TreeView: Peeking into deep neural networks via feature-space partitioning

Jayaraman J. Thiagarajan, Bhavya Kailkhura, Prasanna Sattigeri, Karthikeyan Natesan Ramamurthy

Abstract

- With the advent of highly predictive but opaque deep learning models, it has become more important than ever to understand and explain the predictions of such models. Existing approaches define interpretability as the inverse of complexity and achieve interpretability at the cost of accuracy. This introduces a risk of producing interpretable but misleading explanations. As humans, we are prone to engage in this kind of behavior [1]. In this paper, we take a step in the direction of tackling the problem of interpretability without compromising the model accuracy. We propose to build a Treeview representation of the complex model via hierarchical partitioning of the feature space, which reveals the iterative rejection of unlikely class labels until the correct association is predicted

1. Interpretability in machine learning (skip)
2. Proposed approach

- In this section, we describe the proposed approach in the context of understanding features learned by a deep neural network. We consider only fully connected networks although it is possible to conceive of extensions to other architectures. Figure 1(a) illustrates a simple fully connected deep network with a softmax layer for class prediction. Unraveling the mechanics of the hidden layer representations can provide interesting insights into the trained model. This is a main distinction of our approach compared to other existing methods that attempt to learn a simpler surrogate (e.g. linear models) to explain the predictions for individual examples.

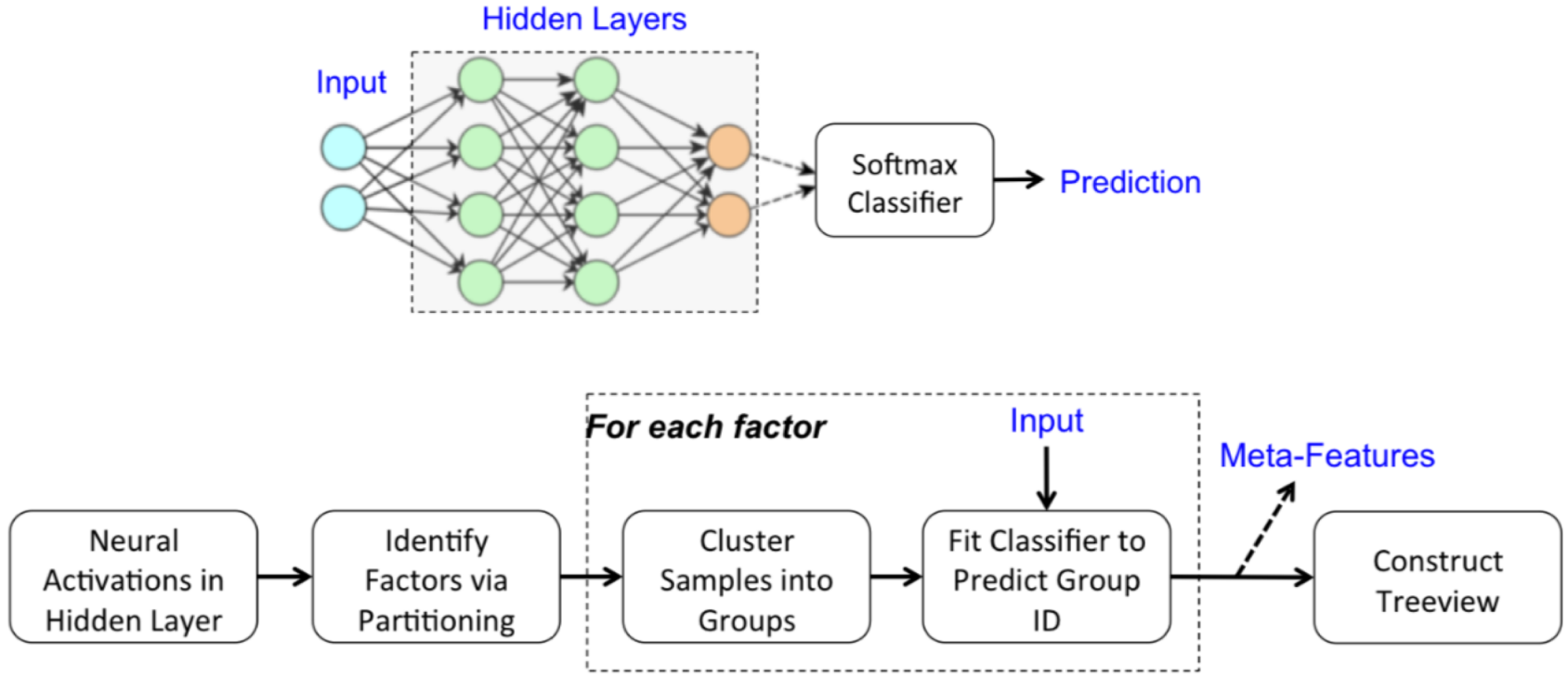


Figure 1: (Top) The hidden representation of even a simple fully connected deep neural network can be highly opaque for interpretation. (Bottom) Proposed approach for building *TreeView* visualizations using a surrogate model.

Figure 1(b) shows an overview of our approach. We decompose the feature space Y into K potentially overlapping factors?. Let us denote the feature dimensions corresponding to these factors using the sets {S1 , . . . , SK }. We obtain K different clusterings of samples, one for each of the subspaces of Y given by Si. We then construct a K−dimensional *meta-feature* for each sample as a collection of its K cluster labels?. The TreeView framework uses these meta-features used in a decision tree to create an easily interpretable visualization of the mechanics of the learned deep network.

2.1 identifying factors

The hidden neurons in a deep neural network learn different aspects of the training data samples that facilitate the task. The same aspect of the training data can be captured by different neurons because of the distributed nature of learned representations. Hence it is reasonable to expect that this feature space can be clustered into factors, each of which represents one aspect of the input space at some level of abstraction. We elaborate this by extending the notation provided before. For layer ℓ, the activations are given by the matrix $\mathbf{Y}_\ell \in \mathbb{R}^{N_\ell \times T}$, where N_ℓ and T are the number of filters and training samples respectively. We cluster $\mathbf{Y}_\ell$ into K factors, $\{\mathbf{F}_i^\ell\}_{i=1}^K$, such that each factor is comprised of a set of hidden neurons that have similar distribution of activations across the whole training set.

Actually the first time mentioning matrix notation: clustering of neurons

2.2 constructing meta-features

The second important step is to create meta-features that are easier to interpret, but still perform well in the learning task. We construct the surrogate model as follows: First, we consider the activations in each factor i, $\mathbf{F}_i^\ell \in \mathbb{R}^{N_i^\ell \times T}$, and cluster the T samples in this factor into L groups. We then create a meta-feature matrix $\mathbf{M} \in \mathbb{R}^{K \times T}$ by aggregating the K cluster labels for each sample. Subsequently, we train a predictor $\mathcal{P}_i^\ell$ that directly predicts the cluster label for a sample using the input space examples. In our case, the predictor used is a random forest classifier. This will enable the analyst to understand each factor using the input examples directly, and circumvent the need to train an approximate model (e.g. linear). A decision tree surrogate is finally created for the neural network classifier using the meta-features.

Actually the first time mentioning L: it is the categorical value for k th entry (L=2 in figure 2, 3)

TreeView Design

Consider a test example whose prediction we want to interpret using the *Treeview* visualization. Assuming that its true label is known, we compute the meta-feature using the predictors corresponding to each of the factors. This meta-feature is used with the decision tree surrogate to predict the label and the sequence of nodes visited in the decision tree during this prediction is traced. As shown in Figure 2, each column in a *TreeView* corresponds to the query if the sample belongs to a specific class, while the rows indicate the sequence of factors required to effectively reject the hypotheses and predict the correct label. We consider a hypothesis to be rejected if the activations corresponding to the factor ($\mathbf{F}_i^\ell$) clearly discriminates the true class and the hypothesis. In addition, the relative ranks of the input space features in the factor-specific predictor $\mathcal{P}_i^\ell$ are shown to allow the user to create a mental map between the class labels and input data.

Figure 2: L =2, K =4

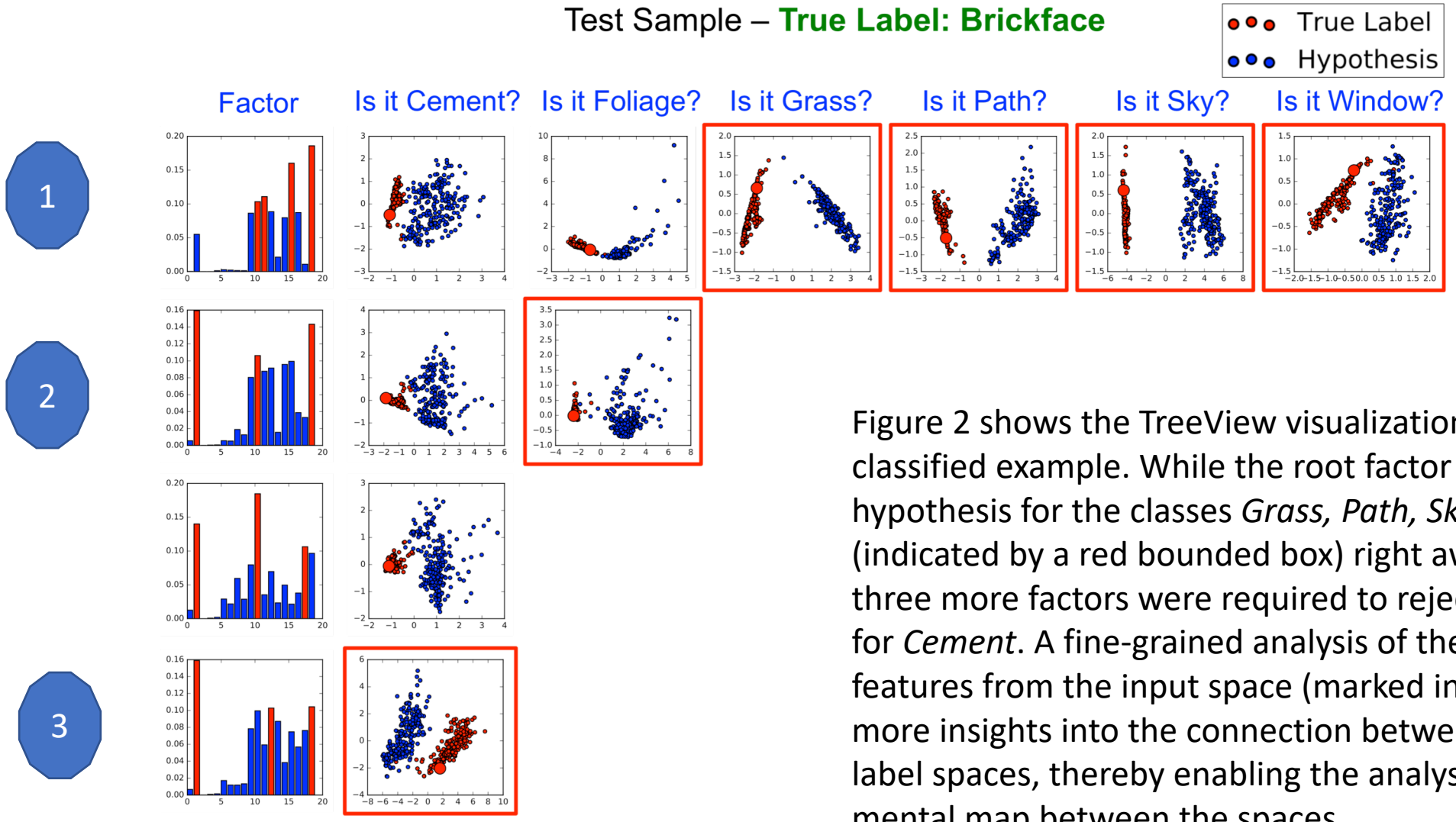
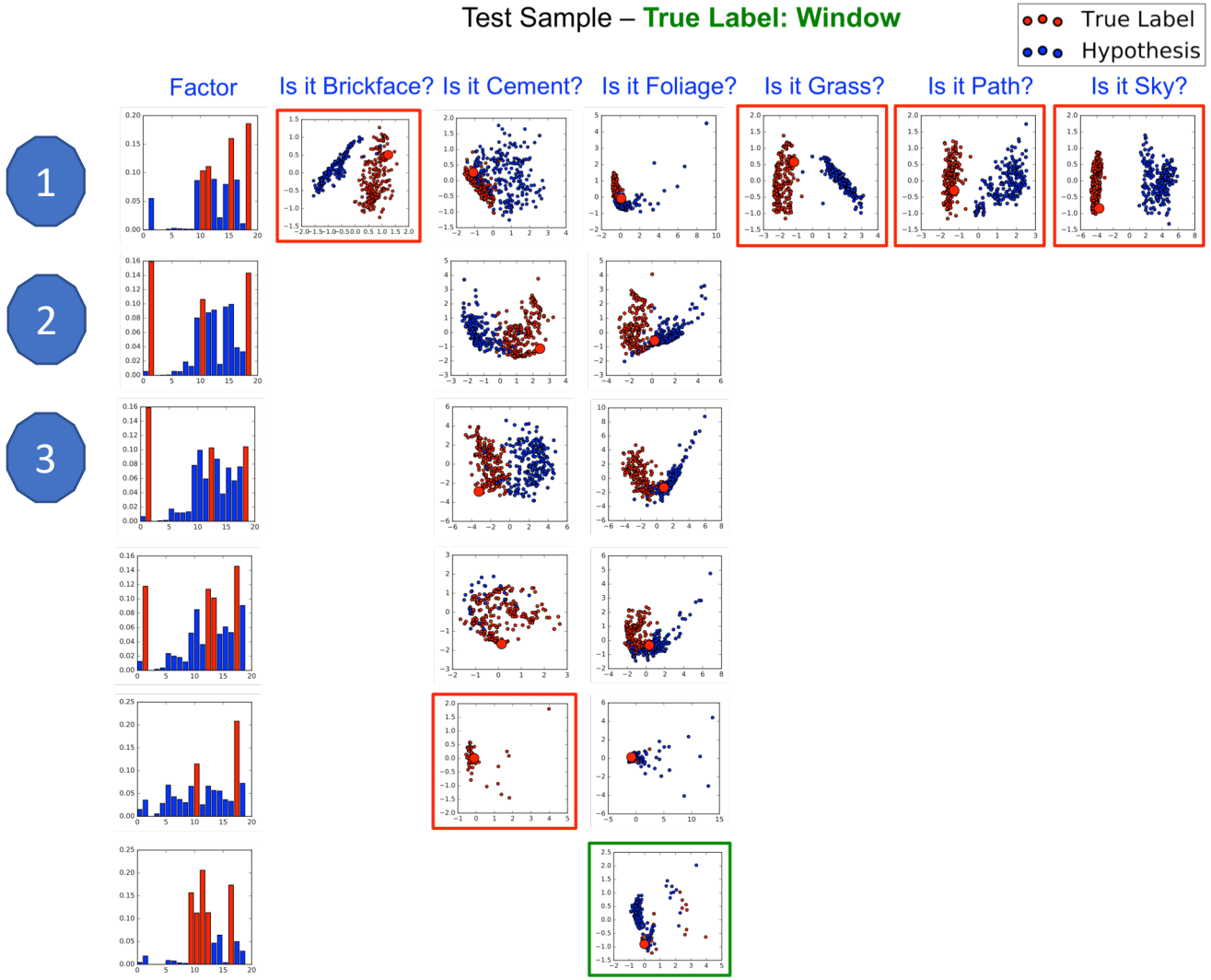


Figure 2 shows the TreeView visualization for a correctly classified example. While the root factor rejects the hypothesis for the classes *Grass, Path, Sky, Window* (indicated by a red bounded box) right away, a sequence of three more factors were required to reject the hypotheses for *Cement*. A fine-grained analysis of the high-ranked features from the input space (marked in red) can provide more insights into the connection between the input and label spaces, thereby enabling the analyst to validate their mental map between the spaces.

Figure 3: *Treeview* visualization for a sample which is wrongly classified by the neural network (L=2, K=6?)



In contrary, the negative example in Figure 3 illustrates a case where the factors identified by the network were unable to clearly discriminate between *Foliage* and the true class label *Window*. The *Treeview* visualization allows a convenient transition between factors, class labels, and the input data space, while staying relevant to the features inferred using the neural network.