Interpreting Individual Classifications of Hierarchical Networks

Will Landecker*, Michael D. Thomure*, Luís M. A. Bettencourt†‡,
Melanie Mitchell*†, Garrett T. Kenyon‡, and Steven P. Brumby‡
*Department of Computer Science
Portland State University
{landeckw,thomure,mm}@cs.pdx.edu
†Santa Fe Institute
bettencourt@santafe.edu
‡Los Alamos National Laboratory
{gkenyon, brumby}@lanl.gov

***Abstract*—Hierarchical networks are known to achieve high classification accuracy on difficult machine-learning tasks. For many applications, a clear explanation of *why* the data was classified a certain way is just as important as the classification itself. However, the complexity of hierarchical networks makes them ill-suited for existing explanation methods. We propose a new method, *contribution propagation*, that gives per-instance explanations of a trained network's classifications. We give theoretical foundations for the proposed method, and evaluate its correctness empirically. Finally, we use the resulting explanations to reveal unexpected behavior of networks that achieve high accuracy on visual object-recognition tasks using well-known data sets.**

I. Introduction

In machine learning, a model is learned from structure in the data. When using machine-learning techniques, one hopes that the learned structure generalizes well to yet-unseen data. However, many machine-learning techniques result in black-box models, making it difficult to understand the nature of the learned structure. With interpretable methods that explain the interactions between data and model, we can ensure that our learned models have learned *relevant* structure from the data, rather than spurious statistics.

This paper focuses on a family of hierarchical networks in which the operations computed at each layer alternate between some type of pattern matching (*e.g.,* convolution, radial basis function) and subsampling (*e.g.,* mean, maximum). Such networks have been used extensively in the machine learning community, and are known to achieve high classification accuracy on a variety of tasks such as phoneme recognition [1], text document classification [2], and visual object recognition [3]. Due to their complexity, such networks are often treated as black boxes, giving accurate solutions to difficult problems without explaining how their solutions were found.

However, for hierarchical networks to be used for applications in which each classification carries high risk (e.g., medical diagnosis or financial decisions), it may be necessary for these networks to explain the evidence for *each* classification. More generally, for any classification method, it is important to verify that the network's results are due not to anomalies or accidental correlations in particular data sets, but to features of the data that make sense for the general task at hand.

Given an instance $\mathbf{x} = (x_1, x_2, \ldots, x_n)$ classified as $\hat{y}(\mathbf{x}) \in \{1, -1\}$, we seek a method to explain the relative importance of each input x_i to the instance's classification. Poulin *et al.* [4] offer such an explanation method for the case of additive classifiers. For an additive classifier $\hat{y}(\mathbf{x}) = \operatorname{sgn}\left[\sum_{i=1}^{n} f_i(x_i) + b\right]$, Poulin *et al.* define the contribution of feature x_i to be $f_i(x_i)$. However, the classifications of hierarchical networks cannot be explained with this method: even if the features calculated by the network are classified by an additive classifier, the high-level features themselves are generally nonadditive functions of the network's low-level inputs.

We address this limitation with a new method called *contribution propagation*. In short, we calculate contributions of high-level features as in Poulin *et al.* [4], and then work backward from the features to the network's input, determining the relative contribution of each node to its parent nodes. This process gives us a well-founded explanation for individual classifications: given a single classified datum, our method explains how important each part of the datum (*i.e.*, each entry in the input vector) was to that classification.

In what follows, we review the literature related to this research, and in Section II, we introduce notation and review the architecture of hierarchical networks. Section III gives the details of the contribution-propagation method. Section IV discusses the implementation details of our network, and we give experimental results in Section V. We conclude with directions for further research and open questions.

A. Related Work

Rule extraction refers to a family of methods that explain the general rules used by a network by analyzing the network's internal weights [5]. Similarly, one can visualize the network topology in order to understand the overall strategies that might be used [6]. The extracted rules are not specific to the model's decision for any particular datum. Rather, they describe the general behavior of the model over a large set

To appear in Computational Intelligence and Data Mining (CIDM) 2013

of data. Thus, rule extraction does not provide the type of information we seek: a fine-grained explanation of what caused a single datum's classification.

In an attempt to provide such explanations, recent research has treated the entire classification system as a single black box in order to provide classifier-independent explanations [7]. These methods avoid dealing with the internal calculations of the system at the expense of sampling from an exponentially large set of possible inputs. For classifiers with large input spaces, as is often the case with hierarchical networks, these methods introduce unwanted dependencies on sampling.

The **gradient** approach is one way to explain individual classifications [8]. Given an instance $\mathbf{x} = (x_1, x_2, \ldots)$ and its classification $\hat{y}(\mathbf{x}) = \text{sgn}[f(\mathbf{x})]$, this approach defines the *importance* of feature x_i to be $\frac{\partial f}{\partial x_i}$. Intuitively, x_i is important if f is sensitive to small changes in x_i. However, the gradient approach fails to tell us how much each feature *contributed* to the output. Consider the simple example of a linear classifier $f(\mathbf{x}) = \mathbf{w}^\top \mathbf{x}$, where $\mathbf{w} = (2, 2)$. An input $\mathbf{x} = (x_1, x_2) = (3, -1)$ gives output

$$f(\mathbf{x}) = 2 \cdot 3 + 2 \cdot (-1) = 6 - 2 = 4.$$

A different input $\mathbf{x}' = (x'_1, x'_2) = (-1, 3)$ gives output

$$f(\mathbf{x}') = 2 \cdot (-1) + 2 \cdot 3 = -2 + 6 = 4.$$

The gradient approach would tell us that our function f is equally sensitive to all inputs in both cases ($\frac{\partial f}{\partial x_i} = 2$). However, this analysis fails to tell us that in the first case, the positive output of f is due to the first feature x_1; and in the second case, the positive output of f is due to the second feature x'_2.

The **contribution** approach of Poulin *et al.* [4], on the other hand, perfectly captures this information. Using this approach, we would determine that for the first example, the contributions of x_1 and x_2 are 6 and -2, respectively. For the second example, the contributions of x'_1 and x'_2 are -2 and 6, respectively. Thus *contributions* (which we explore in more detail below) give us fine-grained explanations of the degree to which each input contributed the output, while the gradient approach loses much useful information about the classification.

II. Preliminaries

A. Notation

We use italicized letters to denote real numbers and bold letters to denote real vectors, such as $\mathbf{x} = (x_1, \ldots, x_n) \in \mathbb{R}^n$. The L2 norm of a vector, denoted $\parallel \mathbf{x} \parallel$, is calculated as $\sqrt{\sum_{i=1}^{n} x_i^2}$. A *node* is a real-valued function. Capital letters X refer to the identity of a node, the function calculated by that node, or the value returned by the node's function. The intended use will be clear from context.

For a given hierarchical network, X_i^ℓ refers to the i^{th} node in the ℓ^{th} layer. The vector $\mathbf{X}^\ell = (X_1^\ell, X_2^\ell, \ldots)$ denotes all nodes in layer ℓ. We write X^ℓ to mean *some* node in the ℓ^{th} layer of the network, or merely X to denote any particular node in the network. The *children* of node X^ℓ, denoted $\mathsf{ch}\left(X^\ell\right)$, are the set of nodes that are the inputs to node X^ℓ. In a slight abuse of notation, we sometimes treat $\mathsf{ch}\left(X^\ell\right)$ as a vector; the intended use will be clear by context. Similarly, the *parents* of node X^ℓ, denoted $\mathsf{pa}\left(X^\ell\right)$, are the set (or vector) of nodes that receive X^ℓ as input. In what follows, we will restrict the inputs of any node to come from only the preceding layer. Thus $\mathbf{X}^{\ell-1} \supseteq \mathsf{ch}\left(X^\ell\right)$ and $\mathbf{X}^{\ell+1} \supseteq \mathsf{pa}\left(X^\ell\right)$. A network has L layers ($\ell \in [1, \ldots, L]$). Thus the output of the network, often called the *feature vector*, is $\mathbf{X}^L$. For convenience, the input to the network is denoted simply $\mathbf{x}$. Lastly, let $\mathbf{x} \in \mathbb{R}^n$, and for any node X_i^1 in the network's first layer, $\mathsf{ch}\left(X_i^1\right) \subseteq \mathbf{x}$.

B. Hierarchical Networks

Hierarchical networks comprise a large family of machine-learning models such as HMAX [3], [9], convolutional neural networks [2], and others. In order to make our analysis concrete, in this work we focus on the well-known family of hierarchical networks described by HMAX. However, it should be noted that our *contribution propagation* method is applicable to the more general class of feed-forward hierarchical networks. We briefly review the architecture of HMAX-like networks here. Details of our implementation are given in the *Methods* section.

In a trained network, a node X_k^ℓ in an odd-numbered layer computes the radial basis function (RBF)

$$X_k^\ell = \exp(-\beta \parallel \mathsf{ch}\left(X_k^\ell\right) - \mathbf{P}_k^\ell \parallel^2),$$

where $\mathbf{P}_k^\ell$ and β are parameters of the model. We refer to $\mathbf{P}_k^\ell$ as the *kernel* of node X_k^ℓ. Following Serre *et al.* [3], we refer to these layers as S (*simple cell*) layers, with $S1$ being the first S layer, and so on. Nodes $X_h^{\ell+1}$ in even layers compute a maximum of their inputs:

$$X_h^{\ell+1} = \max_{X_k^\ell \in \mathsf{ch}\left(X_h^{\ell+1}\right)} X_k^\ell.$$

Again following Serre *et al.* [3], we refer to these layers as C (*complex cell*) layers.

Figure 1 shows a hierarchical network with two S layers and two C layers, whose input $\mathbf{x}$ consists of the gray-scale pixel values of an image. The output of the network (i.e., the output of the $C2$ layer) is the feature vector $\mathbf{X}^L$, which is given to a trainable classifier (in this case, a linear support vector machine (SVM)).

III. Methods

Given an instance $\mathbf{x} = (x_1, x_2, \ldots, x_n)$, its corresponding feature vector $\mathbf{X}^L$ (i.e., the final layer of the network), and the feature vector's binary classification $\hat{y}(\mathbf{X}^L) = \text{sgn}[f(\mathbf{X}^L)] \in \{1, -1\}$, we ask *what portion of the value $f(\mathbf{X}^L)$ came from input x_i?* In this work we assume that the "score" $f(\mathbf{X}^L)$ is an estimate of the classifier's confidence in its classification of $\mathbf{X}^L$ [10].

Rather than answer this question for the entire classifier and network as one large black box, our approach is to analyze each node (as well as the classifier) sequentially. Working from the classifier back to the inputs, we determine the contributions of the nodes in layer L to the classifier, then the contributions of the nodes in layer $L-1$ to the nodes in layer L, and so on until we have calculated the contribution of the inputs (*e.g.,* the contribution of each pixel in the image of Figure 1). This process, which we call *contribution propagation*, is ideal for interpreting the classifications made by complex networks

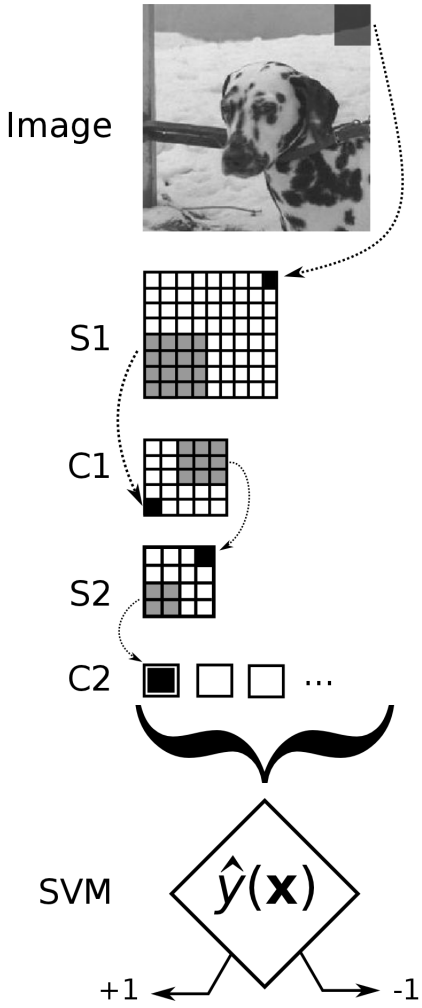

Fig. 1. A hierarchical network taking an image as input. Only a small subset of each layer is shown. Each small square is a node in the network. Computation flows from top (*Image*) to bottom (*SVM*). Dashed arrows illustrate the local connectivity of the network: a small subset of each layer (gray group of nodes) is fed as input to a single node (black) in the following layer. The vector consisting of each $C2$ output is the feature vector used for training and testing the SVM.

which are too large and complicated to interpret directly, yet each individual calculation (*i.e.*, calculating the value output by any single node in the network) is simple.

It is crucial to note that our *contribution-propagation algorithm* is used only in explaining a model's classification. The classification itself is performed by a trained hierarchical model of the user's choosing. Separation between the classification and explanation algorithms allows our explanation method to be used in a variety of settings with a variety of models. Pseudocode for our algorithm is given in Figure 2.

In the remainder of this section, we describe the *contribution-propagation* algorithm at a relatively high level. We then discuss how to apply the algorithm to the particular type of hierarchical network discussed in Section II-B. This involves deriving equations specific to the linear SVM, RBF, and maximum operator.

A. The Contribution-Propagation Algorithm

The purpose of the *contribution-propagation* algorithm is to provide an interpretable explanation of which components of the input were responsible (and to what degree) for a given datum's classification. When the classification is the result of many simple calculations, as is the case with hierarchical networks, we form an overall description of an input's importance (*i.e.,* this pixel was important to the classification) by analyzing the internal calculations of the network (*i.e.,* node X_i^ℓ was important to node $X_j^{\ell+1}$).

The central idea of contribution propagation is that *a node was important to the classification if it was important to its parents, and its parents were important to the classification.* Mathematically,

$$\mathcal{C}\left(X_i^\ell\right) \stackrel{\text{def}}{=} \sum_{X_j^{\ell+1} \in \mathsf{pa}\left(X_i^\ell\right)} \mathcal{C}\left(X_i^\ell | X_j^{\ell+1}\right) \mathcal{C}\left(X_j^{\ell+1}\right). \quad (1)$$

In Equation 1, $\mathcal{C}\left(X_i^\ell\right)$ is the contribution of node X_i^ℓ to the classification; similarly, $\mathcal{C}\left(X_j^{\ell+1}\right)$ is the contribution of node $X_j^{\ell+1}$ to the classification. Finally, $\mathcal{C}\left(X_i^\ell | X_j^{\ell+1}\right)$ is the *partial* contribution of node X_i^ℓ to its parent node $X_j^{\ell+1}$. Informally, this value represents how important X_i^ℓ was to $X_j^{\ell+1}$. We will give explicit definitions for these terms in later sections; for the moment, we complete our general discussion of the algorithm.

The *contribution-propagation* algorithm starts by calculating the contribution $\mathcal{C}\left(X_i^L\right)$ of each top-level feature X_i^L to the classifier. The algorithm then iteratively descends through the layers of the network, determining the contribution of each node to its parent nodes. This process is described in Figure 2.

```
// Given instance x = (x_1, x_2, ..., x_n),
for ℓ = L → 1 do
    for all i do
        Calculate C(X_i^ℓ)
        // calculate contribution of each network node
for all j do
    Calculate C(x_j)
    // calculate contribution of each input
return (C(x_1), C(x_2), ..., C(x_n))
```

Fig. 2. The contribution-propagation algorithm. $\mathcal{C}\left(X_i^\ell\right)$ is the total contribution of node X_i^ℓ. Recall that $\mathbf{x} = (x_1, \ldots, x_n)$ is the input to the network, thus $\mathcal{C}\left(x_j\right)$ is the total contribution of input x_j.

In order to complete the description of contribution propagation, we need only derive the equations for $\mathcal{C}\left(X_k^L\right)$ (the contribution of the network's output nodes) and $\mathcal{C}\left(X_i^\ell | X_j^{\ell+1}\right)$ (the partial contribution of internal nodes to their parents).

We have previously stated that $\mathcal{C}\left(X_i^\ell | X_j^{\ell+1}\right)$ represents how important X_i^ℓ was to $X_j^{\ell+1}$. To make this idea more concrete and well-founded, we impose the following constraints:

$$\sum_{X_i^\ell \in \mathsf{ch}\left(X_j^{\ell+1}\right)} \mathcal{C}\left(X_i^\ell | X_j^{\ell+1}\right) = 1 \quad (2)$$

$$\text{and} \qquad 0 \leq \mathcal{C}\left(X_i^\ell | X_j^{\ell+1}\right) \leq 1. \quad (3)$$

In other words, we require that $\mathcal{C}\left(X_i^\ell | X_j^{\ell+1}\right)$ be a distribution over $\mathsf{ch}\left(X_j^{\ell+1}\right)$. The meaning of this distribution is the *fraction of the value $X_j^{\ell+1}$ that is due to X_i^ℓ*.

With this in mind, we can see that each individual summand from Equation 1, $\mathcal{C}\left(X_i^\ell | X_j^{\ell+1}\right)\mathcal{C}\left(X_j^{\ell+1}\right)$, is the portion of the contribution of $X_j^{\ell+1}$ that is due to X_i^ℓ. Thus Equation 1 in its entirety tells us that a node's contribution $\mathcal{C}\left(X_i^\ell\right)$ is equal to all of the contributions in the parent's layer for which X_i^ℓ is responsible.

We now derive $\mathcal{C}\left(X_i^\ell | X_j^{\ell+1}\right)$ for RBF and maximum functions (the two types of nodes in our network). We will conclude with the definition of $\mathcal{C}\left(X_k^L\right)$, the contribution of the network outputs. This will complete our derivation of the contribution-propagation algorithm.

B. Contribution to Radial Basis Function

Consider a node $X_j^{\ell+1}$ in an S-layer. For convenience, let $\mathbf{X} = \mathsf{ch}\left(X_j^{\ell+1}\right)$ and $\mathbf{P} = \mathbf{P}_j^{\ell+1}$. Then the function computed by $X_j^{\ell+1}$ is the RBF

$$X_j^{\ell+1} = \exp(-\beta \parallel \mathbf{X} - \mathbf{P} \parallel^2). \tag{4}$$

Our goal is to define a function $\mathcal{C}_{\text{RBF}}\left(X_i^\ell | X_j^{\ell+1}\right)$ which is a distribution over $\mathsf{ch}\left(X_j^{\ell+1}\right)$ and which accurately describes the degree to which X_i^ℓ was responsible for the value of $X_j^{\ell+1}$.

Equation 4 is a measure of distance between the vectors $\mathbf{X}$ and $\mathbf{P}$. A closer distance yields a larger RBF value, and a further distance yields a smaller value. Thus we want $\mathcal{C}_{\text{RBF}}\left(X_i^\ell | X_j^{\ell+1}\right)$ to be higher if X_i^ℓ is closer to P_i, meaning when $(X_i^\ell - P_i)^2$ is smaller. Moreover, the distance calculated by the RBF is tuned by the function $s(x) = \exp(-\beta x)$. Thus, we define $\mathcal{C}_{\text{RBF}}\left(X_i^\ell | X_j^{\ell+1}\right)$ as:

$$\mathcal{C}_{\text{RBF}}\left(X_i^\ell | X_j^{\ell+1}\right) \stackrel{\text{def}}{=} \frac{\exp(-\beta(X_i^\ell - P_i)^2)}{Z}, \tag{5}$$

where Z is a normalization term.

Recalling the constraints in Equations 2 and 3, we note that the constraint of Equation 3 implies that $Z > \exp(-\beta(X_i^\ell - P_i)^2)$. To make $\mathcal{C}_{\text{RBF}}\left(X_i^\ell | X_j^{\ell+1}\right)$ satisfy the constraint in Equation 2, it must be normalized so that $\sum_i \mathcal{C}_{\text{RBF}}\left(X_i^\ell | X_j^{\ell+1}\right) = 1$. Thus we simply set the denominator Z in Equation 5 to equal the sum over all children of the RBF, and we arrive at the full definition:

$$\mathcal{C}_{\text{RBF}}\left(X_i^\ell | X_j^{\ell+1}\right) \stackrel{\text{def}}{=} \frac{\exp(-\beta(X_i^\ell - P_i)^2)}{\sum_{X_k^\ell \in \mathsf{ch}\left(X_j^{\ell+1}\right)} \exp(-\beta(X_k^\ell - P_k)^2)}. \tag{6}$$

Equation 6 agrees with our intuition: those children X_i that are close to their target P_i have a higher contribution.

C. Contribution to Maximum Function

A node $X_j^{\ell+1}$ in a C-layer calculates the maximum of its children,

$$X_j^{\ell+1} = \max_{X_i^\ell \in \mathsf{ch}\left(X_j^{\ell+1}\right)} X_i^\ell. \tag{7}$$

As before, our goal is to define the function $\mathcal{C}_{\text{MAX}}\left(X_i^\ell | X_j^{\ell+1}\right)$ which is a distribution over X_i^ℓ and which accurately describes the degree to which X_i^ℓ was responsible for the value of $X_j^{\ell+1}$.

To this end, we note that an input to a *max* function was important to the function if that input was itself the maximum value. Thus we view the *max* function as a type of switch, and we define

$$\mathcal{C}_{\text{MAX}}\left(X_i^\ell | X_j^{\ell+1}\right) \stackrel{\text{def}}{=} \begin{cases} 1/r & \text{if } X_i^\ell \in \mathcal{M} \\ 0 & \text{o.w.} \end{cases} \tag{8}$$

where $\mathcal{M} = \{X_i^\ell \in \mathsf{ch}\left(X_j^{\ell+1}\right) : X_i^\ell = X_j^{\ell+1}\}$, and $\mathcal{M}$ contains r elements. That is, we divide the contribution equally among those inputs that shared the maximum value; those children who were *not* the maximum did not contribute.

D. Contribution to an Additive Classifier

Given a feature vector $\mathbf{X}^L$ classified with an additive classifier as $\hat{y}(\mathbf{X}^L) = \text{sgn}\left[\sum_i f_i(X_i^L) + b\right]$, we define the contribution of feature X_i^L to be $f_i(X_i^L)$ as in Poulin *et al.* [4]. We denote the feature's contribution by writing

$$\mathcal{C}\left(X_i^L\right) \stackrel{\text{def}}{=} f_i(X_i^L). \tag{9}$$

Here we assume the additive classifier is a linear SVM [11]. Given the set $\mathcal{V}$ of support vectors, a linear SVM calculates $\hat{y}(\mathbf{X}^L) = \text{sgn}\left[\sum_{\mathbf{V}\in\mathcal{V}} \alpha_{\mathbf{V}} \langle \mathbf{V}, \mathbf{X}^L \rangle + b\right] = \text{sgn}\left[\sum_i X_i^L \left(\sum_{\mathbf{V}\in\mathcal{V}} \alpha_{\mathbf{V}} V_i\right) + b\right]$, where $\langle \cdot, \cdot \rangle$ denotes the dot product. In this case, Equation 9 becomes

$$\mathcal{C}_{\text{SVM}}\left(X_i^L\right) \stackrel{\text{def}}{=} X_i^L \left(\sum_{\mathbf{V}\in\mathcal{V}} \alpha_{\mathbf{V}} V_i\right). \tag{10}$$

That is, Equation 10 gives the contribution of node X_i^L to the SVM's classification of feature vector $\mathbf{X}^L$. A positive value gives the amount to which node X_i^L contributes to a positive classification (or away from a negative classification); similarly a negative value gives the degree of X_i^L's contribution towards a negative (or away from a positive) classification.

IV. Hierarchical Network Implementation

We implement a four-layer network (Figure 1), based on the network of Serre *et al.* [3]. The input image is preprocessed to form a 256×256 gray-scale image with local contrast enhancement. An S1 kernel is an 11×11-pixel Gabor filter. Using Equation 4, we apply a battery of Gabors at 8 orientations, 2 phases, and 4 scales, with $\beta = 1.0$ for all S1 nodes. For each Gabor configuration, we subsample by centering an S1 node at every other pixel, resulting in a set of 64 S1 output maps, each of size 128×128. A C1 node pools over the two phases and a 5×5 spatial neighborhood of S1 outputs. We again subsample in the same way, resulting in 32 C1 output maps, each of size 64×64. For an S2 node, the input is a 7×7 neighborhood of C1 nodes at all orientations, but at a single scale. The input vector and the kernel of each S2 node are each scaled to unit length ($\parallel \mathbf{X} \parallel = \parallel \mathbf{P} \parallel = 1$). We set $\beta = 5.0$ for every S2 node. For each kernel, there is a corresponding S2 node centered at every other C1 node, resulting in multiple 32×32 S2 output maps, one for each kernel and scale. Finally, a C2 node applies a max operation to all locations and all scales of a single kernel's

S2 map. Thus the output of the C2 layer is a vector with one component per S2 kernel. This *feature vector* is passed to the linear SVM. We use the SVM^{light} package [11] with an unbiased SVM ($b = 0$). This allows a simpler derivation of our method without impacting the accuracy of the network.

V. Results

We apply our contribution-propagation method in order to explain the classifications of hierarchical networks that are trained on different tasks. The first experiment is intentionally simple and controlled, and demonstrates that our approach accurately explains the network's classifications. The second experiment, which uses a well-known, real-world data set, shows how contribution propagation gives new insight into the network's performance.

A. Simple Shapes

In our first experiment, we use a simple artificial visual classification task to verify that our method's explanations are correct and understandable. Each training image contains a simple shape, either an 'L' shape (Figure 3B, positive class) or an inverted 'L' shape (Figure 3C, negative class). Noise is added by rotating the shape uniformly randomly within ± 5 degrees and translating the shape to a random location, and $1/f$ noise is added to the background. The noise ensures that the learned classifier is nontrivial.

Figure 3A shows the two learned S2 kernels around the vertex of the 'L' and inverted 'L' shapes. We input 20 training images (10 positive and 10 negative) to the network, and use the resulting feature vectors to train the SVM.

Our test images contain 9 possible shapes (Fig. 3 D), including both an 'L' and inverted 'L', each placed at a random position in a 3×3 grid and rotated randomly within ± 5 degrees. Again, $1/f$ noise is added to the background. Note that because both the positive and negative objects are present in the test image, we do not expect one classification over the other. The test images were designed to illustrate the authenticity of the contribution-propagation algorithm rather than to test the classification accuracy of the hierarchical network (accuracy will be addressed in the next section). All test images in this toy example were very near the decision boundary, which is reasonable given that both the positive and negative classes are present in each test image.

Using contribution propagation, we explain a test image's classification using false color as follows. First, we trace down through the layers of the network (Fig. 3E-H) using the algorithm presented in Figure 2. Thus Figure 2E shows the contribution of each node in the $S2$ layer, Figure 2F shows the contribution of each node in the $C1$ layer, and so on. This results in the calculation of the contribution $\mathcal{C}(x_i)$ of each pixel x_i (Fig. 3H). Pixels drawn in red contributed to a positive classification ('L'); pixels drawn in blue contributed to a negative classification (inverted 'L'); pixels that did not contribute to the classification are drawn in green.

We see in this visualization (Figure 3H) that our contribution-propagation algorithm correctly colored the pixels surrounding the area matching the learned kernels (red around the 'L' shape, and blue around the inverted 'L' shape). Our algorithm thus explains the classification of "undecided": there was a nearly equal "pull" between the pixels surrounding the 'L' (toward positive classification) and those surrounding the inverted 'L' (toward negative classification). This pixel-level explanation of how the image is interpreted by the network and classifier was provided automatically by our contribution-propagation algorithm, and gives evidence for the correctness of our algorithm.

B. Real-World Images

Next, we use the Caltech101 data set [12] to train the network and a linear, unbiased SVM in a binary classification task using categories of "chair" (*positive* class, corresponding to red in the visualizations) and "dalmatian" (*negative* class, corresponding to blue). The categories contain 60 images each. Using 10 splits for cross validation, we randomly choose 30 training images and 30 test images from each category. Following Serre *et al.* [3], the network "imprints" 1000 S2 kernels randomly from the S2 inputs of the training set, and the SVM is trained on the resulting network's output for each training image. Test images are classified with an average accuracy of 94%, with a 3% standard deviation (a biased SVM achieved 93% accuracy with 1.2% standard deviation).

In Figure 4, we see that some images (A, C) are correctly classified primarily due to the pixels of the object itself (B, D). However, our method also reveals some surprising behavior of the network and classifier (F, H): it appears that some images were correctly classified due to features extracted primarily from the image's background. In Figure 4F, this may be less surprising as the background is quite similar to the dalmatian. However, in Figure 4H, it is unclear why the background (dark red) was taken as evidence for the presence of a chair (or, possibly, absence of a dalmatian). Such an unexpected explanation offered by the contribution-propagation method can be useful to the user who is trying to create a system that will generalize well; the user can see that, at least in some cases, the network is basing its classification on features that are not relevant to the general task, due to either deficiencies in the network or spurious correlations in the data set.

A natural question is how often a correct classification is "surprising" (that is to say, a correctly classified image where the background appears to contribute more than the object). Formulating a metric to define such a surprising classification is beyond the scope of the present work. However, a subjective visual inspection of the classified images reveals 5 of the 60 classifications of test images to be of this nature.

As a final test of the classification explanations provided by the contribution-propagation algorithm, we edit an image to include both a dalmatian and a chair (Figure 4I). This image was classified by the network as negative (dalmatian), and the contribution propagation algorithm explains this classification. Figure 4 (J) shows that, although there were features associated with the chair class on the right side of the image (yellow, light red), the features extracted from the pixels belonging to the dalmatian were weighted more heavily (deep blue).

Some readers may feel that the small number of training images used may cast doubt on the validity of the trained classifier. However, note that researchers often benchmark their computer-vision system by measuring its performance on the

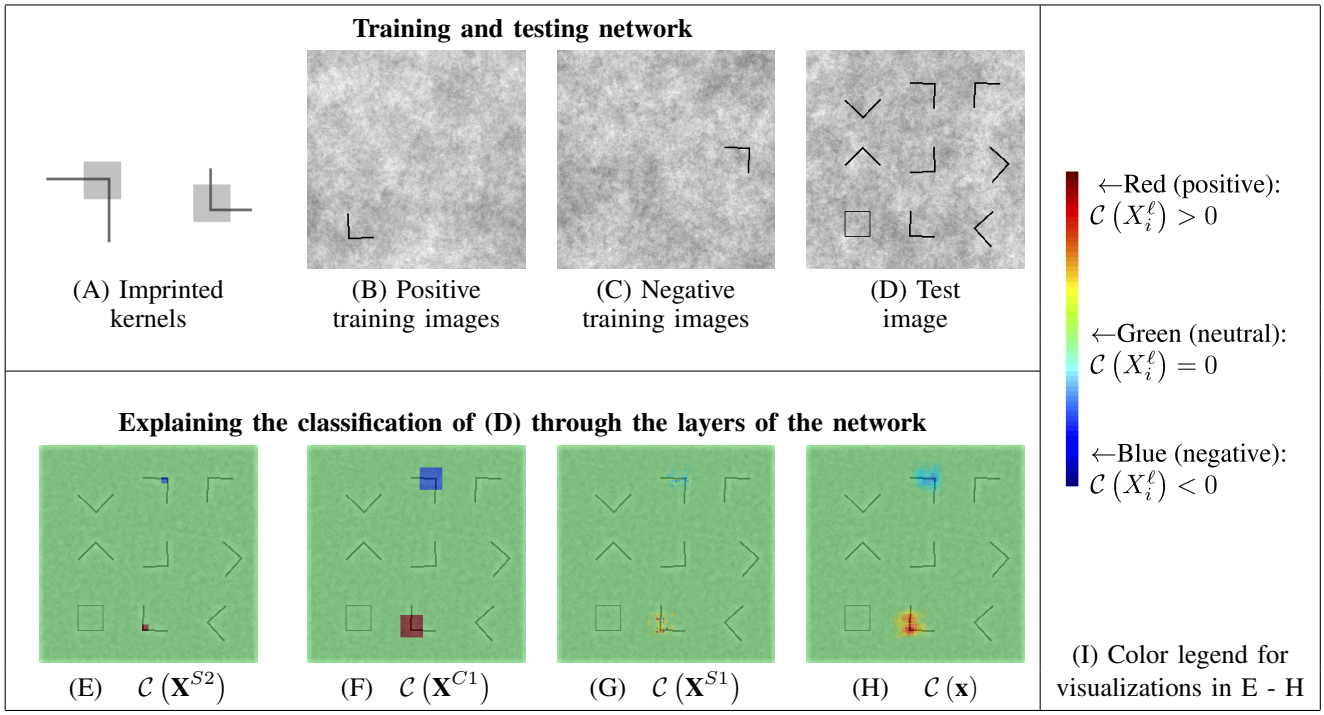

Fig. 3. (Figure best viewed in color.) Visualizing the binary classification of images containing simple shapes. The contribution of the nodes in each layer verifies the correctness of our algorithm. Two S2 kernels are used (shaded squares, A). A linear SVM is trained on images containing either an 'L' shape (B, positive class) or an inverted 'L' (C, negative class). We present a test image (D), and use contribution propagation to visualize the contribution of all layers (E-H). Note that the image is drawn in the background of (E-H) in order to better explain the contribution of each region. Colors correspond to the pixel's contribution, as shown in the legend (I). These visualizations give evidence for the correctness of our contribution propagation algorithm.

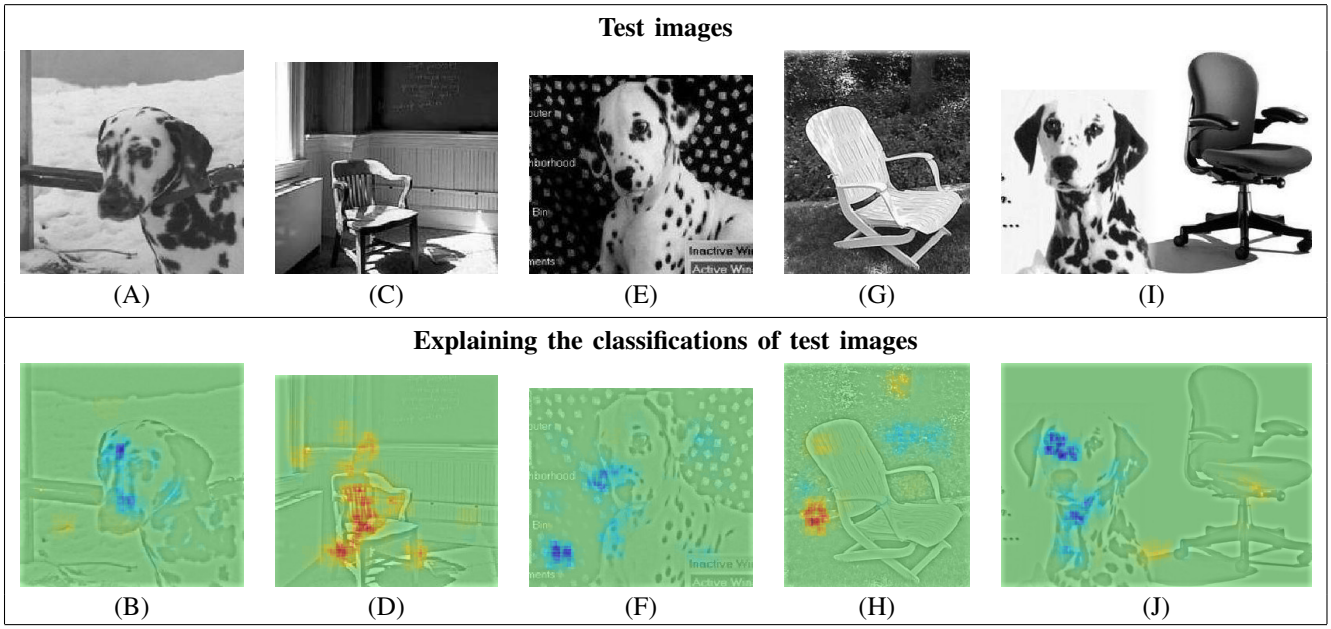

Fig. 4. (Figure best viewed in color.) Visualizations from the classification of chairs (positive) vs. dalmatians (negative), from the Caltech101 database. Colors correspond to the legend in Figure 3 (I): positive contribution (toward *chair*) is denoted with red, and negative contribution (toward *dalmatian*) with blue. Some images (A, C) are correctly classified because of the contribution of pixels that belong to the object being classified (B, blue on dalmatian; D, red on chair). Other images that contain confusing patterns in the background (E, G) are still correctly classified, but partially due to the contribution of background pixels (F, blue on background; H, red on background). An image manipulated to contain both objects (I) is classified as *dalmatian*, and this classification is intuitively explained by the contribution-propagation algorithm (J).

Caltech101 dataset using 30 images per class as training data [13]. Thus our experiment was designed to mimic a benchmarking process familiar to many computer-vision researchers. In this light, the surprising results presented in Figure 4 hint at an important question to the computer-vision community: does high performance on this dataset indicate a system's capacity for object recognition, or merely for learning spurious statistical (background) cues? The prevalence of this dataset makes this question all the more pressing.

VI. Conclusions

We presented a novel method for explaining the classifications of hierarchical networks with additive classifiers, having reviewed why traditional approaches such as a sensitivity analysis fail to give satisfactory explanations. Our method extends the contribution-based explanations of Poulin *et al.* [4], and determines the contribution of each input based on the internal calculations of the network.

We empirically validate our method's explanations with a simple object-recognition task using artificial data. We apply our method to a binary classification task using a well-known set of natural images, revealing surprising artifacts in the way that some images are classified. In particular, we see that some images are correctly classified because of the contribution of pixels belonging to the image's background (Figure 4 F, H). This behavior is surprising when the task is completed with high accuracy, but it is also very useful to the user of the machine-learning algorithm. Such information provided by our method can help the user to tune the algorithm for better generalizability, as well as to create data sets without artifacts in the background.

In order to visualize our method's explanations, we classify with an unbiased SVM. Although the lack of bias may, in some cases, lead to lower classification accuracy, it did not for any task we implemented. We stress that the lack of bias was chosen merely for visualization, and that the contribution-propagation method can be implemented with biased functions as well.

In future work, the contribution-propagation method may also be applied to different types of hierarchical networks, for example those using convolutions or different subsampling methods.

Acknowledgments

This material is based upon work supported by the National Science Foundation under Grant Nos. 1018967 and 0749348, as well as the Laboratory Directed Research and Development program at Los Alamos National Laboratory (Project 20090006DR). Any opinions, findings, conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of the funding agencies.

References

[1] R. Rifkin, J. Bouvrie, K. Schutte, S. Chikkerur, M. Kouh, T. Ezzat, and T. Poggio, "Phonetic classification using hierarchical, feed-forward, spectro-temporal patch-based architectures," Massachusetts Institute of Technology, Tech. Rep. MIT-CSAIL-TR-2007-019, 2007.

[2] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner, "Gradient-based learning applied to document recognition," *Proceedings of the IEEE*, vol. 86, no. 11, 1998.

[3] T. Serre, A. Oliva, and T. Poggio, "A feedforward architecture accounts for rapid categorization," *Proceedings of the National Academy of Sciences*, vol. 104, no. 15, 2007.

[4] B. Poulin, R. Eisner, D. Szafron, P. Lu, R. Greiner, D. Wishart, A. Fyshe, B. Pearcy, C. MAcDonell, and J. Anvik, "Visual explanation of evidence in additive classifiers," in *Proceedings of 18th Conference on Innovative Applications of Artificial Intelligence*. IAAI, July 2006.

[5] L. Fu, "Rule generation from neural networks," *IEEE Transactions on Systems, Man and Cybernetics*, vol. 24, no. 8, 1994.

[6] T. Masquelier and S. Thorpe, "Unsupervised learning of visual features through spike timing dependent plasticity," *PLoS Comp. Bio.*, vol. 3, no. 2, 2007.

[7] E. Strumbelj and I. Kononenko, "An efficient explanation of individual classifications using game theory," *Journal of Machine Learning Research*, vol. 11, no. 1, 2010.

[8] D. Baehrens, T. Schroeter, S. Harmeling, M. Kawanabe, K. Hansen, and K. Müller, "How to explain individual classification decisions," *Journal of Machine Learning Research*, vol. 11, pp. 1803–1831, 2010.

[9] M. Riesenhuber and T. Poggio, "Hierarchical models of object recognition in cortex," *Nature Neuroscience*, vol. 2, no. 11, 1999.

[10] M. Dredze, K. Crammer, and F. Pereira, "Confidence-weighted linear classification," in *International Conference on Machine Learning*, 2008.

[11] T. Joachims, "Making large-scale SVM learning practical." in *Advances in Kernel Methods - Support Vector Learning*, B. Schölkopf, C. Burges, and A. Smola, Eds. MIT Press, 1999.

[12] L. Fei-Fei, R. Fergus, and P. Perona, "Learning generative visual models from few training examples: an incremental Bayesian approach tested on 101 object categories," in *IEEE. CVPR 2004, Workshop on Generative-Model Based Vision*, 2004.

[13] A. Bosch, A. Zisserman, and X. Munoz, "Representing shape with a spatial pyramid kernel," in *Proceedings of the 6th ACM international conference on Image and video retrieval*, 2007.