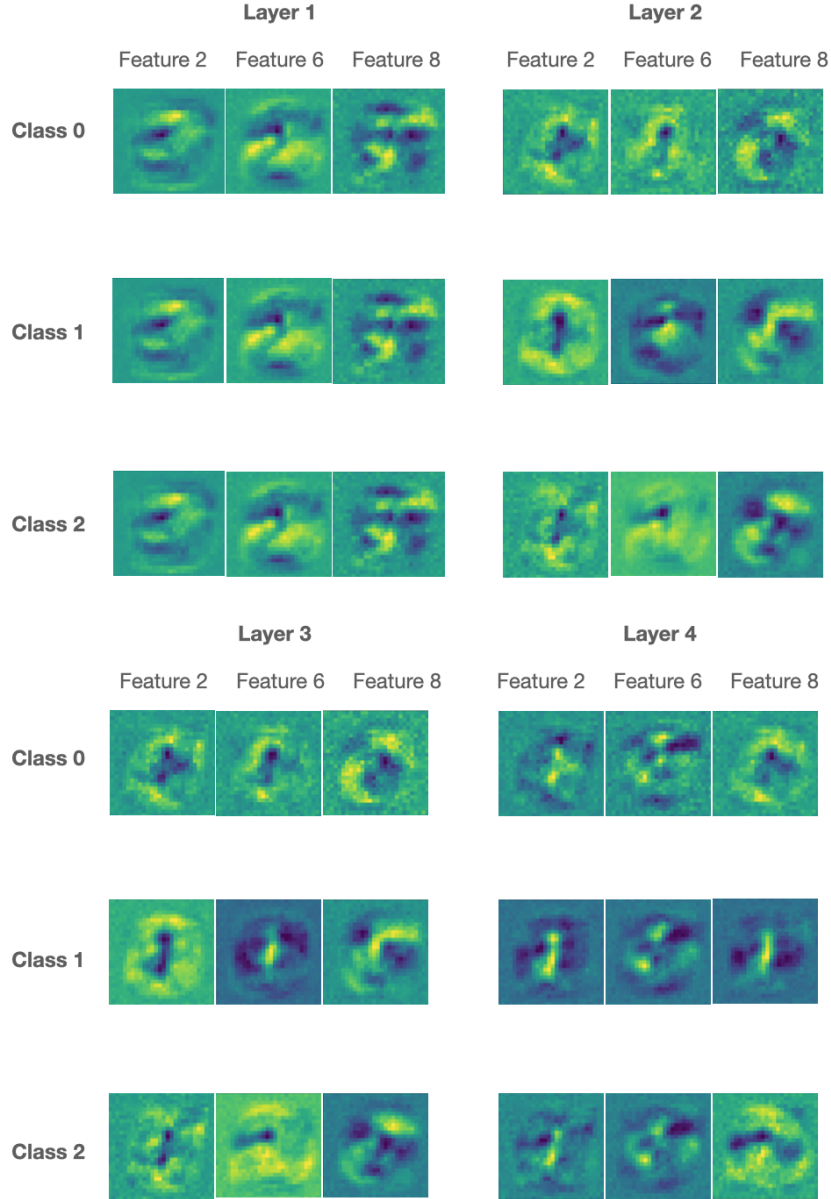


Figure 5: Feature visualisations for layers 1-4 and features 2,6 and 8. The visualisations were obtained by our recursive feature construction process. Class features were computed by averaging over the feature visualisations of all query inputs of that class.

6 Discussion and Future Work

We have proposed, Analysis of Linearized Layer Transformations (ALLT), a novel method for analysing neural network weights that combines linearization, dimensionality reduction and recursive feature visualisation. In the validation of our method, we demonstrated that the computations performed by a layer can indeed be described by only a few components without a significant loss in accuracy and that the feature visualisations encode activity patterns of hidden layers. Finally, we have shown that our method can be used to infer a layerwise classification trace which captures computational differences between correctly and incorrectly classified inputs.

In accordance with Balestrieri and richard baraniuk [2018], we found that multilayer perceptrons learn input dependent linear transformations that are organized primarily based on the different input classes. Further, we showed that fully-connected layers can be viewed as a battery of feature detectors with associated output projections. Like [Geva et al., 2020] who showed that feed-forward layers in the transformer model act as key-value memories, we emphasize that this interpretation can help in understanding the computations of neural networks. Similar to Fong and Vedaldi [2018], our results suggest that decoding combinations of activity patterns rather than single unit activities is a promising approach to explain the meaning of intermediate representations.

There are possible improvements to our method. First, the validation of the feature visualisations has revealed that the reconstruction of network activity deviates by a constant factor. This discrepancy should be addressed in future applications. Second, the existing pipeline can be easily modified to yield a fixed set of output projections while capturing the non-linearity in the variation of the feature detectors. This would allow to understand the computations in the context of the output space. In a next promising step, both directions could be combined to understand the computations in a complementary way. Third, the procedure used to determine the decision trace could potentially be improved. The current method finds structure in the transformation landscape based on ground truth classes. More complex models may learn intermediate representations that cannot be described with this grouping. Unsupervised clustering could remedy this by finding the inherent structure of the transformation landscape. Lastly, the uncertainty analysis may be extended to a Bayesian framework that could explicitly inform about the uncertainty of a network's decision.

In future work we plan to use ALLT to analyse more complex neural networks. Interesting candidates are encoder modules, as found in autoencoders or transformer models, which generate compressed representations of the input. ALLT could potentially provide insights into how these representations are computed.

References

- Randall Balestrieri and richard baraniuk. A spline theory of deep learning. In Jennifer Dy and Andreas Krause, editors, *Proceedings of the 35th International Conference on Machine Learning*, volume 80 of *Proceedings of Machine Learning Research*, pages 374–383. PMLR, 10–15 Jul 2018. URL <https://proceedings.mlr.press/v80/balestrieri18b.html>.
- Nick Cammarata, Gabriel Goh, Shan Carter, Chelsea Voss, Ludwig Schubert, and Chris Olah. Curve circuits. *Distill*, 2021. doi: 10.23915/distill.00024.006. <https://distill.pub/2020/circuits/curve-circuits>.
- Li Deng. The mnist database of handwritten digit images for machine learning research [best of the web]. *IEEE Signal Processing Magazine*, 29(6):141–142, 2012. doi: 10.1109/MSP.2012.2211477.
- Dumitru Erhan, Yoshua Bengio, Aaron Courville, and Pascal Vincent. Visualizing higher-layer features of a deep network. *University of Montreal*, 1341(3):1, 2009.
- Ruth Fong and Andrea Vedaldi. Net2vec: Quantifying and explaining how concepts are encoded by filters in deep neural networks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2018.
- Mor Geva, Roei Schuster, Jonathan Berant, and Omer Levy. Transformer feed-forward layers are key-value memories, 2020. URL <https://arxiv.org/abs/2012.14913>.
- G. H. Golub and C. Reinsch. *Singular Value Decomposition and Least Squares Solutions*, pages 134–151. Springer Berlin Heidelberg, Berlin, Heidelberg, 1971. ISBN 978-3-662-39778-7. doi: 10.1007/978-3-662-39778-7_10. URL https://doi.org/10.1007/978-3-662-39778-7_10.
- Nathan Halko, Per-Gunnar Martinsson, and Joel A. Tropp. Finding structure with randomness: Probabilistic algorithms for constructing approximate matrix decompositions. 2009. doi: 10.48550/ARXIV.0909.4061. URL <https://arxiv.org/abs/0909.4061>.
- Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2016.
- Honglak Lee, Chaitanya Ekanadham, and Andrew Ng. Sparse deep belief net model for visual area v2. In J. Platt, D. Koller, Y. Singer, and S. Roweis, editors, *Advances in Neural Information Processing Systems*, volume 20. Curran Associates, Inc., 2007. URL <https://proceedings.neurips.cc/paper/2007/file/4daa3db355ef2b0e64b472968cb70f0d-Paper.pdf>.
- Anurag Ranjan, Joel Janai, Andreas Geiger, and Michael J. Black. Attacking optical flow, 2019. URL <https://arxiv.org/abs/1910.10053>.
- David Silver, Thomas Hubert, Julian Schrittwieser, Ioannis Antonoglou, Matthew Lai, Arthur Guez, Marc Lanctot, Laurent Sifre, Dhharshan Kumaran, Thore Graepel, Timothy Lillicrap, Karen Simonyan, and Demis Hassabis. Mastering chess and shogi by self-play with a general reinforcement learning algorithm, 2017. URL <https://arxiv.org/abs/1712.01815>.
- LJP van der Maaten and GE Hinton. Visualizing high-dimensional data using t-sne. *Journal of Machine Learning Research*, 9(nov):2579–2605, 2008.
- Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. In I. Guyon, U. Von Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 30. Curran Associates, Inc., 2017. URL <https://proceedings.neurips.cc/paper/2017/file/3f5ee243547dee91fbd053c1c4a845aa-Paper.pdf>.
- Chelsea Voss, Nick Cammarata, Gabriel Goh, Michael Petrov, Ludwig Schubert, Ben Egan, Swee Kiat Lim, and Chris Olah. Visualizing weights. *Distill*, 2021. doi: 10.23915/distill.00024.007. <https://distill.pub/2020/circuits/visualizing-weights>.
- Jason Yosinski, Jeff Clune, Anh Nguyen, Thomas Fuchs, and Hod Lipson. Understanding neural networks through deep visualization, 2015. URL <https://arxiv.org/abs/1506.06579>.

- Quan-shi Zhang and Song-chun Zhu. Visual interpretability for deep learning: a survey. *Frontiers of Information Technology & Electronic Engineering*, 19(1):27–39, Jan 2018. ISSN 2095-9230. doi: 10.1631/FITEE.1700808. URL <https://doi.org/10.1631/FITEE.1700808>.
- Yu Zhang, Peter Tiño, Aleš Leonardis, and Ke Tang. A survey on neural network interpretability. *IEEE Transactions on Emerging Topics in Computational Intelligence*, 5(5):726–742, 2021. doi: 10.1109/TETCI.2021.3100641.