SHAPLEY EXPLANATION NETWORKS

Anonymous authors
Paper under double-blind review

ABSTRACT

Shapley values have become one of the most popular feature attribution explanation methods. However, most prior work has focused on post-hoc Shapley explanations, which can be computationally demanding (exponential time complexity) and preclude model regularization based on Shapley explanations during training. Thus, we propose to incorporate Shapley values themselves as latent representations in deep models—thereby making Shapley explanations first-class citizens in the modeling paradigm. This intrinsic explanation approach enables layer-wise explanations, explanation regularization of the model during training, and fast explanation computation at test time. We define the *Shapley transform* that transforms the input into a *Shapley representation* given one or more scalar functions. We operationalize the Shapley transform as a neural network module and construct both shallow and deep networks, called SHAPNETs, by composing Shapley modules. We prove that our Shallow SHAPNETs compute the exact Shapley values and our Deep SHAPNETs maintain the missingness and accuracy properties of Shapley values. We demonstrate on synthetic and real world datasets that our SHAPNETs enable layer-wise Shapley explanations, novel Shapley regularizations during training, and fast computation while maintaining reasonable classification accuracy.

1 INTRODUCTION

Explaining the predictions of machine learning models has become increasingly important for many crucial applications such as healthcare, recidivism prediction, or loan assessment. Explanations based on feature importance are one key approach to explaining a model prediction. More specifically, additive feature importance explanations have become popular, and in Lundberg & Lee (2017), the authors argue for theoretically-grounded additive explanation method called SHAP based on Shapley values—a way to assign credit to members of a group developed in cooperative game theory (Shapley, 1953). Lundberg & Lee (2017) defined three intuitive theoretical properties called local accuracy, missingness, and consistency, and proved that only SHAP explanations satisfy all three properties.

Despite these elegant theoretically-grounded properties, exact Shapley value computation has exponential time complexity in the general case. To alleviate the computational issue, several methods have been proposed to approximate Shapley values via sampling and weighted regression (Kernel SHAP), a modified backpropagation step (Deep SHAP) (Lundberg & Lee, 2017), utilization of the expectation of summations (Ancona et al., 2019), or making assumptions on underlying data structures (Chen et al., 2019). To avoid approximation, the model class could be restricted to allow for simpler computation. Along this line, Lundberg et al. (2020) propose a method for computing exact Shapley values for tree-based models such as random forests or gradient boosted trees. However, even if this drawback is overcome, prior Shapley work has focused on *post-hoc* explanations, and thus, the explanation approach cannot aid in model design or training.

On the other hand, Generalized Additive Models (GAM) as explored in Lou et al. (2012; 2013); Caruana et al. (2015) (via tree boosting) and Wang et al. (2018) (via neural networks) can be seen as interpretable model class that exposes the exact Shapley explanation directly. In particular, the output of a GAM model is simply the summation of interaction-free functions: $f_{\text{GAM}}(\boldsymbol{x}) = \sum_s f_s(x_s)$, where $f_s(\cdot)$ are univariate functions that can be arbitrarily complex. Interestingly, the Shapley explanation values, often denoted $\phi_s(\boldsymbol{x}, f)$, for a GAM are exactly the values of the independent function, i.e., $\forall s, \phi_s(\boldsymbol{x}, f_{\text{GAM}}) = f_s(x_s)$. Hence, the prediction and the corresponding exact SHAP explanation can be computed simultaneously for GAM models. However, GAM models are inherently

limited in their representational power, particularly for perceptual data such as images in which deep networks are state-of-the-art. Thus, prior work is either post-hoc (which precludes leveraging the method during training) or limited in its representational power (e.g., GAMs).

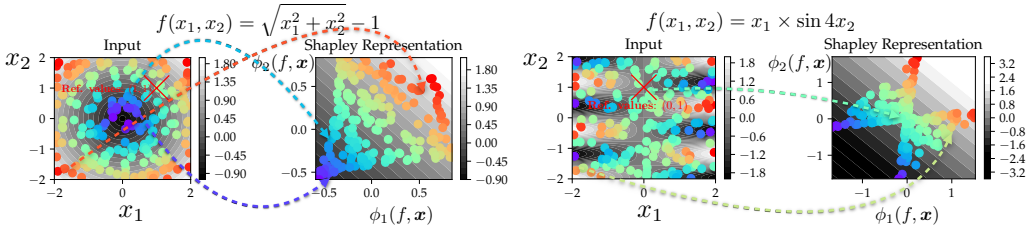

Figure 1: Shapley representation of two-dimensional functions. Such representation can span beyond one dimensional manifold and depends on both the inner function and the reference values. In both groups, the gray-scale background indicates the respective function value while the rainbow-scale color indicates correspondence between input (left) and Shapley representation (right) along with function values—red means highest and purple means lowest function values. The red cross in the input plots represents the reference values for both inputs. More details in subsection 2.1.

To overcome these drawbacks, we propose to incorporate Shapley values themselves as learned latent representations (as opposed to post-hoc) in deep models—thereby making Shapley explanations first-class citizens in the modeling paradigm. Intuitive illustrations of such representation are provided in Fig. 1 and detailed discussion in subsection 2.1. We summarize our core contributions as follows:

- We formally define the *Shapley transform* and prove a simple but useful linearity property for constructing networks. We also develop important extensions of the base case to enhance the representational power of the transform.
- We develop a novel network architecture SHAPNET, which include Shallow and Deep SHAPNET, that intrinsically provides layer-wise explanations (i.e., explanations at every layer of the network) in the same forward pass as the prediction.
- We prove that a Shallow SHAPNET explanations are the *exact* Shapley values—thus satisfying all three SHAP properties—and prove that Deep SHAPNET explanations maintain the missingness and local accuracy properties.
- We propose a novel *explanation regularization* on Shapley values of the model during training enabled by the fact that the explanation is a latent representation in our model.
- We demonstrate empirically that our SHAPNETs can provide new capabilities such as layerwise explanations and novel explanation regularizations while maintaining comparable performance to other deep models. We also show that our method compares favorably with Shapley-based post-hoc explanation methods.

Extended Related Works Section We present related works that motivated our work above and in the text where appropriate. Due to space limit, for extended literature review, we refer to Appendix C.

2 SHAPLEY EXPLANATION NETWORKS

Background We give a short introduction to SHAP explanations and their properties as originally introduced in Lundberg & Lee (2017). Given a model $C : \mathbb{R}^d \mapsto \mathbb{R}$ that is not inherently interpretable (e.g., neural nets), additive feature-attribution methods form a linear approximation of the function over simplified binary inputs, denoted $\boldsymbol{z} \in \{0, 1\}^d$, indicating the "presence" and "absence" of each feature, respectively: i.e., a local linear approximation $D(\boldsymbol{z}) = a_0 + \sum_{i=1}^{d} a_i z_i$. While there are different ways to model "absence" and "presence", in this work, we take a simplified viewpoint: "presence" means that we keep the original value whereas "absence" means we replace the original value with a reference value, which has been validated in Sundararajan & Najmi (2020) as Baseline Shapley. If we denote the reference vector for all features by $\boldsymbol{r}$, then we can define a simple mapping function between $\boldsymbol{z}$ and $\boldsymbol{x}$ as $\Psi_{\boldsymbol{x},\boldsymbol{r}}(\boldsymbol{z}) = \boldsymbol{z} \odot \boldsymbol{x} + (1 - \boldsymbol{z}) \odot \boldsymbol{r}$, where $\odot$ denotes element-wise product (eg, $\Psi_{\boldsymbol{x},\boldsymbol{r}}([0, 1, 0, 1, 1]) = [r_1, x_2, r_3, x_4, x_5]$). A simple generalization is to group certain

features together and consider including or removing all features in the group. Lundberg & Lee (2017) propose three properties that additive feature attribution methods should intuitively satisfy. The first property called *local accuracy* states that the approximate model D at $\boldsymbol{z} = \mathbf{1}$ should match the output of the model C at the corresponding $\boldsymbol{x}$, i.e., $C(\boldsymbol{x}) = D(\mathbf{1}) = \sum_{i=0}^{d} a_i$. The second property called *missingness* formalizes the idea that features that are "missing" from the input $\boldsymbol{x}$ (or correspondingly the zeros in $\boldsymbol{z}$) should have zero attributed effect on the output of the approximation, i.e., $z_i = 0 \Rightarrow a_i = 0$. Finally, the third property called *consistency* formalizes the idea that if one model always sees a larger effect when removing a feature, the attribution value should be larger (see (Lundberg & Lee, 2017) for full definition).

Definition 1 (SHAP Values (Lundberg & Lee, 2017))**.** *SHAP values are defined as:*

$$\phi_i\left(C, \boldsymbol{x}\right) \triangleq \sum_{\tilde{\boldsymbol{z}} \subseteq \boldsymbol{z} \setminus i} \frac{|\tilde{\boldsymbol{z}}|!\,(d - |\tilde{\boldsymbol{z}}| - 1)!}{d!} \left[C \circ \Psi_{\boldsymbol{x},\boldsymbol{r}}\left(\tilde{\boldsymbol{z}} \cup \{i\}\right) - C \circ \Psi_{\boldsymbol{x},\boldsymbol{r}}\left(\tilde{\boldsymbol{z}}\right)\right] , \tag{1}$$

where $|\tilde{\boldsymbol{z}}|$ is the number of non-zero entries, $\boldsymbol{z} \setminus i$ means $\boldsymbol{z}_j = 1$ for all $j \neq i$ and $\boldsymbol{z}_i = 0$, $\tilde{\boldsymbol{z}} \subseteq \boldsymbol{z} \setminus i$ represents all $\tilde{\boldsymbol{z}}$ whose non-zeros are subsets of the non-zero entries of $\boldsymbol{z} \setminus i$.

2.1 Shapley transform and Shapley representation

To develop our idea of using Shapley values as latent representations in all layers of the network, we first define the *Shapley transform*.

Definition 2 (Shapley transform)**.** *Given a c-tuple of scalar functions $f^{(1:c)} \triangleq [f^{(1)}, \cdots, f^{(c)}] \in \mathcal{F}^c$ where $\mathcal{F} = \{f \colon \mathbb{R}^d \mapsto \mathbb{R}\}$ is the set of scalar functions on d inputs, we define the* Shapley transform *$\Omega \colon (\mathbb{R}^d \times \mathcal{F}^c) \mapsto \mathbb{R}^{c \times d}$ to be:*

$$\Omega(\boldsymbol{x}, f^{(1:c)}) \triangleq \begin{bmatrix} \phi(f^{(1)}, \boldsymbol{x})^T \\ \vdots \\ \phi(f^{(c)}, \boldsymbol{x})^T \end{bmatrix} = \begin{bmatrix} \phi_1(f^{(1)}, \boldsymbol{x}) & \cdots & \phi_d(f^{(1)}, \boldsymbol{x}) \\ \vdots & \ddots & \vdots \\ \phi_1(f^{(c)}, \boldsymbol{x}) & \cdots & \phi_d(f^{(c)}, \boldsymbol{x}) \end{bmatrix} , \tag{2}$$

where $\phi(f^{(j)}, \boldsymbol{x}) \in \mathbb{R}^d$ for $1 \leq j \leq c$ is the Shapley value vector of $\boldsymbol{x}$ with respect to function $f^{(j)}$.

Definition 3 (Shapley representation)**.** *Given a tuple of functions $f^{(1:c)}$ as in Def. 2 and an input instance $\boldsymbol{x}$, we simply define a Shapley representation $Z \in \mathbb{R}^{c \times d}$ to be: $Z \triangleq \Omega(\boldsymbol{x}, f^{(1:c)})$.*

Remark 4 (Channel dimension of representation)**.** We will denote the dimension c corresponding to the rows of the representation the *channel* dimension as it is similar to the channel dimension in images. In this formulation, just as one pixel has three channels in RGB images, each of the d input features will have c channels in its representation (each from a scalar function $f^{(j)}$).

We now present a simple but useful property of Shapley transforms (proof in Appendix A).

Lemma 5 (Linear transformations of Shapley transforms are Shapley transforms)**.** *The linear transform, denoted by a matrix $A \in \mathbb{R}^{c' \times c}$, of a Shapley representation, $Z \triangleq \Omega(\boldsymbol{x}, f^{(1:c)})$, is itself a Shapley transform for modified functions $\tilde{f}^{(1:c')}$, i.e.,*

$$AZ \equiv A\Omega(\boldsymbol{x}, f^{(1:c)}) = \Omega(\boldsymbol{x}, \tilde{f}^{(1:c')}) \;\; \textit{where} \;\; \tilde{f}^{(k)}(\boldsymbol{x}) = \mathbf{a}_k^T f^{(1:c)}(\boldsymbol{x}) , \tag{3}$$

$\mathbf{a}_k \in \mathbb{R}^c$ is the k-th column of A, and $f^{(1:c)}(\boldsymbol{x}) \triangleq [f^{(1)}(\boldsymbol{x}), \cdots, f^{(c)}(\boldsymbol{x})] \in \mathbb{R}^c$.

Remark 6 (Row-wise sparsity curbs computation)**.** One of the main bottleneck of computing Shapley values is the exponential complexity in the number of input features that renders large-scale computing of Shapley values infeasible. To alleviate such issue, we would like to promote the sparsity in row-wise computation. Consider a function f depending on a subset of input features, (e.g. $f^{(j)}(x_1, x_2, x_3) = 5x_2 + x_3$ only depends on x_2 and x_3), we will refer to such a subset *active set* , $\mathcal{A}(f)$ ($\mathcal{A}(f^{(j)}) = \{2, 3\}$ for the example above) and its complement *dummy set*, $\mathcal{D}(f)$ ($\mathcal{D}(f^{(j)}) = \{1\}$ for the above example). In principle, we would like to keep the cardinality of the active set low: $\left|\mathcal{A}(f^{(j)})\right| \ll d$ for $1 \leq j \leq c$, s.t. we can limit the computational overhead, as features in the dummy set will be trivially assigned Shapley values of 0's.

To give a concrete instantiation of the Shapley transform, we propose a simple but novel wrapper around an arbitrarily complex function f called a *Shapley module* F that explicitly computes the Shapley values locally but only for the active set of inputs. We provide one of the examples in Fig. 2.

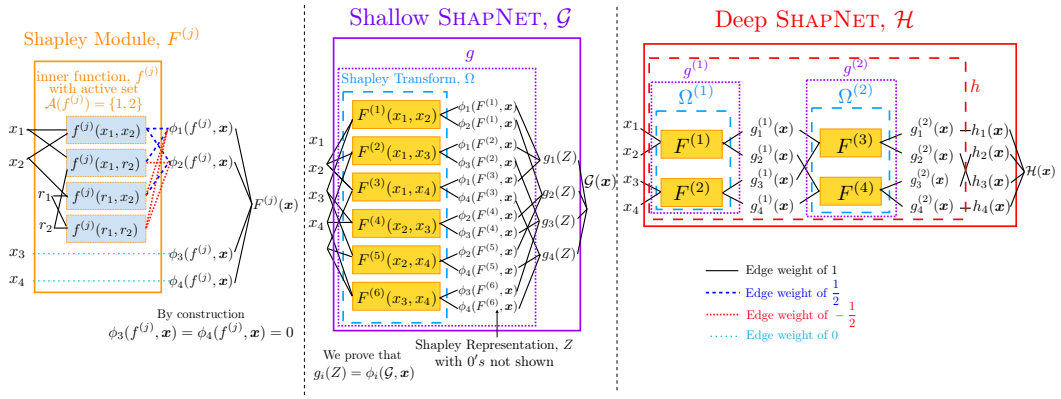

Figure 2: An example construction of SHAPNETs: we construct SHAPNET from simple Shapley modules (left) which explicitly compute the SHAP explanation values for the arbitrary function f. Shallow SHAPNET are based on computing many Shapley modules in parallel—in particular, we show computing all pairs of features. Deep SHAPNET are composed of Shallow SHAPNET blocks where the output explanation of the previous layer is used as input for the next layer. We show an example using disjoint subsets for each Shallow SHAPNET layer and then using a butterfly permutation as in the Fast Fourier Transform (FFT) algorithm to enable complex dependencies. The edge weights shown here are direct result of Eqn. 1 with $d = 2$.

Definition 7 (Shapley Module)**.** *Given an arbitrarily complex function $f^{(j)} \colon \mathbb{R}^d \mapsto \mathbb{R}$ parameterized by a neural network, and a reference vector $\boldsymbol{r} \in \mathbb{R}^d$, a* Shapley module *$F \colon \mathbb{R}^d \mapsto \mathbb{R}^d$ is defined as: $F(\boldsymbol{x} \mid f^{(j)}, \boldsymbol{r}) \triangleq \left[\phi_1(f^{(j)}, \boldsymbol{x}), \ldots, \phi_d(f^{(j)}, \boldsymbol{x})\right]^T$, where $\phi_i(f^{(j)}, \boldsymbol{x}) = 0$ if $i \in \mathcal{D}(f^{(j)})$.*

As one can infer, the output of a Shapley module constitutes one row of Eqn. 2, and with the cardinality of the active sets low, we can induce sparsity in rows of the representation Z in Eqn. 2.

Remark 8 (Vector-valued functions as tuples)**.** Def. 2 & Def. 3 can be trivially extended to vector-valued functions: $f \colon \mathbb{R}^d \to \mathbb{R}^c$, where we simply view each output as a scalar function, i.e., $f^{(j)}(\boldsymbol{x}) \equiv [f(\boldsymbol{x})]_j$. As an example, suppose that f is a multi-class classifier with c classes, then each of the $f^{(j)}$ would be the logits for the j-th class.

Remark 9 (Tuples of vector-valued functions: the meta-channels)**.** Suppose we are to compute the Shapley Transform of a tuple of vector-valued functions $(f^{(1)}(\boldsymbol{x}), \cdots, f^{(c)}(\boldsymbol{x}))$ where $f^{(j)} \colon \mathbb{R}^d \mapsto \mathbb{R}^n$, we will need to compute $\phi_i(f^{(j)}, \boldsymbol{x})$ for each output of $f^{(j)}$. In this case, we have again created another dimension as $\phi_i(f^{(j)}, \boldsymbol{x})$ now would become a vector instead of remaining as a scalar (by computing the Shapley value for each of the output dimensions of $f^{(j)}$). We coin this the *meta-channel* dimension, as it is the channel of channels: for each of the c channels of one feature, we have n scalars to represent it. Therefore, the co-domain of Shapley transform (Def. 2) becomes $\mathbb{R}^{(c \times n) \times d}$ where the Shapley representation Z resides. For a concrete example, superpixel representation of images has *groups* (features in our case) of pixels (channels in our case), and each pixel inside the groups has RGB channels (meta-channels here).

2.2 SHALLOW SHAPNET

To aid in our notation, we briefly define a simple dimension-wise summation operator.

Definition 10 (Dimension Sum Operator $\text{sum}^{[\alpha]}$)**.** *Given an input tensor $X \in \mathbb{R}^{\alpha_1 \times \alpha_2 \times \alpha_3 \cdots}$, $\text{sum}^{[\alpha_j]}$ will denote an operator that sums along the j-th dimension corresponding to the letter in brackets.*

For example, if $X \in \mathbb{R}^{c \times d}$, then $\text{sum}^{[c]}(X) = \sum_{i=1}^{c} \boldsymbol{x}_i$. Note that this is a simple linear transformation so we can use it to create various simplified Shapley transforms via Lemma 5. We present our Shallow SHAPNET and its accompanying Theorem 12 which is an extension of Lemma 5 with complete proof in Appendix A.

Definition 11 (Shallow SHAPNET)**.** *A Shallow* SHAPNET $\mathcal{G}$ *is defined as*

$$\mathcal{G}(\boldsymbol{x}) = \text{sum}^{[d]} \circ g(\boldsymbol{x}; f^{(1:c)}) = \text{sum}^{[d]} \circ (\text{sum}^{[c]} \circ \Omega(\boldsymbol{x}, f^{(1:c)})) = \sum_{i=1}^{d} \sum_{j=1}^{c} \phi_i(f^{(j)}, \boldsymbol{x}), \quad (4)$$

where $g(\boldsymbol{x}; f^{(1:c)})$ *will be called the Shallow* SHAPNET *explanation.*

Theorem 12. *Shallow* SHAPNETs *compute the exact Shapley values, i.e.,* $g_i(\boldsymbol{x}; f^{(1:c)}) = \phi_i(\mathcal{G}, \boldsymbol{x})$.

Note that per Remark 9, each individual $\phi_i(f^{(j)}, \boldsymbol{x})$ could be a vector. This is useful when the output of a model needs to be a vector, e.g. multi-class classification tasks. The vector-valued nature in this scenario allows us to have one importance value for each class. In this case, when performing prediction, we simply use element-wise summation. An example is shown in Fig. 2.

2.3 DEEP SHAPNET

While Shallow SHAPNET is quite powerful, it does not enable inter-layer Shapley representations within a deep model. To enable this, we construct deep SHAPNETs by cascading the output of Shallow SHAPNETs from Def. 11. Until this point the only input space we have discussed is $\mathbb{R}^d$. However, as presented in Remark 9 and subsection 2.2, the Shapley representation $Z \in \mathbb{R}^{n \times d}$ is a matrix, and that would mean the next Shallow SHAPNET layer inside the deep SHAPNET would have a matrix as input. To achieve this, we generalize Shapley transform $\mathcal{S}$ by modifying $\Psi_{\boldsymbol{x},\boldsymbol{r}}$:

Definition 13 (Generalized Shapley Transform)**.** *Shapley transform is generalized by simply changing the mapping* $\Psi_{\boldsymbol{x},\boldsymbol{r}}$ *in the computation of* $\phi_i(f^{(j)}, \boldsymbol{x})$ *as in Def. 1 and in Def. 2, to be* $\Psi_{\boldsymbol{x},\boldsymbol{r}} : \mathbb{R}^d \mapsto \mathbb{R}^{(n \times d)}$ *that treats each of the* d *vectors as a single input feature. In other words, with* $\boldsymbol{x}, \boldsymbol{r} \in \mathbb{R}^{n \times d}$, $\boldsymbol{z} \in \mathbb{R}^d$, $1 < i < n$, $1 < j < d$ *indexing rows and columns, respectively:*

$$\Psi_{\boldsymbol{x},\boldsymbol{r}}(\boldsymbol{z}) = [\boldsymbol{x}_{ij} \cdot z_j + (1 - z_j) \cdot \boldsymbol{r}_{ij}], \quad (5)$$

combined with, of course, the extension we discussed in Remark 9.

Definition 14 (Deep SHAPNET)**.** *A Deep* SHAPNET $\mathcal{H}$ *is defined as a cascade of Shallow* SHAPNETs *with generalized Shapley transform* $\mathcal{H}$*:*

$$\mathcal{H} = \text{sum}^{[d]} \circ h = \text{sum}^{[d]} \circ \text{sum}^{[c]} \circ \mathcal{S}^{(L)} \circ \text{sum}^{[c]} \circ \mathcal{S}^{(L-1)} \cdots \circ \text{sum}^{[c]} \circ \mathcal{S}^{(2)} \circ \text{sum}^{[c]} \circ \Omega, \quad (6)$$

where $h(\boldsymbol{x})$ *is known as the* Deep SHAPNET explanation *and all the reference values are set to 0 except for the first Shapley transform* Ω *(whose reference values will depend on the application).*

To ground our Deep SHAPNETs, we present the following theoretical properties about the explanations of Deep SHAPNETs (proof in Appendix B).

Theorem 15. *The Deep* SHAPNET *explanation* $h(\boldsymbol{x})$ *defined in Def. 14 provides an explanation that satisfies both local accuracy and missingness.*

Deep SHAPNET with Disjoint Pairs and Butterfly Connections It is immediately obvious that the computational graph determined by the choice of c-tuple functions $f^{(1:\,c)}$ dictates how the features interact with each other in a in Shallow SHAPNETs and hence a Deep SHAPNET. For our experiments, we focus on one particular construction of a Deep SHAPNET based on disjoint pairs (active sets of Shapley modules do not overlap: $\cap_{j=1}^{c} \mathcal{A}(f^{(j)}) = \emptyset$) in each layer and a butterfly permutation between each layer to allow interactions between many different pairs of features—similar to the Fast Fourier Transform (FFT) butterfly construction. This also means that the cardinality of the active set of all the Shapley modules are set to 2, making the overhead for computing the Shapley values roughly 4 $\times$ that of the underlying function. An example of this specific deep network construction can be seen on the right of Fig. 2. We emphasize that this permutation is one of the choices that enable fast feature interactions for constructing deep SHAPNETs from Shallow SHAPNETs. We do not claim it is necessarily the best but believe it is a reasonable choice if no other assumptions are made. One could also construct SHAPNETs based on prior belief (an example for images below).

Deep SHAPNET for Images Here we describe one of the many possibilities to work on image dataset with SHAPNETs. This is largely inspired by works including Desjardins et al. (2015); Dinh et al. (2017); Kingma & Dhariwal (2018). Still, the Deep SHAPNET here consists of different layers of Shallow SHAPNETs. We begin by describing the operations in each of the consecutive Shallow SHAPNETs, and then cascading the output from one stage to the next.

Canonical convolution operation is composed with three consecutive operations: sliding-window (unfolding the image representation tensor to acquire small patches matching the the filter), dot product between filters and the small patches from the images, and folding the resulting representation into the usual image representation tensor. We merely 1) replace the dot product operation with Shapley modules (similar to that in Desjardins et al. (2015)) and 2), since we have the same number of pixels, we can put the corresponding output vector representation $\mathbb{R}^{n'}$ back in the location of the original pixel. To create a hierarchical representation of images, we use à-trous convolution with increasing dilation as the Shapley transform goes from one stage to the next, similar to Chen et al. (2017); Chen et al. (2018). To reduce computation, we can choose a subset of the meta-channels of each pixels to take as inputs, similar to Dinh et al. (2017). For the final prediction, we do a global summation pooling, given that the final Shapley representation has the same number of meta-channels as the number of classes. The specific structure of the c-tuple functions of each Shallow SHAPNET and more detailed discussion on this entire model can be found in subsection G.3.

2.4 EXPLANATION REGULARIZATION DURING TRAINING

One of the important benefits to SHAPNETs is that we can regularize the explanations *during training*. The main idea is to regularize the last layer explanation so that the model learns to attribute features in ways aligned with human priors–e.g. sparse or smooth. This is quite different from smoothing the explanation *post-hoc* as in saliency map smoothing methods (Ross et al., 2017; Smilkov et al., 2017; Sundararajan et al., 2017; Yeh et al., 2019), but falls more into the lines of usage of interpretations of models (Liu & Avci, 2019; Erion et al., 2019; Rieger et al., 2020). For ℓ_1 regularization, the main idea is similar to sparse autoencoders in which we assume the latent representation is sparse.This is related to the sparsity regularization in LIME (Ribeiro et al., 2016) but is fundamentally different because it actually changes the learned model rather than just the explanation. For ℓ_∞ regularization, the idea is to smooth the Shapley values so that none of the input features become too important individually. This could be useful for example in a security setting where the model should not be too sensitive to any one sensor because the sensors could be adversarial attacked. Finally, with ad-hoc knowledge on the attribution appropriate for the task at hand, the user could specify different regularizations.

3 EXPERIMENTS & VISUALIZATIONS

We will (1) validate that our SHAPNET models can be quite expressive despite the intrinsic explanation design, (2) demonstrate that our intrinsic SHAPNET explanations perform comparably or better than posthoc explanations, and (3) highlight the novel capabilities of layer-wise explanations and explanation regularization enabled by the design of SHAPNET. More details in Appendix G.

Datasets First, we create a synthetic regression dataset by sampling the input $\boldsymbol{v} \in [-0.5, 0.5]^{16}$ from the uniform distribution where the true (synthetic) regression model is $f(\boldsymbol{v}) = \prod_i v_i + \sum_i v_i^2 + 0.05\epsilon$, where ϵ is sampled from a standard normal distribution. Second, we choose two real-world datasets from Dua & Graff (2017): Yeast ($d = 8$) and Breast Cancer Wisconsin (Diagnostic) ($d = 30$), which are both multi-class classification datasets. Third, we present some experiments on MNIST (LeCun & Cortes, 2010), FashionMNIST (Xiao et al., 2017), and Cifar-10 (Krizhevsky, 2009) datasets to validate SHAPNET on higher-dimensional datasets.

SHAPNET **model performance** We would first like to validate that our Deep and Shallow SHAPNET models can be comparable in performance to other models—i.e., that our Shapley module structure does not significantly limit the representational power of our model. Thus, we define DNN models that are roughly comparable in terms of computation or the number of parameters—since our networks require roughly four times as much computation as a vanilla DNN but share many parameters. For our comparison DNNs, we set up a general feedforward network with residual connections (He et al., 2016). The performance of the models are shown in Table 1 in which the loss

Table 1: Model performance (loss for synthetic or accuracy for others, averaged over 50 runs)

Datasets \ Models	Deep SHAPNET	DNN (eq. comp.)	DNN (eq. param.)	Shallow SHAPNET	GAM
Synthetic (loss)	3.37e-3	3.93e-3	6.62e-3	3.11e-3	3.36e-3
Yeast	0.585	0.576	0.575	0.577	0.597
Breast Cancer	0.959	0.966	0.971	0.958	0.969

Datasets \ Models	Deep SHAPNET for Images
MNIST	0.992
FashionMNIST	0.912
Cifar-10	0.820

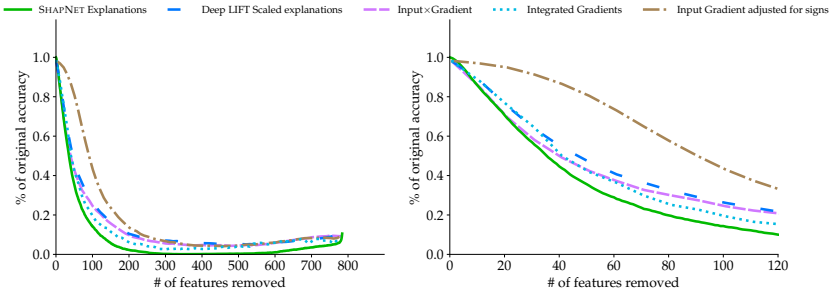

Figure 3: Our intrinsic Deep SHAPNET explanations perform better than post-hoc explanations in identifying the features that contribute most to the prediction as seen in this figure showing the accuracy after removing the top k features identified by each explanation method. Figure on the right is a zoomed-in version to show more detail for the first 120 features.

is shown for the synthetic dataset (lower is better) and classification accuracy for the other datasets (higher is better). While there is a very slight degradation in performance, our lower-dimensional models are comparable in performance even with the structural restriction. While there is some gap between state-of-the-art for image data, our image-based deep SHAPNET models can indeed do reasonably well even on these high-dimensional non-linear classification problems, but we emphasize that they also provide fundamentally new capabilities as explored later. Details are in Appendix G.

Intrinsic SHAPNET explanations compared to post-hoc explanations We now compare the intrinsic SHAPNET explanations with other post-hoc explanation methods. First, because our intrinsic SHAPNET explanations are simultaneously produced with the prediction, the explanation time is merely the cost of a single forward pass (we provide a wall-clock time experiment in Appendix D). For the lower-dimensional datasets, we validate our intrinsic explanations compared to other SHAP-based post-hoc explanations by computing the difference to the true SHAP values (which can be computed exactly or approximated well in low dimensions). Our results show that Shallow SHAPNET indeed gives the true SHAP values up to numerical precision as proved by Theorem 12, and Deep SHAPNET explanations are comparable to other post-hoc explanations (result table and more discussion in Appendix E). For image-based data, we will evaluate our high-dimensional explanations based on dropping the top-k features as identified by the explanation and measuring the corresponding accuracy of the classifier. Ideally, the top features would drastically reduce the classification accuracy—showing that our explanations do highlight the important features. We compare Deep SHAPNET explanations with DeepLIFT (Shrikumar et al., 2017) (scaled to get DeepSHAP (Lundberg & Lee, 2017)), Integrated Gradients (Sundararajan et al., 2017), Input×Gradients(Shrikumar et al., 2016) and vanilla gradients adjusted for signs. As shown in Fig. 3, Deep SHAPNET offers high-quality explanations as the model's accuracy dropped the fastest with the explanation values.

New capability: Layer-wise explanations With the layer-wise explanation structure, we can probe into the Shapley representation at each stage of a Deep SHAPNET as in Fig. 5. We also provide a set of pruning experiment shown in Fig. 7 of Appendix F, where we show how pruning the values in the Shapley representations of different layer changes model performance.

New capability: Explanation regularization during training We experiment with ℓ_1 and ℓ_∞ explanation regularizations during training to see if the regularizations significantly affect the underlying model behavior. For the lower-dimensional datasets, we compute the accuracy, the sparsity, and the coefficient of variation (CV, standard deviation divided by mean) for our Deep SHAPNET. We see in Table 2 that our ℓ_∞ regularization spreads the feature importance more evenly over features (i.e., low CV), ℓ_1 regularization increases the sparsity (i.e., higher sparsity), and both either improve

Table 2: Explanation regularization experiments with Deep SHAPNETs (averaged over 50 runs)

Metrics \ Models	Yeast			Breast Cancer Wisconsin		
	ℓ_∞ Reg.	ℓ_1 Reg.	No Reg.	ℓ_∞ Reg.	ℓ_1 Reg.	No Reg.
Coefficient of variation for abs. SHAP	**0.768**	1.23	1.05	**1.28**	NaN	2.04
Sparsity of SHAP values	0.003	**0.00425**	0.00275	0.429	**0.841**	0.209
Accuracy	**0.592**	**0.592**	0.587	0.957	**0.960**	**0.960**

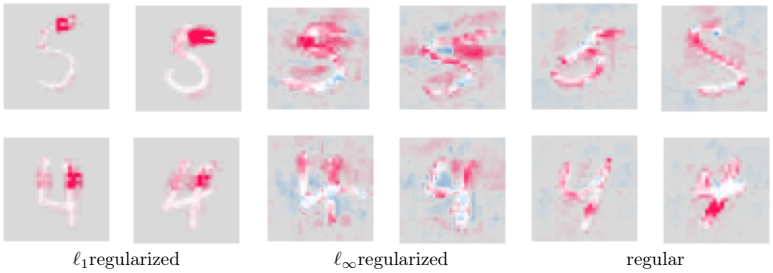

Figure 4: MNIST SHAPNET explanations for different regularizations qualitatively demonstrate the effects of regularization. We notice that ℓ_1 only puts importance on a few key features of the digits while ℓ_∞ spreads out the contribution over more of the image. Red and blue correspond to positive and negative contribution respectively. More visualization of the explanations, including the other classes and more in-depth discussion, can be found in subsection G.4.

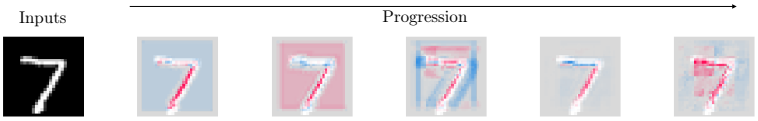

Figure 5: Deep SHAPNETs enable us to peek into the progression of feature importance during the forward pass with layer-wise explainability. The blocks appear in early stages presumably because the first two layers computes the Shapley representations in a relatively small neighborhood, and since all the neighboring pixels share the same values, they output the same blocks. We simply take the average of the contribution over all channels in the Shapley representation and scale to $[-1, 1]$. Expanded figures (Fig. 11, Fig. 12, Fig. 13, & Fig. 14) in subsection G.5 with discussions.

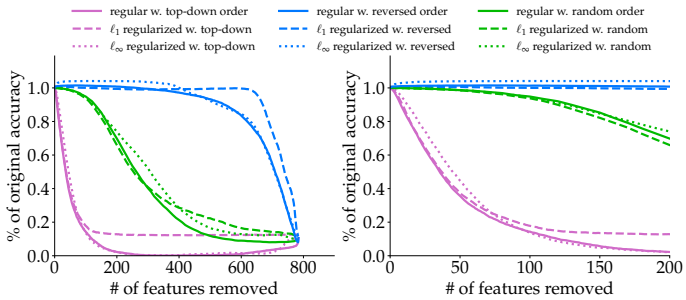

Figure 6: Both ℓ_1 and ℓ_∞ regularizations often introduce some robustness to the model against feature removal (the dotted and dashed lines are often above the solid lines). Three different feature orders are used: 1) most positive to most negative (top-down), 2) most negative to most positive (reversed), and 3) randomly chosen as a baseline. ℓ_1 regularization puts most importance on the first 100 features with relatively low importance to other features (seen in both top-down and reversed). ℓ_∞ regularization improves robustness when removing the top 50 or so features (as seen in the right figure which is zoomed in on the top 200 features).

or marginally degrade the accuracy. For MNIST data, we visualize the explanations from different regularization models in Fig. 4. For a more rigorous study of regularizers on image datasets, we perform the top-k feature removal experiment with different regularizers in Fig. 6, which shows that our regularizers produce some model robustness towards missing features. Moreover, Fig. 7 in Appendix F also shows that ℓ_1 regularized model achieves the best result under *value pruning* across different layers and ℓ_∞ performs the worst. Details are in Appendix F.

REFERENCES

David Alvarez Melis and Tommi Jaakkola. Towards robust interpretability with self-explaining neural networks. In S. Bengio, H. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi, and R. Garnett (eds.), *Advances in Neural Information Processing Systems 31*, pp. 7775–7784. Curran Associates, Inc., 2018. URL http://papers.nips.cc/paper/8003-towards-robust-interpretability-with-self-explaining-neural-networks.pdf.

Marco Ancona, Cengiz Oztireli, and Markus Gross. Explaining deep neural networks with a polynomial time algorithm for shapley value approximation. In Kamalika Chaudhuri and Ruslan Salakhutdinov (eds.), *Proceedings of the 36th International Conference on Machine Learning*, volume 97 of *Proceedings of Machine Learning Research*, pp. 272–281, Long Beach, California, USA, 09–15 Jun 2019. PMLR. URL http://proceedings.mlr.press/v97/ancona19a.html.

Marco Ancona, Cengiz Öztireli, and Markus Gross. Shapley Value as Principled Metric for Structured Network Pruning. *arXiv:2006.01795 [cs]*, June 2020. URL http://arxiv.org/abs/2006.01795. arXiv: 2006.01795.

Jimmy Lei Ba, Jamie Ryan Kiros, and Geoffrey E. Hinton. Layer Normalization. *arXiv:1607.06450 [cs, stat]*, July 2016. URL http://arxiv.org/abs/1607.06450. arXiv: 1607.06450.

Rich Caruana, Yin Lou, Johannes Gehrke, Paul Koch, Marc Sturm, and Noemie Elhadad. Intelligible models for healthcare: Predicting pneumonia risk and hospital 30-day readmission. In *Proceedings of the 21th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '15, pp. 1721–1730, New York, NY, USA, 2015. Association for Computing Machinery. ISBN 9781450336642. doi: 10.1145/2783258.2788613. URL https://doi.org/10.1145/2783258.2788613.

Jianbo Chen, Le Song, Martin J. Wainwright, and Michael I. Jordan. L-shapley and c-shapley: Efficient model interpretation for structured data. In *International Conference on Learning Representations*, 2019. URL https://openreview.net/forum?id=S1E3Ko09F7.

L. Chen, G. Papandreou, I. Kokkinos, K. Murphy, and A. L. Yuille. Deeplab: Semantic image segmentation with deep convolutional nets, atrous convolution, and fully connected crfs. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 40(4):834–848, 2018.

Liang-Chieh Chen, George Papandreou, Florian Schroff, and Hartwig Adam. Rethinking atrous convolution for semantic image segmentation. *CoRR*, abs/1706.05587, 2017. URL http://arxiv.org/abs/1706.05587.

Guillaume Desjardins, Karen Simonyan, Razvan Pascanu, and koray kavukcuoglu. Natural neural networks. In C. Cortes, N. D. Lawrence, D. D. Lee, M. Sugiyama, and R. Garnett (eds.), *Advances in neural information processing systems 28*, pp. 2071–2079. Curran Associates, Inc., 2015. URL http://papers.nips.cc/paper/5953-natural-neural-networks.pdf.

Laurent Dinh, Jascha Sohl-Dickstein, and Samy Bengio. Density estimation using Real NVP. *ICLR*, 2017. URL https://openreview.net/forum?id=HkpbnH9lx.

Dheeru Dua and Casey Graff. UCI machine learning repository, 2017. URL http://archive.ics.uci.edu/ml.

Gabriel G. Erion, Joseph D. Janizek, Pascal Sturmfels, Scott Lundberg, and Su-In Lee. Learning explainable models using attribution priors. *CoRR*, abs/1906.10670, 2019. URL http://arxiv.org/abs/1906.10670.

Riccardo Guidotti, Anna Monreale, Salvatore Ruggieri, Dino Pedreschi, Franco Turini, and Fosca Giannotti. Local rule-based explanations of black box decision systems. *CoRR*, abs/1805.10820, 2018a. URL http://arxiv.org/abs/1805.10820.

Riccardo Guidotti, Anna Monreale, Salvatore Ruggieri, Franco Turini, Fosca Giannotti, and Dino Pedreschi. A survey of methods for explaining black box models. *ACM Comput. Surv.*, 51(5), August 2018b. ISSN 0360-0300. doi: 10.1145/3236009. URL https://doi.org/10.1145/3236009.

K. He, X. Zhang, S. Ren, and J. Sun. Deep residual learning for image recognition. In *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 770–778, Las Vegas, NV, June 2016. doi: 10.1109/CVPR.2016.90.

Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization. 2014. URL http://arxiv.org/abs/1412.6980. cite arxiv:1412.6980Comment: Published as a conference paper at the 3rd International Conference for Learning Representations, San Diego, 2015.

Durk P Kingma and Prafulla Dhariwal. Glow: Generative flow with invertible 1x1 convolutions. In S. Bengio, H. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi, and R. Garnett (eds.), *Advances in Neural Information Processing Systems 31*, pp. 10215–10224. Curran Associates, Inc., 2018. URL http://papers.nips.cc/paper/8224-glow-generative-flow-with-invertible-1x1-convolutions.pdf.

A. Krizhevsky. Learning multiple layers of features from tiny images. 2009.

Yann LeCun and Corinna Cortes. MNIST handwritten digit database. 2010. URL http://yann.lecun.com/exdb/mnist/.

Frederick Liu and Besim Avci. Incorporating priors with feature attribution on text classification. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pp. 6274–6283, Florence, Italy, July 2019. Association for Computational Linguistics. doi: 10.18653/v1/P19-1631. URL https://www.aclweb.org/anthology/P19-1631.

Yin Lou, Rich Caruana, and Johannes Gehrke. Intelligible models for classification and regression. In *Proceedings of the 18th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '12, pp. 150–158, New York, NY, USA, 2012. Association for Computing Machinery. ISBN 9781450314626. doi: 10.1145/2339530.2339556. URL https://doi.org/10.1145/2339530.2339556.

Yin Lou, Rich Caruana, Johannes Gehrke, and Giles Hooker. Accurate intelligible models with pairwise interactions. In *Proceedings of the 19th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '13, pp. 623–631, New York, NY, USA, 2013. Association for Computing Machinery. ISBN 9781450321747. doi: 10.1145/2487575.2487579. URL https://doi.org/10.1145/2487575.2487579.

Scott M Lundberg and Su-In Lee. A Unified Approach to Interpreting Model Predictions. In I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett (eds.), *Advances in Neural Information Processing Systems 30*, pp. 4765–4774. Curran Associates, Inc., 2017. URL http://papers.nips.cc/paper/7062-a-unified-approach-to-interpreting-model-predictions.pdf.

Scott M. Lundberg, Gabriel Erion, Hugh Chen, Alex DeGrave, Jordan M. Prutkin, Bala Nair, Ronit Katz, Jonathan Himmelfarb, Nisha Bansal, and Su-In Lee. From local explanations to global understanding with explainable AI for trees. *Nature Machine Intelligence*, 2(1):56–67, January 2020. ISSN 2522-5839. doi: 10.1038/s42256-019-0138-9. URL https://doi.org/10.1038/s42256-019-0138-9.

W. James Murdoch, Peter J. Liu, and Bin Yu. Beyond word importance: Contextual decomposition to extract interactions from LSTMs. In *International Conference on Learning Representations*, 2018. URL https://openreview.net/forum?id=rkRwGg-0Z.

Vinod Nair and Geoffrey E. Hinton. Rectified linear units improve restricted boltzmann machines. In *Proceedings of the 27th International Conference on International Conference on Machine Learning*, ICML'10, pp. 807–814, Madison, WI, USA, 2010. Omnipress. ISBN 9781605589077.

Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Kopf, Edward Yang, Zachary DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. Pytorch: An imperative style, high-performance deep learning library. In H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett (eds.), *Advances in Neural Information Processing Systems 32*, pp.

8024–8035. Curran Associates, Inc., 2019. URL http://papers.neurips.cc/paper/9015-pytorch-an-imperative-style-high-performance-deep-learning-library.pdf.

Prajit Ramachandran, Barret Zoph, and Quoc V. Le. Searching for Activation Functions. *arXiv:1710.05941 [cs]*, October 2017. URL http://arxiv.org/abs/1710.05941. arXiv: 1710.05941.

Marco Tulio Ribeiro, Sameer Singh, and Carlos 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*, KDD '16, pp. 1135–1144, New York, NY, USA, 2016. Association for Computing Machinery. ISBN 9781450342322. doi: 10.1145/2939672.2939778. URL https://doi.org/10.1145/2939672.2939778.

Marco Tulio Ribeiro, Sameer Singh, and Carlos Guestrin. Anchors: High-precision model-agnostic explanations. In *AAAI Conference on Artificial Intelligence*, 2018. URL https://www.aaai.org/ocs/index.php/AAAI/AAAI18/paper/view/16982.

Laura Rieger, Chandan Singh, W. James Murdoch, and Bin Yu. Interpretations are useful: penalizing explanations to align neural networks with prior knowledge. Proceedings of Machine Learning Research, Vienna, Austria, 12–18 July 2020. URL http://https://arxiv.org/abs/1909.13584.

Andrew Slavin Ross, Michael C. Hughes, and Finale Doshi-Velez. Right for the Right Reasons: Training Differentiable Models by Constraining their Explanations. *arXiv:1703.03717 [cs, stat]*, May 2017. URL http://arxiv.org/abs/1703.03717. tex.ids: rossRightRightReasons2017a arXiv: 1703.03717.

Lloyd S Shapley. A value for n-person games. *Contributions to the Theory of Games*, 2(28):307–317, 1953.

Avanti Shrikumar, Peyton Greenside, Anna Shcherbina, and Anshul Kundaje. Not just a black box: Learning important features through propagating activation differences. *CoRR*, abs/1605.01713, 2016. URL http://arxiv.org/abs/1605.01713.

Avanti Shrikumar, Peyton Greenside, and Anshul Kundaje. Learning important features through propagating activation differences. In Doina Precup and Yee Whye Teh (eds.), *Proceedings of the 34th International Conference on Machine Learning*, volume 70 of *Proceedings of Machine Learning Research*, pp. 3145–3153, International Convention Centre, Sydney, Australia, 06–11 Aug 2017. PMLR. URL http://proceedings.mlr.press/v70/shrikumar17a.html.

Karen Simonyan, Andrea Vedaldi, and Andrew Zisserman. Deep inside convolutional networks: Visualising image classification models and saliency maps. In *Workshop at International Conference on Learning Representations*, 2014.

Chandan Singh, W. James Murdoch, and Bin Yu. Hierarchical interpretations for neural network predictions. In *International Conference on Learning Representations*, 2019. URL https://openreview.net/forum?id=SkEqro0ctQ.

Daniel Smilkov, Nikhil Thorat, Been Kim, Fernanda B. Viégas, and Martin Wattenberg. Smoothgrad: removing noise by adding noise. *CoRR*, abs/1706.03825, 2017. URL http://arxiv.org/abs/1706.03825.

Mukund Sundararajan and Amir Najmi. The many shapley values for model explanation. Proceedings of Machine Learning Research, Vienna, Austria, 12–18 July 2020. URL http://https://arxiv.org/abs/1908.08474.

Mukund Sundararajan, Ankur Taly, and Qiqi Yan. Axiomatic attribution for deep networks. In Doina Precup and Yee Whye Teh (eds.), *Proceedings of the 34th International Conference on Machine Learning*, volume 70 of *Proceedings of Machine Learning Research*, pp. 3319–3328, International Convention Centre, Sydney, Australia, 06–11 Aug 2017. PMLR. URL http://proceedings.mlr.press/v70/sundararajan17a.html.

Xiaoqian Wang, Hong Chen, Jingwen Yan, Kwangsik Nho, Shannon L Risacher, Andrew J Saykin, Li Shen, Heng Huang, and for the ADNI. Quantitative trait loci identification for brain endophenotypes via new additive model with random networks. *Bioinformatics*, 34(17):i866–i874, 2018. ISSN 1367-4803. doi: 10.1093/bioinformatics/bty557. URL https://doi.org/10.1093/bioinformatics/bty557. _eprint: https://academic.oup.com/bioinformatics/article-pdf/34/17/i866/25702591/bty557.pdf.

Han Xiao, Kashif Rasul, and Roland Vollgraf. Fashion-mnist: a novel image dataset for benchmarking machine learning algorithms, 2017.

Bing Xu, Naiyan Wang, Tianqi Chen, and Mu Li. Empirical Evaluation of Rectified Activations in Convolutional Network. *arXiv:1505.00853 [cs, stat]*, November 2015. URL http://arxiv.org/abs/1505.00853. arXiv: 1505.00853.

Chih-Kuan Yeh, Cheng-Yu Hsieh, Arun Suggala, David I Inouye, and Pradeep K Ravikumar. On the (in)fidelity and sensitivity of explanations. In H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett (eds.), *Advances in Neural Information Processing Systems 32*, pp. 10967–10978. Curran Associates, Inc., 2019. URL http://papers.nips.cc/paper/9278-on-the-infidelity-and-sensitivity-of-explanations.pdf.

A Proof of Theorem 12 & Lemma 5

Proof of Lemma 5. For each row of $W \triangleq AZ$, we can show that it is the Shapley value for $\tilde{f}^{(k)}$ using the linearity of Shapley values (the original motivating axiom of Shapley values, see Shapley (1953, Axiom 4 & Axiom 5)):

$$\boldsymbol{w}_k = \sum_j a_{k,j}\phi(f^{(j)}, \boldsymbol{x}) = \phi(\sum_j a_{k,j} f^{(j)}, \boldsymbol{x}) = \phi(\tilde{f}^{(k)}, \boldsymbol{x})\,. \tag{7}$$

Combining this result for all rows with Def. 2, we arrive at the result. □

We define a feature selector function $\pi^{(w)}$ as follows:

Definition 16 (Feature selector function)**.** $\pi^{(w)} : \mathbb{R}^d \mapsto \mathbb{R}^{|w|}$ *(for any $w \in \mathcal{W}$, where $\mathcal{W}$ again indexes the set of all component Shapley modules in a given Shallow* SHAPNET *as in Def. 11) is defined to select the input features by the index w, which contains the indices of the supposed input features of $F^{(w)}$:*

$$\pi^{(w)}(\boldsymbol{x}) = [x_i] \text{ where } i \in w. \tag{8}$$

Lemma 17. *The composite function $F'^{(w)}$ for a given Shapley module $F^{(w)}$, defined as*

$$F'^{(w)} = F^{(w)} \circ \pi^{(w)}, \tag{9}$$

has the following Shapley values for its input features

$$\phi_i\left(F'(x), \boldsymbol{x}\right) = \begin{cases} \phi\left(F^{(w)}, \boldsymbol{x}\right) = \alpha_i^{(w)} & \text{if } i \in w \\ 0 & \text{otherwise}, \end{cases} \tag{10}$$

Proof. This holds since the absence (set to reference values) of a feature does not make a difference to the final output of $F'^{(w)}$. □

Now we are ready to present proof for Theorem 12:

Proof. We consider a simpler, and yet more general case: consider where each of the modules defined in Def. 7, we first transform any Shapley module $F^{(w)}$ in the given Shallow SHAPNET G to $F'^{(w)}$. Then, by Def. 11, the zeros in Lemma 17 do not make any difference whether to the output of G or its associated β_i's, and yet β_i's are nothing but summations of Shapley values of different $F^{(w)}$'s and by Lemma 5,

$$\beta_i = \phi_i\left(G, \boldsymbol{x}\right), \tag{11}$$

which completes the proof. □

B PROOF OF THEOREM 15

Proof. The proof of local accuracy is trivial by construction since $\mathcal{H}(\boldsymbol{x}) \triangleq \text{sum}^{[d]}(h(\boldsymbol{x}))$ from Def. 14. The proof of missingness is straightforward via induction on the number of layers, where the claim is that if $x_i = r_i$ (i.e., the input for one feature equals the reference value), then $\forall \ell,\ h_i^{(\ell)}(\boldsymbol{x}) = 0$ (i.e., the pre-summation outputs of the Shapley modules in all layers corresponding to feature i will all be zero). For the base case of the first layer, we know that if $x_i = r_i$, then we know that $Z_i^{(1)} = \mathbf{0}$ by the construction of Shapley modules. For the future layers, the inductive hypothesis is then that if previous layer representation $Z_i^{(\ell-1)}$ of a feature i is $\mathbf{0}$, then the representation from the current layer ℓ for this feature remains the same, i.e., $Z_i^{(\ell)} = \mathbf{0}$. Because the reference values for all layers except the first layer are zero, then we can again apply the property of our Shapley modules and thus the inductive hypothesis will hold for $\ell > 1$. Thus, the the claim holds for all layers ℓ, and $h_i(\boldsymbol{x}) = \text{sum}^{[c]} Z_i^{(\ell)}$ will also be zero—which proves missingness for Deep SHAPNETS. □

C EXTENDED LITERATURE REVIEW

Post-hoc feature-attribution explanation methods, gained traction with the introduction of LIME explanations (Ribeiro et al., 2016) and its variants (Ribeiro et al., 2018; Guidotti et al., 2018a;b), which formulate an explanation as a local approximation to the model near the target instance. LIME attempts to highlight which features were important for the prediction. Saliency map explanations for image predictions based on gradients also assign importance to each feature (or pixel) (Simonyan et al., 2014; Shrikumar et al., 2017). SHAP explanations attempt to unify many previous feature attribution methods under a common additive feature attribution framework (Lundberg & Lee, 2017; Lundberg et al., 2020), which is the main motivation for our work.

Intrinsically interpretable models are an alternative to post-hoc explanation methods. In particular, GAM (Lou et al., 2012) and its extension (Lou et al., 2013; Caruana et al., 2015) construct models that can be displayed as series of line graphs or heatmaps. Because the entire model can be displayed and showed to users, it is possible for a user to directly edit the model based on their domain knowledge. In addition, Alvarez Melis & Jaakkola (2018) proposed an architecture that are complex and explicit if trained with a regularizer tailored to that architecture. Our methodology sits *in between* post-hoc and intrinsic approaches by constructing an explainable model but using SHAP post-hoc explanation concepts and allowing more complex feature interactions.

Recent works also focus on Shapley value approximations in deep neural nets, where Deep Approximate Shapley Propagation (DASP) approximates in polynomial time (Ancona et al., 2019), L-Shapley and C-Shapley do in linear (Chen et al., 2019) with the Markov assumption, and our method performs approximation with a constant complexity, which additionally allows regularizing the explanations according to human priors and hence modifying the underlying model during training. This falls in line with previous works (Ross et al., 2017; Liu & Avci, 2019; Erion et al., 2019; Rieger et al., 2020) each of which adds an extra term in the loss function measuring the disagreement between the explanations and human priors. Ross et al. (2017) proposed to regularize the input gradients as explanations, which creates the need for second-order gradient. Liu & Avci (2019) uses Integrated Gradients (IG) (Sundararajan et al., 2017), which is an extension to Shapley values (Sundararajan & Najmi, 2020). Due to the use of IG, several samples (for one explanation) are additionally warranted. Erion et al. (2019) proposed Expected Gradient to rid of reference values in IG and suggested the usage of a single sample. Rieger et al. (2020) adds to this line of work with Contextual Decomposition (Murdoch et al., 2018; Singh et al., 2019) which avoids the need of computation of second-order gradients and extra samples. Our work differs with these four in that we do not need a second order gradient and that we are focusing on the more theoretically-grounded Shapley values. The explanation-related regularization in Alvarez Melis & Jaakkola (2018) differs with ours in that the regularizer is used to *promote* interpretability and ours uses regularization to modify the underlying model to conform to prior belief. We also draw close similarity to works that that attempt to smooth saliency map explanations (Smilkov et al., 2017; Sundararajan et al., 2017; Yeh et al., 2019) which acts on explanation methods to induce certain properties in explanations but not the actual models.

Table 3: Time for computing explanations (averaged over 1000 runs)

Explanations \ Models	Deep SHAPNET	DNN (eq. comp.)	DNN (eq. param.)
Ours	20.47	N/A	N/A
Deep SHAP	46.10	83.38	8.56
Kernel SHAP (40)	60.78	130.04	11.52
Kernel SHAP (77)	72.61	201.88	13.53
Kernel SHAP (def.)	598.79	789.08	142.85

D WALL-CLOCK TIME FOR EXPLANATION EXPERIMENT

To validate the efficiency of computing an explanation, we compare the wall-clock time of our SHAPNET explanations to other approximation methods related to Shapley-values including Deep SHAP and Kernel SHAP with different numbers of samples—40, 77, and default ($2 \times d + 2^{11}$). The wall-clock computation time in seconds can be seen in Table 3. We note importantly that this comparison is a bit odd because other Shapley-value explanation methods are *post-hoc* explanations, whereas SHAPNET is both a model and an intrinsic explanation method. Thus, the most fair comparison in terms of time is to have the post-hoc explanation methods explain our SHAPNET model as seen in the first column of Table 3. Nevertheless, we also show the wall-clock times for the post-hoc methods on our comparable DNNs in columns two and three of Table 3—note that SHAPNET cannot explain other DNNs hence the N/A's. From the first column, we can see that our method is inherently faster than the others when applied to our model. However, smaller model (i.e., DNN w. equivalent parameters) may come with faster explanations.

E LOWER-DIMENSIONAL COMPARISON BETWEEN SHAPNET EXPLANATIONS AND POST-HOC EXPLANATIONS

For lower-dimensional datasets, we seek validation for the explanations $h(\boldsymbol{x})$ provided by Deep SHAPNETs by comparing to the true Shapley values of our models (which can be computed exactly or aproximated well in low dimensions). Our metric is the normalized ℓ_1 norm between the true Shapley values denoted ϕ and the explanation denoted γ defined as:

$$\ell(\phi, \gamma) = \frac{\|\phi - \gamma\|_1}{d \sum_i \phi_i}. \tag{12}$$

Note that we measure the average difference from vector-output classifiers by means discussed in subsection E.1. The experimental setup and results are presented in subsection E.2.

E.1 COMPARING SHAP APPROXIMATION ERRORS WITH VECTOR OUTPUT MODELS

To compare the approximation errors for every output of the model, we use a weighted summation of the explanations for different outputs, where the weights are the models' output after passing into the soft-max function. Thus, we up weight the explanations that have a higher probability rather than taking a simple average. Thus, in the extremes, if the predicted probability for a label is zero, then the difference in explanation is ignored, while if the predicted probability for a label is one, then we put all the weight on that single explanation.

Concretely, for a classification model $C : \mathbb{R}^d \mapsto \mathbb{R}^n$, where d is the number of input features and n is the number of classes, we have the output vector of such model with an input instance $\boldsymbol{x} \in \mathbb{R}^d$ as $C(\boldsymbol{x}) \in \mathbb{R}^n$. For each output scalar value $[C(\boldsymbol{x})]_j$, we compute the approximation of the SHAP values w.r.t. that particular scalar for feature indexed i, denoted by $\gamma_i^{(j)}$, and hence its corresponding errors as measured in normalized ℓ_1 distance as discussed in Eqn. 12: $\ell_1^{(j)}$. The final error measure that we compare between classifiers ℓ_1^* is simply a weighted sum version of the normalized ℓ_1:

$$\ell_1^* = \sum_{j=1}^{n} [\text{softmax}(C(\boldsymbol{x}))]_j \cdot \ell_1^{(j)},$$

where the softmax is taken on the non-normalized output of the classifiers. Note that ℓ_1^j is also taken on the raw output.

E.2 SETUP AND RESULTS

Because the explanation difference from SHAP is independent of the training procedure but solely depends on the methods, we compare the performance of the explanation methods on untrained models that have randomly initialized parameters (i.e., they have the model structure but the parameters are random)—this is a valid model it is just not a model trained on a dataset. Additionally, we consider the metric after training the models on the synthetic, yeast, and breast cancer datasets. Note that for even a modest number of dimensions, computing the exact Shapley values is infeasible so we use a huge number of samples (2^{16}) with the kernel SHAP method to approximate the true Shapley values as the ground truth for all but the Yeast dataset, which only requires 2^8 samples for exact computation. Note that the extended Shapley value (ext.) requires 2^{14} times of model evaluation for a single explanation which is already extremely expensive, and thus included merely for comparison. Results of the average difference can be seen in Table 4. From the results in Table 4, we can see that indeed our Shallow SHAPNET model produces explanations that are the exact Shapley values (error by floating point computation) as Theorem 12 states. Additionally, we can see that even though our Deep SHAPNET model does not compute exact Shapley values, our explanations are actually similar to the true Shapley values.

We also provide in Table 5 the standard deviation of the approximation errors as in Table 4. These are the same runs as those in Table 4.

Table 4: Average difference from exact Shapley values for different models

Models \ Explanations	Ours	DeepSHAP	Kernel SHAP (77)	Kernel SHAP (def.)	Kernel SHAP (ext.)
Untrained models (100 independent trials each entry)					
Deep SHAPNET	1.29e-06	2.89e-05	1.42e+03	2.13e+05	5.00e-11
Shallow SHAPNET	2.97e-07	0.802	0.0733	4.05e-03	1.05e-03
Regression models on synthetic dataset (20 independent trials each entry)					
Deep SHAPNET	4.99e-03	3.71	4.91e-03	3.05e-04	2.26e-05
Shallow SHAPNET	4.19e-08	3.88	4.05e-03	2.39e-04	2.75e-05
Classification models on Yeast dataset (50 independent trials each entry)					
Deep SHAPNET	0.0504	0.307	0.01467	0	0
Shallow SHAPNET	2.20e-07	0.516	0.0132	0	0
Classification models on Breast Cancer Wisconsin (Diagnostic) (50 independent trials each entry)					
Deep SHAPNET	0.0167	0.120	0.0369	4.09e-03	8.05e-04
Shallow SHAPNET	1.20e-04	0.0581	0.0224	1.91e-03	2.03e-04

Table 5: Standard deviation of Shapley approximation errors on different datasets

Models \ Explanations	Ours	DeepSHAP	Kernel SHAP (77)	Kernel SHAP (def.)	Kernel SHAP (ext.)
Untrained models					
Deep SHAPNET	0.734	5.23	7.67e+08	4.25e+09	9.20e-03
Shallow SHAPNET	7.55e-07	1.57	0.124	8.02e-03	2.98e-03
Regression models on synthetic dataset					
Deep SHAPNET	3.28e-03	5.15	3.13e-03	8.79e-05	2.23e-05
Shallow SHAPNET	3.09e-04	1.28	4.75e-03	1.62e-04	6.33e-06
Classification models on Yeast dataset					
Deep SHAPNET	0.105	0.358	0.0288	0	0
Shallow SHAPNET	5.35e-07	0.894	0.0304	0	0
Classification models on Breast Cancer Wisconsin (Diagnoistic)					
Deep SHAPNET	0.0104	0.0359	0.0987	3.61e-03	1.72e-03
Shallow SHAPNET	0.0109	0.246	0.0141	1.93e-03	2.91e-04

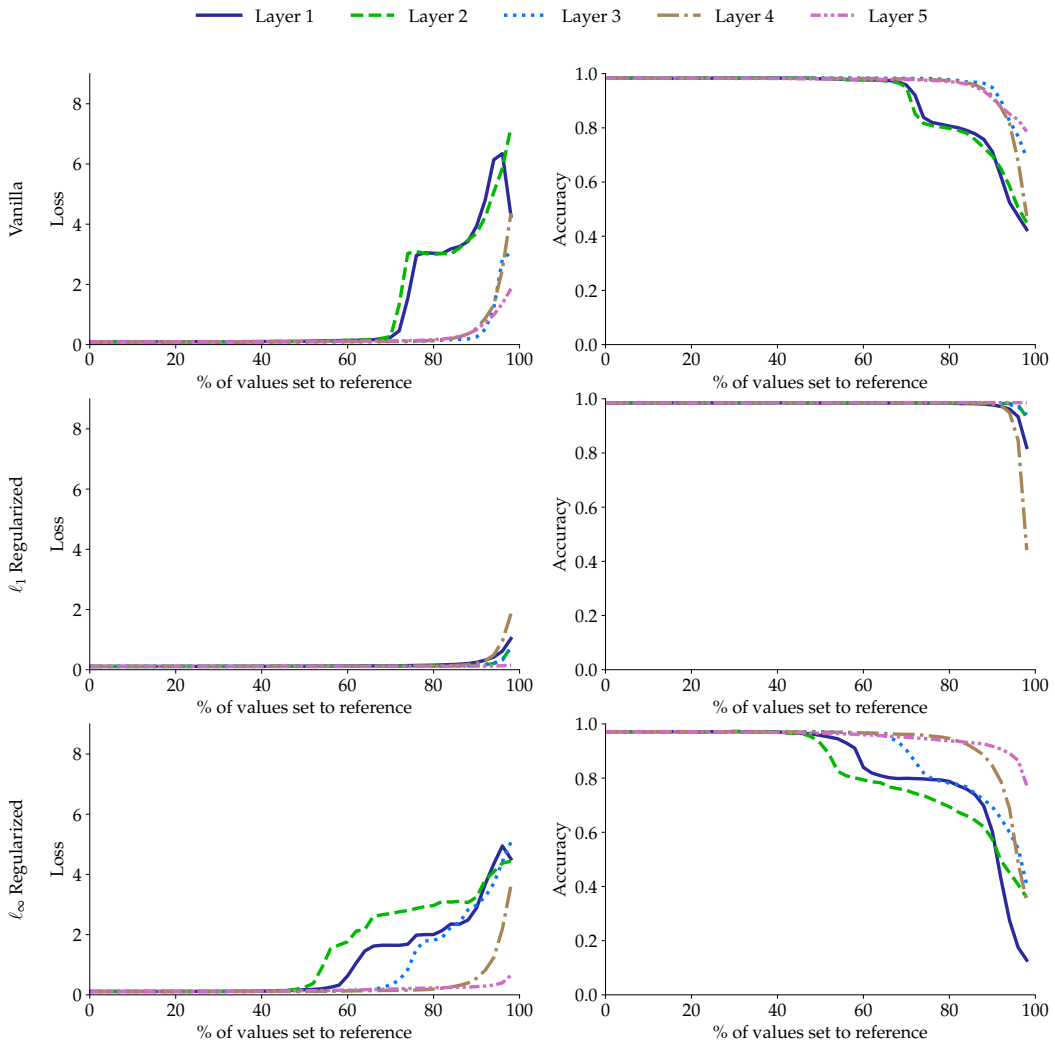

Figure 7: 1) Removing values from earlier layers seems to have a stronger effect on the model's performance than removing from later layers. In addition, it seems most of the values can be removed and retain the accuracy. 2) ℓ_1 regularized model performs better under pruning across all layers while ℓ_∞ performs the worst, which is expected as it tries to spread out the importance among input features.

F PRUNING EXPERIMENT

We also conduct a set of pruning (set to reference values) experiment. This is, again, enabled by the layer-wise explainable property of Deep SHAPNET. Recent work has investigated the usage of Shapley values for pruning (Ancona et al., 2020), but our work means to showcase the ability even further. Results are shown in Fig. 7.

We perform the pruning experiment with all three MNIST-traiend models as discussed in subsection 2.3 with no, ℓ_1, and ℓ_∞ regularizer, respectively. The values are removed (set to reference) by the order of their magnitude (from smallest to largest). We can see that most of the values might be dropped and the model performance stays unhinged. Moreover, ℓ_1 seems to have the best robustness against pruning, which makes sense as it induces sparsity in explanation values. Future works are to be done on the computational cost reduction under such pruning technique.

G MORE EXPERIMENTAL DETAILS

For all the experiments described below, the optimizers are Adam (Kingma & Ba, 2014) with default parameters in PyTorch (Paszke et al., 2019).

G.1 TIMING EXPERIMENTS

For Table 3, the timing is for the untrained (randomly initialized) models with 16 features. The numbers after Kernel SHAP indicates the number of samples allowed for Kernel SHAP's sampling method. Time is measured in seconds on CPU with 1000 independent trials per cell as Kernel SHAP supports solely CPU from the implementation of the authors of Lundberg & Lee (2017). The model of the CPU used for testing is Intel(R) Xeon(R) Silver 4114 CPU @ 2.20GHz.

We perform explanation on randomly initialized 16-feature Deep SHAPNET model for 1000 rounds. The 16-feature setup means we will have $(16/2) \times \log_2 16 = 32$ Shapley modules inside the entire model spanned out in 4 stages according to the Butterfly mechanism. The inner function f for every module in this setting is set as a fully-connected neural network of 1 hidden layer with 100 hidden units. For the two reference models, the model equivalent in computation time has 8 layers of 1800 hidden units in each layer, while the model equivalent in parameters has 11 layers of 123 hidden units in each layer. The output of the models are all scalars.

G.2 YEAST AND BREAST CANCER DATASETS EXPERIMENT SETUPS

This setup applies for the experiments involving both datasets, including those experiments for model performance and those that showcase the ability for explanation regularization. The experiments on the two datasets were run for 10 rounds each with different random initialization and the end results were obtained by averaging. For each round, we perform 5-fold cross validation on the training set, and arrive at 5 different models, giving us 50 different models to explain for each datasets. For the two datasets, we do 9 training epochs with a batch size of 32.

For the preprocessing procedures, we first randomly split the data into training and testing sets, with 75% and 25% of the data respectively. Then we normalize the training set by setting the mean of each feature to 0 and the standard deviation to 1. We use the training sets' mean and standard deviation to normalize the testing set, as is standard practice.

The inner functions of the Shapley modules are fully connected multilayer neural networks. The first and second stages of two models on the two datasets are identical in structure (not in parameter values). The first stage modules have 2 inputs and 25 outputs, with a single hidden layer of size 50. The modules in the second stage have 100 input units and 50 output units with two hidden layer of size 100. The non-linearity is introduced by ReLU (Nair & Hinton, 2010) in the inner functions. The rest of the specifics about the inner function are discussed below. Note that the dimensionality of the output from the last stage differs with that of the input to the next stage, as is expected by Shapley module where the input has twice the dimensionality of that of the output.

Model for Yeast dataset The Yeast dataset has 8 input features and hence 4 Shapley modules at each of the $\log_2 8 = 3$ stages. Within each stage, the structure of the inner function is identical except for the parameters. For the Yeast dataset, the third (last) stage comprises four-layer fully-connected model with inputs from $50 \times 2 = 100$ units to the number of classes, which is 10, with two hidden layers of 150 units.

Model for Breast Cancer Wisconsin (Diagnostic) This dataset has 30 input features, which is not a exponent of 2, but we can still construct a Deep SHAPNET by setting two extra variables to zero. In fact, in doing so we can simplify the Shapley modules since we know which of the modules in the network always have one or both of its input being 0 at all times. To construct the model, we realize that the number of stages is 5 and the number of modules at each stage is 16. For the Breast Cancer Wisconsin (Diagnostic) dataset, in the the third stage of the model, we use a model with 100 inputs units and 75 output units with two hidden layers of 150 units. The fourth stage has input units of $75 \times 2 = 150$ and output of 100 units with two hidden layers of size 200. The fifth (last) stage has a input dimension of 200, two hidden layers of 250 units and output of size 10.

G.3 SPECIFICS FOR VISION TASKS

G.3.1 STRUCTURE USED FOR VISION EXPERIMENTS

For all of the vision tasks, we use the same structure with the exception of input channel counts. We first define a base structure that we use in different Shallow SHAPNET layers, in each of which we simply change the number of parameters.

Base structure This structure has 4 hyper-parameters with non-linearity introduced by Swish :

1. *Input dimension*: This is the number of features d times the number of meta-channels (Remark 9) n. This will determine the dimensionality of input for the underlying function $f^{(j)}$, which is $\mathbb{R}^{(d\times n)}$, and hence the number of neurons in the first linear layer.
2. *number of hidden layers*: this is very self-evident by name. Each of these is Linear-Swish(Ramachandran et al., 2017)-LayerNorm(Ba et al., 2016). Also, we add residual connections wherever possible.
3. *Hidden size*: the number of neurons in the hidden linear layers.
4. *Output size* (n'): the number of output neurons in the final linear layer.

The output of the first linear layer is fed into the hidden layers after activated by Swish function, which is

$$\text{Swish}(x) = x \cdot \frac{1}{e^{-\beta\cdot x}+1},$$

where β is fixed to be 1 in our setting. We found activation functions like Swish and LeakyReLU (Xu et al., 2015) can reduce overfitting in our case as opposed to ReLU.

Note that $n' \times d$ from the previous layer should be matching the *Input dimension* hyper-parameter with the exception of factoring out (Dinh et al., 2017; Kingma & Dhariwal, 2018), where we take a subset of the meta-channels of the previous layer's output representation as the input meta-channels to the current Shallow SHAPNET.

Note that the inner functions are shared within one Shallow SHAPNET. For all of MNIST, FashionMNIST, and Cifar-10 dataset, we present our structure in Table 6.

Table 6: Hyper-parameter settings for each Shallow SHAPNET of a Deep SHAPNET for images. For Cifar-10, input dimension of the inner function of the first Shallow SHAPNET is 12 (3×4), and 4 (1×4) for MNIST or FashionMNIST.

Shallow Index \ Hyper-parameters	*Input dimension*	*number of hidden layers*	*Hidden size*	*Output size*
1	12 or 4	1	64	128
2	256	1	64	64
3	256	2	64	64
4	256	2	64	64
5	512	2	128	10

À-trous convolution À-trous convolution is used for learning hierarchical representations of images. In our model, the dilation in the sliding window operation is set to double each stage: 1 for the first, 2 for the second, 4 for the third, 8 for the fourth and 16 for the last stage.

G.3.2 TRAINING RECIPE

Learning process As discussed before, all the models are trained with Adam in PyTorch with the default parameters. The batch size is set to 64 for all the vision tasks. No warm-up or any other scheduling techniques are applied. The FashionMNIST & MNIST models in Table 1 are trained for 10 epochs while the MNIST models with which we investigate the new capabilities (layer-wise explanations & Shapley explanation regularizations) in Section 3 are trained with just 5 epochs. The Cifar-10 models are trained for 120 epochs.

Preprocessing For both MNIST dataset and FashionMNIST dataset, we normalize the training and testing data with mean of 0.1307 and standard deviation of 0.3081, this gives us non-zero background and we would therefore set the reference values to 0's, allowing negative or positive contribution from the background. Both FashionMNIST and MNIST share the same preprocessing pipeline: 1) for training dataset, we extend the resolution from 28×28 to 36×36 and then randomly crop to 32×32, and then normalize. 2) for testing data, we simply pad the image to 32×32 and normalize them.

For Cifar-10 dataset, during training, we first pad 4 pixels on four sides of the image and then randomly crop back to 32×32, perform a horizontal flip with probability 0.5, and normalize the data with means $0.4914, 0.4822, 0.4465$ and standard deviaiton of $0.2023, 0.1994, 0.2010$ for each channel, respectively. For testing, we simply normalize the data with the same mean and standard deviation.

G.4 Illustrations of explanation regularization experiments

We present visual representations of our models' explanation on MNIST (LeCun & Cortes, 2010) dataset in Fig. 8, Fig. 9, and Fig. 10. In all three figures, the gray images from left to right are explanations laid out for all 10 classes from 0 to 9, where the red pixels indicate positive contribution and blue pixels indicate negative contribution, with magnitudes presented in the color bar below.

From an intuitive visual interpretation, we argue that our explanation regularizations are indeed working in the sense that ℓ_∞ regularization is indeed smoothing the explanations (to put more weight on a large number of pixels) and that ℓ_1 regularization limits the number of pixels contributing to the final classification.

One can notice that, by comparing the color bars below the figures, the contribution magnitudes for ℓ_∞ and ℓ_1 regularized models are lower than those of the unregularized counterpart. We have expected this to happen as both regularization techniques require to make one or more of the attribution values to be smaller. However, we note that this change in scale of the attribution values should not impact the accuracy of the prediction as the softmax operation at the end will normalize across the outputs for all of the classes.

More interesting discussions are provided in the caption of Fig. 8, Fig. 9, and Fig. 10, and we strongly encourage readers to have a read.

G.5 Complete visualization for progression in Deep SHAPNet

We also provide results of the progression visualization in three Deep SHAPNETs. To visualize such explanations, we need to encode a tensor into a single scalar value, in Fig. 5, Fig. 11, & Fig. 12, we simply take the average of all the explanation values for that pixel. We also provide a version in which we take the average of magnitude in Fig. 13 & Fig. 14. Note that Fig. 5, Fig. 11, & Fig. 13 are scaled version of the explanations, since the first few layers have relatively lower magnitude in scalar values in the representation. The unscaled versions are in Fig. 12 & Fig. 14.

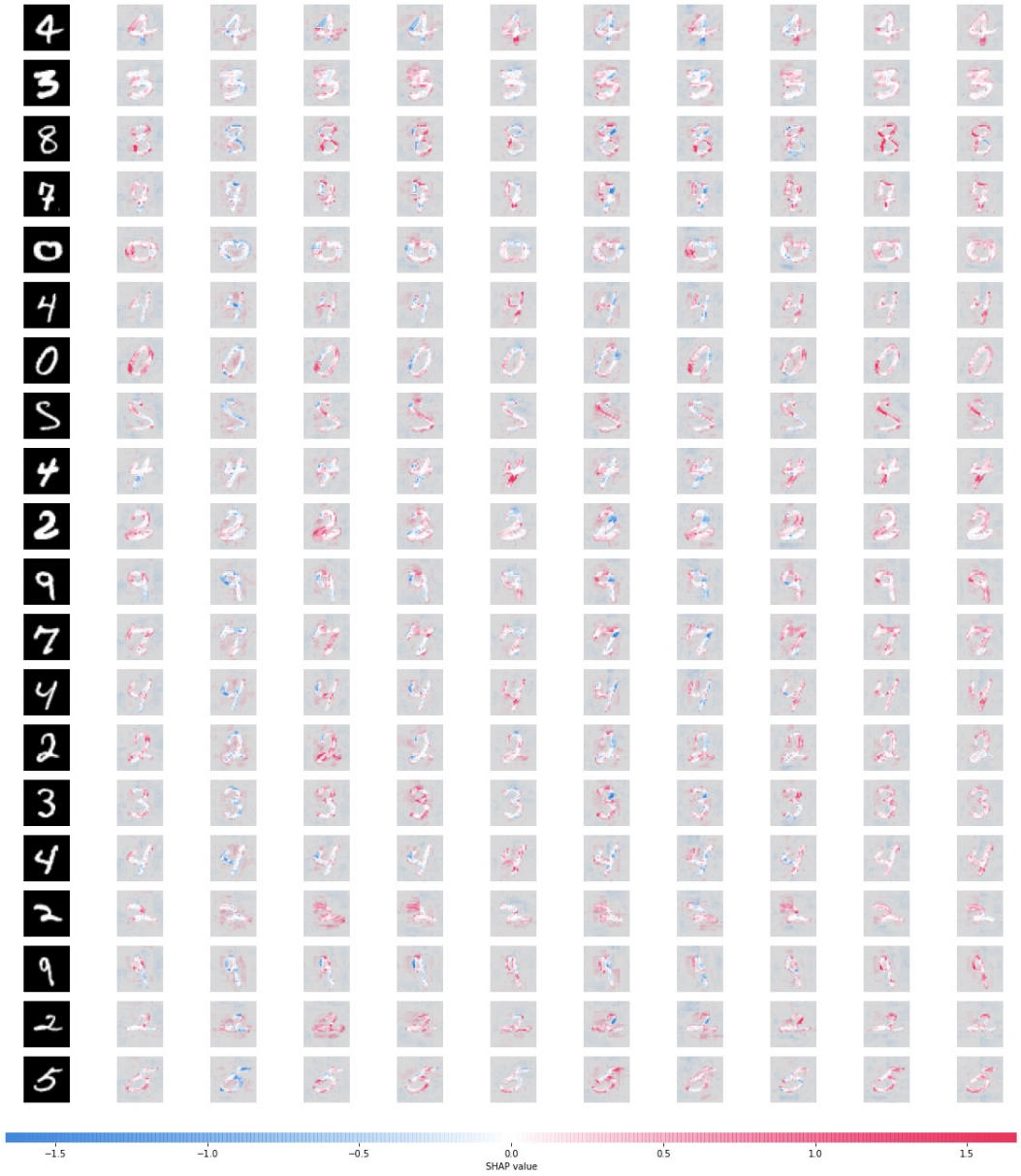

Figure 8: The explanations produced by our model trained without explanation regularization for all 10 classes. From left to right are class index (also corresponding digits) from 0 to 9.

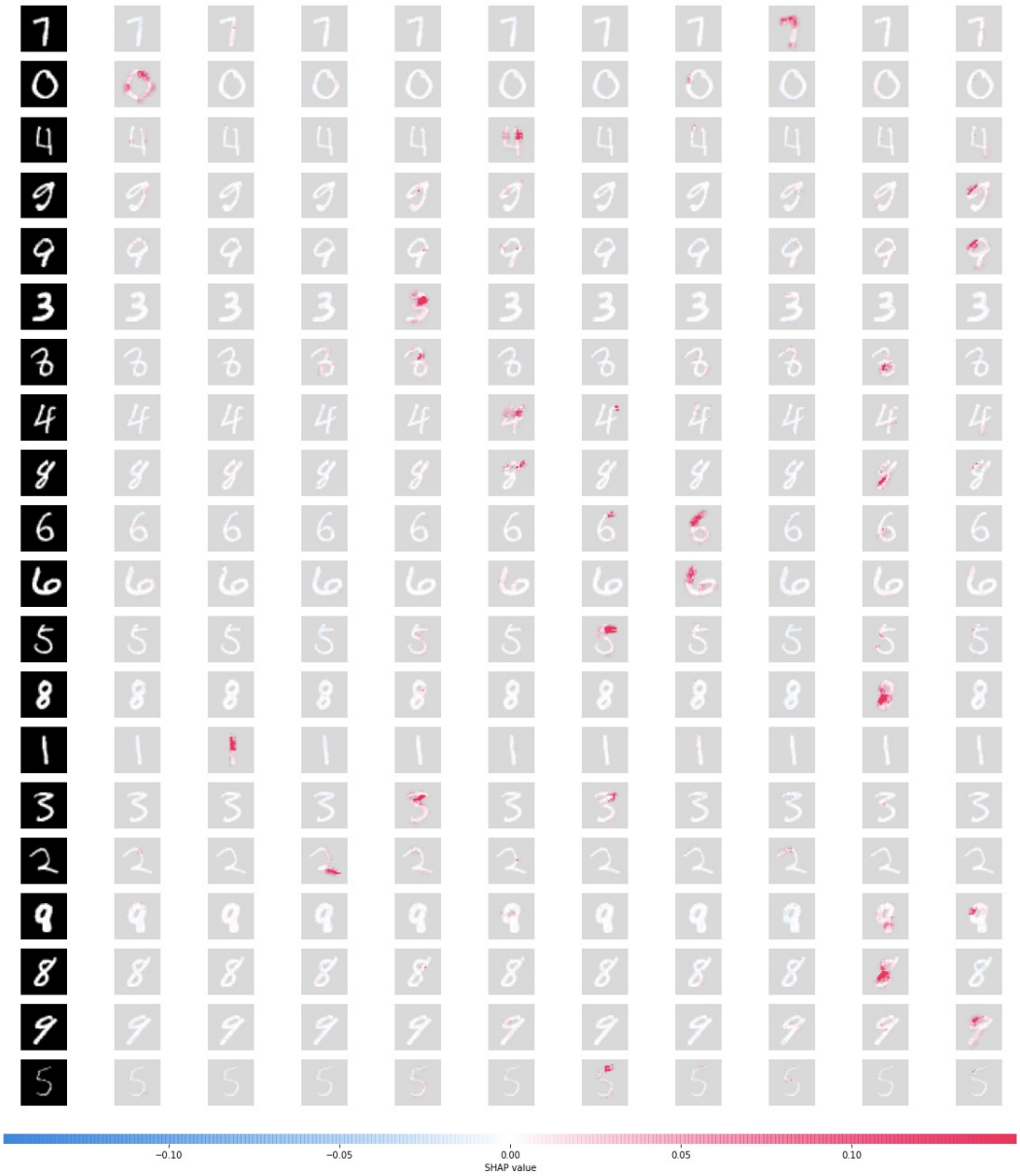

Figure 9: The explanations produced by our model trained with ℓ_1 explanation regularization for all 10 classes. Our goal with this regularization term is to promote sparsity in feature importance, which, by comparison with Fig. 8 and Fig. 10, one can easily argue has been successfully achieved. This shows that ℓ_1 regularization, combined with cross entropy loss, does (almost) remove negative contribution and only focuses on each digit's distinguishing features.

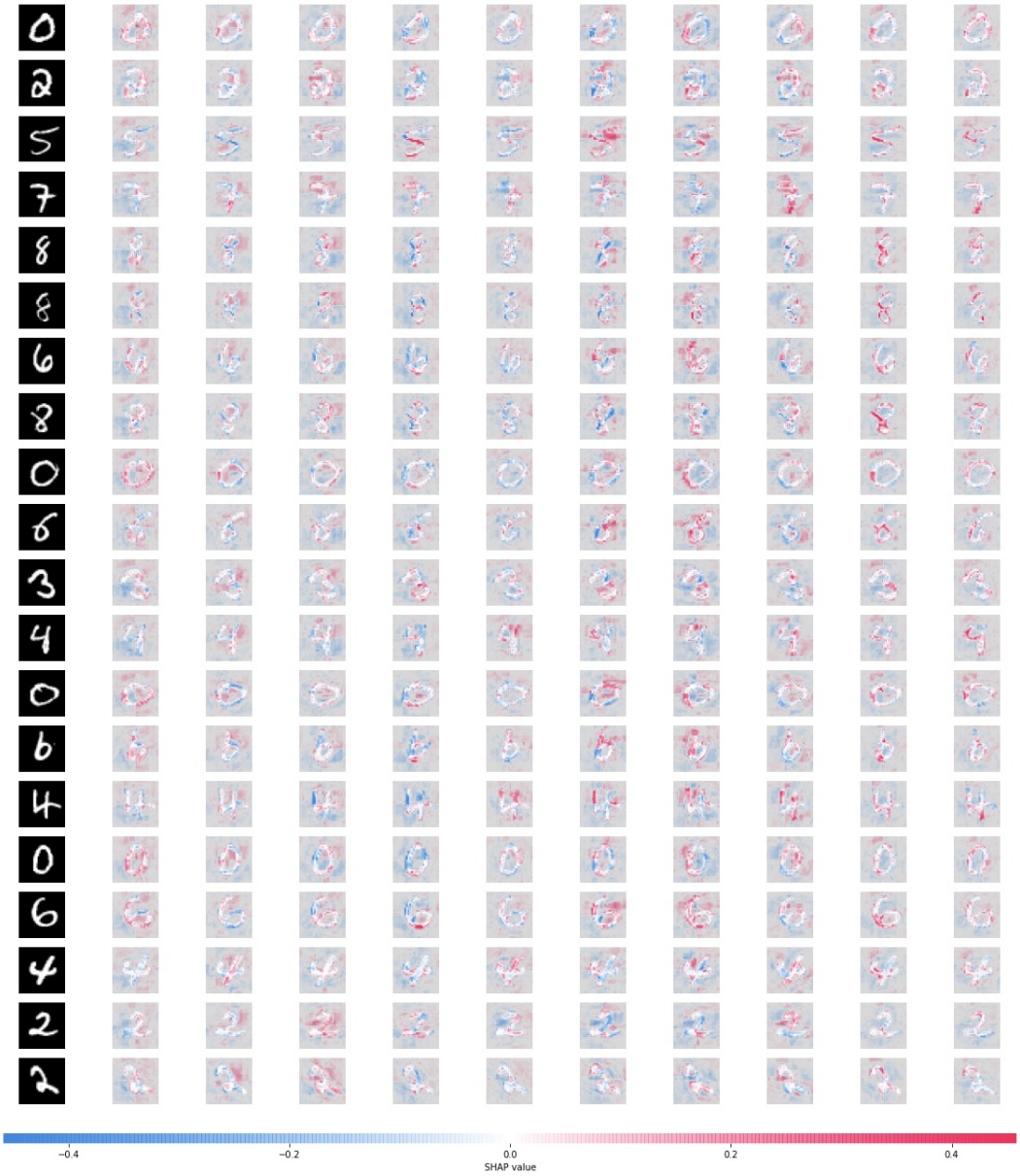

Figure 10: The explanations produced by our model trained with ℓ_∞ explanation regularization for all 10 classes. However, it is still easy to notice which part of the images contribute positively or negatively for a particular class. For example, look at the second last digit—a '2', where, in the left part of the digit under where there the stroke starts, there is a positive contribution. This indicates that the classifier recognize the absense of a complete circle and thereby is happy to classify this digit as 2. For another instance, if we are focusing on the first column, i.e., the digit 0, we can almost see a 0-shaped pattern in negative contribution if the input digit is not 0, which means the classifier is expecting such pattern if the input is to be classified as 0. From the figure, it is easy to conclude that with ℓ_∞ regularization, the contribution among pixels are more laid out and less concentrated than they would be other wise as shown in Fig. 8.

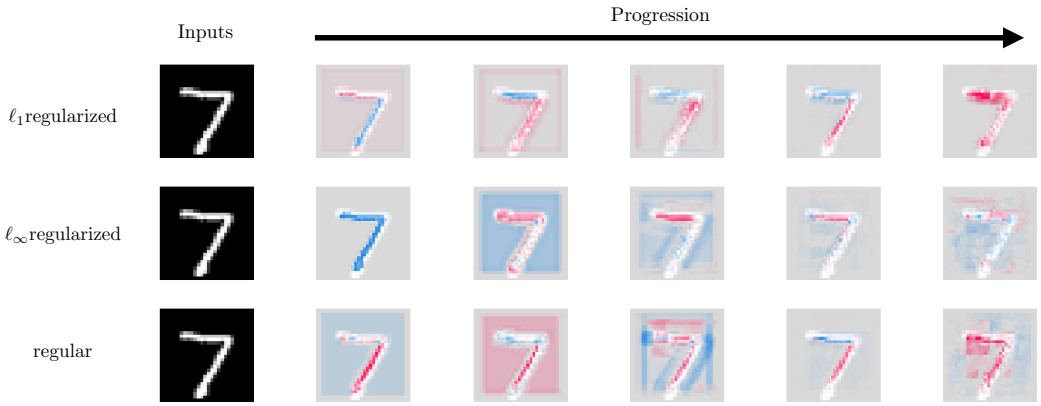

Figure 11: The complete version of provided in Fig. 5

Figure 12: The unscaled version of Fig. 11

Figure 13: The mean of absolute values version of Fig. 11

Figure 14: The unscaled mean of absolute values version of Fig. 11