HarsanyiNet: Computing Accurate Shapley Values in a Single Forward Propagation

Lu Chen [*1] **Siyu Lou** [*1 2] **Keyan Zhang** [1] **Jin Huang** [1] **Quanshi Zhang** [1]

Abstract

The Shapley value is widely regarded as a trustworthy attribution metric. However, when people use Shapley values to explain the attribution of input variables of a deep neural network (DNN), it usually requires a very high computational cost to approximate relatively accurate Shapley values in real-world applications. Therefore, we propose a novel network architecture, the *HarsanyiNet*, which makes inferences on the input sample and simultaneously computes the exact Shapley values of the input variables in a single forward propagation. The HarsanyiNet is designed on the theoretical foundation that the Shapley value can be reformulated as the redistribution of Harsanyi interactions encoded by the network.

1. Introduction

Explainable artificial intelligence (XAI) has received considerable attention in recent years. A typical direction of explaining deep neural networks (DNNs) is to estimate the salience/importance/contribution of an input variable (*e.g.*, a pixel of an image or a word in a sentence) to the network output. Related studies have been termed the *attribution methods* (Bach et al., 2015; Selvaraju et al., 2017; Sundararajan et al., 2017; Lundberg & Lee, 2017). In comparison with most attribution methods designed without solid theoretical supports, the Shapley value (Shapley, 1953) has been proved the only solution in game theory that satisfies the *linearity*, *dummy*, *symmetry*, and *efficiency* axioms (Young, 1985). Therefore, the Shapley value is widely considered a relatively trustworthy attribution for each input variable.

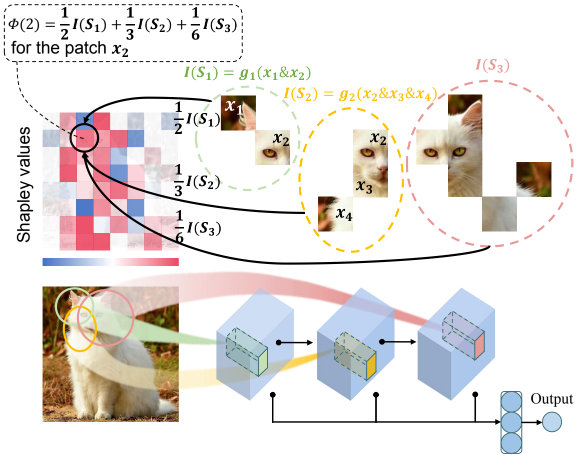

Figure 1. Overview of the HarsanyiNet. The HarsanyiNet encodes different Harsanyi interactions, each representing an AND relationship between different patches. Shapley values can be computed as re-allocation of Harsanyi interactions.

However, using the Shapley value in real-world applications is often impractical because (1) computing the exact Shapley value is NP-hard, and (2) existing approximation techniques (Castro et al., 2009; Lundberg & Lee, 2017) often confront a dilemma in that approximating Shapley values with an acceptable accuracy usually requires to conduct a huge number of network inferences.

Thus, in this paper, we aim to directly jump out of the above dilemma and design a neural network, namely *HarsanyiNet*, which simultaneously conducts model inference on the input sample and computes the exact Shapley value for each input variable in a single forward propagation[1].

Specifically, the theoretical foundation for the HarsanyiNet is that the Shapley value of an input variable can be reformulated as a redistribution of its different Harsanyi interactions (Harsanyi, 1963) encoded by the DNN. Formally, given a DNN and an input sample with n variables, a Harsanyi interaction S represents an AND relationship between the variables in S, which is encoded by the DNN. A DNN usually encodes many different Harsanyi interactions. Each Harsanyi interaction makes a specific numerical

[*]Equal contribution [1]Shanghai Jiao Tong University, China [2]Eastern Institute for Advanced Study, China. Correspondence to: Quanshi Zhang is the corresponding author. He is with the Department of Computer Science and Engineering, the John Hopcroft Center, at the Shanghai Jiao Tong University, China. <zqs1022@sjtu.edu.cn>.

Proceedings of the 40th International Conference on Machine Learning, Honolulu, Hawaii, USA. PMLR 202, 2023. Copyright 2023 by the author(s).

[1] `https://github.com/csluchen/harsanyinet`

contribution, denoted by $I(S)$, to the inference score of the DNN. Let us take the interaction between image patches $S=\{eye, nose, mouth\}$ as a toy example. If all these patches co-appear, then they form a face pattern and make a specific interaction effect $I(S)$ to the confidence score of face detection. Masking any patch will destroy this pattern and remove the interaction effect, *i.e.*, making $I(S)=0$.

Because it is proven that the Shapley value can be computed using Harsanyi interactions, the activation of each intermediate neuron in the HarsanyiNet is designed to represent a specific Harsanyi interaction. The intermediate neuron is termed as the *Harsanyi unit*. Such a network design enables us to derive the exact Shapley value of an input variable using activation scores of Harsanyi units.

The proposed HarsanyiNet has significant advantages over existing approaches for approximating Shapley values by conducting a single network inference.

• Existing approximation methods, *e.g.*, DeepSHAP (Lundberg & Lee, 2017) and FastSHAP (Jethani et al., 2021), estimate Shapley values with considerable errors, but the HarsanyiNet can generate **exact** Shapley values, which is both theoretically guaranteed and experimentally verified.

• The only existing work allowing to compute accurate Shapley values in a single forward propagation is the ShapNet (Wang et al., 2021). However, the ShapNet is designed to only encode interactions between at most k input variables ($k \ll n$, they set $k = 4$), and the computational cost of the ShapNet's inference (forward propagation) is 2^k times more than that of traditional networks. Alternatively, this study also provides another network (*i.e.*, Deep ShapNet) to encode more complex interactions, but it cannot guarantee the accuracy of the computed Shapley values. In comparison, the HarsanyiNet does not limit the number of the input variables within the interaction, thereby ensuring broader applicability and exhibiting significantly better performance.

Moreover, we implement two specific HarsanyiNets in the experiment, the *Harsanyi-MLP* extended from multi-layer perceptrons (MLP) and the *Harsanyi-CNN* developed on convolutional neural networks (CNN).

The contributions of this paper can be summarized as follows. (1) We propose a novel neural network architecture, the *HarsanyiNet*, which can simultaneously perform model inference and compute exact Shapley values in one forward propagation. (2) Following the paradigm of HarsanyiNet, we design Harsanyi-MLP and Harsanyi-CNN for tabular data and image data, respectively. (3) The HarsanyiNet does not constrain the representation of specific interactions, but it can still guarantee the accuracy of Shapley values.

2. Related Work

Estimating the importance/attribution/saliency of input variables represents a typical direction in XAI. In general, previous attribution methods usually computed the attributions of input variables based on gradient (Simonyan et al., 2014; Springenberg et al., 2015; Shrikumar et al., 2016; Selvaraju et al., 2017; Sundararajan et al., 2017), via back-propagation of attributions (Bach et al., 2015; Montavon et al., 2017; Shrikumar et al., 2017), and based on perturbations on the input variables (Ribeiro et al., 2016; Zintgraf et al., 2017; Fong & Vedaldi, 2017; Lundberg & Lee, 2017; Plumb et al., 2018; Covert et al., 2021; Deng et al., 2021; Chen et al., 2022).

2.1. Shapley values

Unlike other attribution methods, the Shapley value is designed in game theory. Let us consider the following cooperative game, in which a set of n players $N = \{1, 2, \ldots, n\}$ collaborate with each other and finally win a reward R. The Shapley value (Shapley, 1953) is then developed as a fair approach to allocating the overall reward R to the n players. The Shapley value $\phi(i)$ is defined as the compositional reward allocated from R to the player $i \in N$, and we can consider $\phi(i)$ reflects the numerical contribution of the player i in the game.

Definition 1 (Shapley values)**.** *The Shapley value $\phi(i)$ of an input variable i is given by*

$$\phi(i) := \sum_{S \subseteq N \setminus \{i\}} \frac{|S|!(n-|S|-1)!}{n!} \left[V(S \cup \{i\}) - V(S) \right], \quad (1)$$

where $V : 2^N \mapsto \mathbb{R}$ denotes the reward function, i.e.*, $\forall S \subseteq N, V(S)$ measures the reward if a subset S of players participate in the game. Thus, $V(\emptyset) = 0$, and $V(N) = R$ denotes the overall reward won by all players in N.*

Faithfulness. The Shapley value is widely considered a relatively faithful attribution, because Young (1985) has proved that it is the unique game-theoretic solution that satisfies the *linearity*, *dummy*, *symmetry* and *efficiency* axioms for ideal attributions. Please see Appendix A for details.

In addition to estimating the importance of each input variable, the Shapley value is also widely used to estimate the importance of every single data point in a whole dataset, which can be, for instance, used to address data evaluation problem (Jia et al., 2019a; 2021).

2.2. Dilemma of computational complexity versus approximation accuracy

The biggest challenge of applying Shapley values to real-world applications is the NP-hard computational complexity. According to Equation (1) and Section 3, when we compute

Shapley values for input variables of a DNN, it requires to conduct inference on all the 2^n different masked samples. To alleviate the computational burden, many approximation methods (Castro et al., 2009; Lundberg & Lee, 2017; Covert & Lee, 2021) have been proposed. However, as Figure 3 shows, a higher approximation accuracy of Shapley values usually requires more network inferences (*e.g.*, inferences on as many as thousands of masked samples).

Specifically, some approaches estimated Shapley values via sampling techniques (Castro et al., 2009; Strumbelj & Kononenko, 2010; Okhrati & Lipani, 2021; Mitchell et al., 2022), and some converted the approximation of Shapley values to a weighted least squares problem (Lundberg & Lee, 2017; Simon & Vincent, 2020; Covert & Lee, 2021). However, these methods all faced a dilemma, *i.e.*, a more accurate approximation required higher computational costs.

Other studies accelerated the computation by assuming a specific distribution of data (Chen et al., 2019), ignoring small interactions between input variables (Wang et al., 2022), or learning an explainer model to directly predict the Shapley value (Jethani et al., 2021). However, these methods could not generate fully accurate Shapley values. Wang et al. (2021) proposed the ShapNets. The ShapNet was constrained to only encode interactions between a small number of variables (usually less than 4). When the ShapNet was extended to encode interactions between more variables, it could no longer estimate exactly accurate Shapley values. In comparison, our HarsanyiNet can accurately compute Shapley values in a single forward propagation, and it is not constrained to encode specific interactions, thereby exhibiting much more flexibility and better performance.

3. Methodology

The Shapley value defined in Equation (1) is widely used to estimate attributions of input variables in a DNN. We consider the DNN as a cooperative game, and consider input variables $\mathbf{x} = [x_1, x_2, ..., x_n]^\intercal$ as players, $N = \{1, 2, \ldots, n\}$. $v(\mathbf{x}) \in \mathbb{R}$ corresponds to the network prediction score[2] on $\mathbf{x}$. Let $\mathbf{x}_S$ denote a masked sample, where input variables in $N \setminus S, S \subseteq N$ are masked by baseline values[3]. In this way, we can define the total reward gained by the input variables in S as the inference score on the masked sample $\mathbf{x}_S$, *i.e.*, $V(S) := v(\mathbf{x}_S) - v(\mathbf{x}_\emptyset)$. Thus, the Shapley value $\phi(i)$ in Equation (1) measures the importance of the i-th input variable to the network prediction score.

3.1. Preliminaries: Harsanyi interactions

The Harsanyi interaction (or the Harsanyi dividend) (Harsanyi, 1963) provides a deeper insight into the essential reason why the Shapley value is computed as in Equation (1). A DNN usually does not use each individual input variable to conduct model inference independently. Instead, the DNN models the interactions between different input variables and considers such interactions as basic inference patterns. To this end, the Harsanyi interaction $I(S)$ measures the interactive effect between each subset $S \subseteq N$ of input variables, which is encoded by a DNN.

Definition 2 (Harsanyi interactions)**.** *The Harsanyi interaction between a set of variables in* S w.r.t. *the model output* $v(\mathbf{x})$ *is recursively defined* $I(S) := V(S) - \sum_{L \subsetneq S} I(L) = v(\mathbf{x}_S) - v(\mathbf{x}_\emptyset) - \sum_{L \subsetneq S} I(L)$ *subject to* $I(\emptyset) := 0$.

According to Definition 2, the network output can be explained as the sum of all Harsanyi interactions, *i.e.*, $v(\mathbf{x}) - v(\mathbf{x}_\emptyset) = \sum_{S \subseteq N} I(S)$. Essentially, each Harsanyi interaction $I(S)$ reveals an AND relationship between all the variables in set S. Let us consider a visual pattern $S = \{\mathit{eye}, \mathit{nose}, \mathit{mouth}\}$ for face detection as a toy example. If the image patches of *eye*, *nose*, and *mouth* appear together, the co-appearance of the three parts forms a visual pattern and makes a numerical contribution $I(S)$ to the classification score $v(\mathbf{x})$ of the face. Otherwise, masking any part in S will deactivate the pattern and remove the interactive effect, *i.e.*, making $I(S) = 0$.

Grabisch et al. (2016) and Ren et al. (2023a) further proved that the Harsanyi interaction also satisfies the four properties, namely *linearity*, *dummy*, *symmetry* and *efficiency*.

3.2. Harsanyi interactions compose Shapley values

We jump out of the dilemma of computational complexity versus approximation accuracy mentioned in Section 2. Theorem 1 allows us to derive a novel neural network architecture, namely *HarsanyiNet*, which uses Harsanyi interactions to simultaneously perform model inference and compute exact Shapley values in a single forward propagation.

• **Basic requirements for the HarsanyiNet.** The key idea of the HarsanyiNet is to let different intermediate-layer neurons to represent different Harsanyi interactions. Later, we will prove that we can use such a network design to compute the exact Shapley values in a single forward propagation. Specifically, let us introduce the following two designs in the HarsanyiNet.

Firstly, as shown in Figure 1, the HarsanyiNet has L cascaded blocks in the neural network. In each block, we add

[2] As in previous studies (Jethani et al., 2021; Wang et al., 2022), if the network has a scalar output, then $v(\mathbf{x})$ can be formulated directly as the network output. If the network has a vector output, *e.g.*, multi-category classification, we may define $v(\mathbf{x})$ as the output dimension corresponding to the ground-truth category.

[3] The baseline value can be set as zero, mean value over different inputs or other statistic values in previous studies (Lundberg & Lee, 2017; Covert & Lee, 2021).

an AND operation layer between a linear layer and a ReLU layer. Given an input sample $\mathbf{x}$, let $z_u^{(l)}(\mathbf{x})$ denote the u-th dimension of the output feature vector $\mathbf{z}^{(l)}(\mathbf{x}) \in \mathbb{R}^{m^{(l)}}$ in the l-th linear layer. The HarsanyiNet is designed to let each feature dimension $z_u^{(l)}(\mathbf{x})$ satisfy the following two requirements, and $z_u^{(l)}(\mathbf{x})$ is also called a *Harsanyi unit*. Theorem 3 will show how to use the Harsanyi unit to compute exact Shapley values directly.

Requirement 1 (**R1**). *The neural output $z_u^{(l)}(\mathbf{x})$ is exclusively determined by a specific set of input variables $\mathcal{R}_u^{(l)} \subseteq N$, namely the receptive field of neuron $z_u^{(l)}(\mathbf{x})$. In other words, none of the other input variables in $N \setminus \mathcal{R}_u^{(l)}$ affect the neuron activation,* i.e.*, given two arbitrary samples $\mathbf{x}$ and $\mathbf{x}'$, if $\forall\, i \in \mathcal{R}_u^{(l)}$, $x'_i = x_i$, then $z_u^{(l)}(\mathbf{x}') = z_u^{(l)}(\mathbf{x})$.*

Requirement 2 (**R2**). *Masking any variables in the receptive field $\mathcal{R}_u^{(l)}$ of the neuron $z_u^{(l)}$ will make $z_u^{(l)}(\mathbf{x}) = 0$. Specifically, let $\mathbf{x}_S$ denote the sample obtained by masking variables in the set $N \setminus S$ in the sample $\mathbf{x}$. Then, given any masked sample $\mathbf{x}_S$, the neuron $z_u^{(l)}$ must satisfy the property $z_u^{(l)}(\mathbf{x}_S) = z_u^{(l)}(\mathbf{x}) \cdot \prod_{i\in\mathcal{R}_u^{(l)}} \mathbb{1}(i \in S)$.*

These two requirements indicate that a Harsanyi unit $z_u^{(l)}$ must represent an AND relationship between input variables in $\mathcal{R}_u^{(l)}$. Changing variables outside the receptive field $\mathcal{R}_u^{(l)}$ will not affect the neural activation of the Harsanyi unit, *i.e.*, $z_u^{(l)}(\mathbf{x}_S) = z_u^{(l)}(\mathbf{x})$, but masking any variables in $\mathcal{R}_u^{(l)}$ will deactivate the unit, *i.e.*, making $z_u^{(l)}(\mathbf{x}_S) = 0$.

Secondly, although the HarsanyiNet may have various types of outputs (including a scalar output, a vectorized output, a matrix output, and a tensor output), **each dimension of the network output is designed as a weighted sum of all Harsanyi units**. Let $\mathbb{v}(\mathbf{x})$ denote the multi-dimensional output, and let $v(\mathbf{x})$ be an arbitrary output dimension parameterized by $\{\mathbf{w}_v^{(l)}\}$. Then, we get

$$\mathbb{v}(\mathbf{x}) = [v(\mathbf{x}), v'(\mathbf{x}), \dots]^\intercal, \quad v(\mathbf{x}) = \sum_{l=1}^{L} (\mathbf{w}_v^{(l)})^\intercal \mathbf{z}^{(l)}(\mathbf{x}), \quad (2)$$

where $\mathbf{w}_v^{(l)} \in \mathbb{R}^{m^{(l)}}$ denotes the weight for the specific output dimension v. As shown in Figure 1, the above equation can be implemented by adding skip connections to connect the Harsanyi units in all L layers to the HarsanyiNet output.

• **Proving that we can compute accurate Shapley values in a single forward propagation.** The preceding paragraphs only outline the two requirements for Harsanyi units, and Section 3.3 will introduce how to force neurons to meet such requirements. Before that, we derive Theorem 3 to prove that the above requirements allow us to compute the exact Shapley values in a forward propagation.

Theorem 1 (Connection between Shapley values and Harsanyi interactions, proof in (Harsanyi, 1963))**.** *The Shapley value $\phi(i)$ equals to the sum of evenly distributed Harsanyi interactions that contain i, i.e.,*

$$\phi(i) = \sum\nolimits_{S\subseteq N: S\ni i} \frac{1}{|S|} I(S). \quad (3)$$

Theorem 1 demonstrates that we can understand the Shapley value $\phi(i)$ as a uniform reassignment of each Harsanyi interaction $I(S)$ which includes the variable i. For example, let us consider a toy model that uses *age (a)*, *education (e)*, *occupation (o)*, and *marital status (m)* to infer the income level. We assume that we can only decompose four non-zero Harsanyi interactions, *i.e.*, $v(\mathbf{x}) = \sum_{S\subseteq N=\{a,e,o,m\}} I(S) = I(\{a,o\}) + I(\{a,e\}) + I(\{a,o,m\}) + I(\{o,m\})$ to simplify the story. We uniformly allocate the numerical contribution $I(\{a,o,m\})$ to variables *age*, *occupation*, and *marital status*, with each receiving $\frac{1}{3}I(\{a,o,r\})$ as a component of its attribution. In this way, each input variable accumulates compositional attributions from different Harsanyi interactions, *e.g.*, $\hat{\phi}(a) = \frac{1}{2}I(\{a,o\}) + \frac{1}{2}I(\{a,e\}) + \frac{1}{3}I(\{a,o,m\})$. Such an accumulated attribution $\hat{\phi}(a)$ equals to the Shapley value $\phi(a)$.

Lemma 1 (Harsanyi interaction of a Harsanyi unit, proof in Appendix C)**.** *Let us consider the output of a Harsanyi unit $z_u^{(l)}(\mathbf{x})$ as the reward. Then, let $J_u^{(l)}(S)$ denote the Harsanyi interaction* w.r.t. *the function $z_u^{(l)}(\mathbf{x})$. Then, we have $J_u^{(l)}(\mathcal{R}_u^{(l)}) = z_u^{(l)}(\mathbf{x})$, and $\forall S \neq \mathcal{R}_u^{(l)}$, $J_u^{(l)}(S) = 0$, according to the two requirements R1 and R2.*

Theorem 2 (Proof in Appendix B)**.** *Let a network output $v(\mathbf{x}) \in \mathbb{R}$ be represented as $v(\mathbf{x}) = \sum_{l=1}^{L} (\mathbf{w}_v^{(l)})^\intercal \mathbf{z}^{(l)}(\mathbf{x})$, according to Equation* (2)*. In this way, the Harsanyi interaction between input variables in the set S computed on the network output $v(\mathbf{x})$ can be represented as $I(S) = \sum_{l=1}^{L} \sum_{u=1}^{m^{(l)}} w_{v,u}^{(l)} J_u^{(l)}(S)$.*

Theorem 2 shows that the Harsanyi interaction $I(S)$ *w.r.t.* network output $v(\mathbf{x})$ can be represented as the sum of Harsanyi interactions $J_u^{(l)}(S)$ computed on different Harsanyi units $z_u^{(l)}(\mathbf{x})$. In this manner, we plug the conclusions in Lemma 1 and Theorem 2 into Equation (3), and we derive the following theorem.

Theorem 3 (**Deriving Shapley values from Harsanyi units in intermediate layers**, proof in Appendix B)**.** *The Shapley value $\phi(i)$ can be computed as*

$$\phi(i) = \sum\nolimits_{l=1}^{L} \sum\nolimits_{u=1}^{m^{(l)}} \frac{1}{|\mathcal{R}_u^{(l)}|} w_{v,u}^{(l)} z_u^{(l)}(\mathbf{x}) \mathbb{1}(\mathcal{R}_u^{(l)} \ni i). \quad (4)$$

Theorem 3 demonstrates that the Shapley value $\phi(i)$ can be directly computed using the outputs of Harsanyi units $\mathbf{z}^{(l)}(\mathbf{x})$ in the intermediate layers in forward propagation.

Cost of computing HarsanyiNet. We conduct one network inference to obtain the outputs of all Harsanyi units $z_u^{(l)}(\mathbf{x})$. Then, we compute Shapley values based on Equation (4), whose computational cost is $\mathcal{O}(nM)$, where $M = \sum_{l=1}^{L} m^{(l)}$ denotes the total number of Harsanyi units. The computational cost $\mathcal{O}(nM)$ is negligible, compared to the heavy computational cost of one forward propagation. **Therefore, we can roughly consider the overall cost of computing Shapley values as one forward propagation.**

3.3. Designing the HarsanyiNet towards R1 and R2

This subsection first introduces the detailed design of the HarsanyiNet. The basic idea is to construct a neural network in which each neuron represents an AND relationship between its children nodes in the previous layers, and the neuron's receptive field can be computed as the union of the receptive fields of its children nodes. Then, Theorem 4 proves that such a network design satisfies the requirements R1 and R2 in Section 3.2.

Harsanyi blocks. As Figure 1 shows, the HarsanyiNet contains L cascaded *Harsanyi blocks.* Specifically, we use tuple (l, u) to denote the u-th neuron in the l-th Harsanyi block's linear layer. Each neuron (l, u) has a set of children nodes $\mathcal{S}_u^{(l)}$. The children nodes in $\mathcal{S}_u^{(l)}$ can be selected from all neurons in all $(l-1)$ previous Harsanyi blocks[4]. Alternatively, we can just select children nodes from the $(l$–$1)$-th block, as a simplified implementation. The children nodes $\mathcal{S}_u^{(l)}$ which can be learned for each neuron (l, u) will be introduced later. Thus, given the children set $\mathcal{S}_u^{(l)}$, the neural activation $z_u^{(l)}(\mathbf{x})$ of the neuron (l, u) is computed by applying the linear, AND, and ReLU operations

$$g_u^{(l)}(\mathbf{x}) = (\mathbf{A}_u^{(l)})^{\intercal} \cdot \left(\mathbf{\Sigma}_u^{(l)} \cdot \mathbb{z}^{(l-1)} \right). \quad \text{//Linear operation on children nodes} \tag{5}$$

$$h_u^{(l)}(\mathbf{x}) = g_u^{(l)}(\mathbf{x}) \cdot \prod_{(l',u') \in \mathcal{S}_u^{(l)}} \mathbb{1}(z_{u'}^{(l')}(\mathbf{x}) \neq 0). \quad \text{//AND operation} \tag{6}$$

$$z_u^{(l)}(\mathbf{x}) = \text{ReLU}(h_u^{(l)}(\mathbf{x})). \quad \text{//Non-linear operation} \tag{7}$$

In the above equations, $\mathbb{z}^{(l-1)} = [\mathbf{z}^{(1)}(\mathbf{x})^{\intercal}, \mathbf{z}^{(2)}(\mathbf{x})^{\intercal}, \ldots, \mathbf{z}^{(l-1)}(\mathbf{x})^{\intercal}]^{\intercal} \in \mathbb{R}^{M^{(l)}}$ vectorizes the neurons in all the previous $(l-1)$ blocks. The children set $\mathcal{S}_u^{(l)}$ is implemented as a binary diagonal matrix $\mathbf{\Sigma}_u^{(l)} \in \{0,1\}^{M^{(l)} \times M^{(l)}}$, which selects children nodes of the neuron (l, u) from all $M^{(l)} = \sum_{l'=1}^{l-1} m^{(l')}$ neurons in all the $(l-1)$ previous blocks. $\mathbf{A}_u^{(l)} \in \mathbb{R}^{M^{(l)}}$ denotes the weight vector.

The three operations in Equations (5)–(7) ensure that each Harsanyi unit $z_u^{(l)}(\mathbf{x})$ represents an AND relationship among its children nodes (more discussions in Appendix D).

To implement the children selection in Equation (5), we compute the binary diagonal matrix $\mathbf{\Sigma}_u^{(l)}$ by setting $(\mathbf{\Sigma}_u^{(l)})_{i,i} = \mathbb{1}((\boldsymbol{\tau}_u^{(l)})_i > 0)$, where $\boldsymbol{\tau}_u^{(l)} \in \mathbb{R}^{M^{(l)}}$ is a trainable parameter vector. Note that during the training phase, the gradient of the loss function cannot pass through $\mathbf{\Sigma}_u^{(l)}$ to $\boldsymbol{\tau}_u^{(l)}$ in the above implementation; therefore, we employ Straight-Through Estimators (STE) (Bengio et al., 2013) to train the parameter $\boldsymbol{\tau}_u^{(l)}$. The STE uses $(\mathbf{\Sigma}_u^{(l)})_{i,i} = \mathbb{1}((\boldsymbol{\tau}_u^{(l)})_i > 0)$ in the forward-propagation and set $\partial(\mathbf{\Sigma}_u^{(l)})_{i,i} / \partial(\boldsymbol{\tau}_u^{(l)})_i = \beta e^{-(\boldsymbol{\tau}_u^{(l)})_i} / (1 + e^{-(\boldsymbol{\tau}_u^{(l)})_i})^2$ in the back-propagation process, where β is a positive scalar. Besides, to reduce the optimization difficulty of the AND operation in Equation (6), we approximate the AND operation as $h_u^{(l)}(\mathbf{x}) = g_u^{(l)}(\mathbf{x}) \left[\prod_{u'=1}^{M^{(l)}} (\mathbf{\Sigma}_u^{(l)} \cdot \tanh(\gamma \cdot \mathbf{\Sigma}_u^{(l)} \mathbb{z}^{(l-1)}) + (\mathbf{I} - \mathbf{\Sigma}_u^{(l)}) \cdot \mathbf{1})_{u'} \right]^{1/\mathbf{tr}(\mathbf{\Sigma}_u^{(l)})}$, where γ is a positive scalar. Here, each output dimension of the function $\tanh(\cdot)$ is within the range of $[0, 1)$, since $\mathbb{z}^{(l-1)}$ passes through the ReLU operation, and $\forall u, \mathbb{z}_u^{(l-1)} \geq 0$.

Input and receptive field. Let us set $\mathbf{z}^{(0)} = \mathbf{x} - \mathbf{b} \in \mathbb{R}^n$ as the input of the linear operation in the first Harsanyi block, and let us define the baseline value b_i as the masking state of each input variable x_i. To further simplify the implementation, we adopt a single value baseline $\mathbf{b}$[5]. It is worth noting that more sophisticated baseline values have been discussed in (Lundberg & Lee, 2017; Covert et al., 2020; Sundararajan & Najmi, 2020; Chen et al., 2022). Based on Equations (5)–(7), we can obtain that the receptive field $\mathcal{R}_u^{(l)}$ of a Harsanyi unit (l, u) can be computed recursively, as follows.

$$\mathcal{R}_u^{(l)} := \cup_{(l',u') \in \mathcal{S}_u^{(l)}} \mathcal{R}_{u'}^{(l')}, \quad \textit{s.t. } \mathcal{R}_u^{(1)} := \mathcal{S}_u^{(1)}. \tag{8}$$

Theorem 4 (proof in Appendix B)**.** *Based on Equations* (5)*–*(7)*, the receptive field $\mathcal{R}_u^{(l)}$ of the neuron $z_u^{(l)}$ automatically satisfies the two requirements R1 and R2.*

This theorem proves that setting each neuron $z_u^{(l)}$ based on Equations (5)–(7) can successfully encode an AND relationship between input variables in $\mathcal{R}_u^{(l)}$. In other words, only the input variables in the receptive field $\mathcal{R}_u^{(l)}$ can affect neural output $z_u^{(l)}(\mathbf{x})$, and masking any input variables in $\mathcal{R}_u^{(l)}$ will make $z_u^{(l)}(\mathbf{x}) = 0$. In particular, let us consider the inference on a masked sample $\mathbf{x}_S$ as an example. According to Theorem 4, the masked sample $\mathbf{x}_S$ is implemented by setting $\forall i \notin S, x_i = b_i$. Subsequently, this masked sample can exclusively activate all Harsanyi units subject to $\mathcal{R}_u^{(l)} \subseteq S$. All other Harsanyi units are not activated, *i.e.*, $\forall \mathcal{R}_u^{(l)} \nsubseteq S, z_u^{(l)}(\mathbf{x}_S) = 0$.

[4] In particular, children nodes in $\mathcal{S}_u^{(1)}$ are directly selected from the input variables.

[5] We simply set the baseline value $\mathbf{b} = \mathbf{0}$, since we use the ReLU function as the non-linear operation.

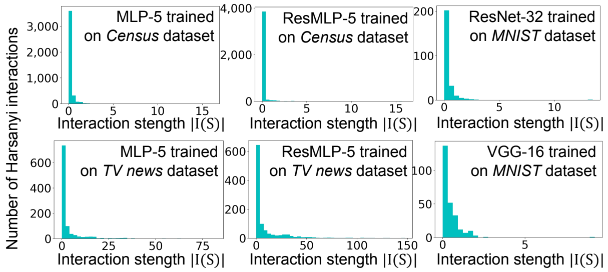

Figure 2. The histogram of interaction strength $|I(S)|$ of different Harsanyi interactions encoded by a DNN.

3.4. Discussion on the sparsity of Harsanyi interactions

Theoretically, a model can encode at most 2^n different Harsanyi interactions, but each AI model has its own limitation in encoding interactions, which are far less than 2^n. The HarsanyiNet learns at most $M = \sum_{l=1}^{L} m^{(l)}$ Harsanyi interactions, as proved in Lemma 1 and Theorem 2. Thus, the next question is whether the HarsanyiNet has sufficient representation capacity to handle real-world applications.

To this end, recent studies have observed (Deng et al., 2022; Ren et al., 2023a; Li & Zhang, 2023) and mathematically proved (Ren et al., 2023b) that traditional DNNs often encode only a few Harsanyi interactions in real-world applications, instead of learning all 2^n Harsanyi interactions. To be precise, the network output can be represented as

$$v(\mathbf{x}) = \sum\nolimits_{S \in 2^N} I(S) = \sum\nolimits_{S \in \Omega} I(S) + \epsilon, \qquad (9)$$

where $\Omega \subseteq 2^N = \{S' \subseteq N\}$ denotes a small set of Harsanyi interactions with considerable interaction strength $|I(S)|$. All other Harsanyi interactions have negligible interaction strength, *i.e.*, $|I(S)| \approx 0$, which can be considered noisy inference patterns. $\epsilon = \sum_{S' \in 2^N \setminus \Omega} I(S')$ is relatively small.

Furthermore, we conducted new experiments to verify the sparsity of Harsanyi interactions in DNNs. Given a trained network v and an input sample $\mathbf{x}$, we computed the interaction strength $|I(S)|$ of all 2^n Harsanyi interactions *w.r.t.*all $S \subseteq N$. We followed (Ren et al., 2023a) to compute the interaction strength $|I(S)|$ of different Harsanyi interactions[6]. Please see Appendix F.6 for more details. Figure 2 shows the extracted Harsanyi interactions. Such experiments were conducted on various DNNs, including MLP, the residual MLP[7] used in Touvron et al. (2022), the residual net with 32 layers (ResNet-32) (He et al., 2016) and the VGG net with 16 layers (VGG-16) (Simonyan et al., 2014), on the Census Income (Dua & Graff, 2017), the TV news commercial detection (Dua & Graff, 2017), and the MNIST (LeCun & Cortes, 2010) datasets. Only a few Harsanyi interactions were found to be salient. Most Harsanyi interactions were close to zero, and could be considered noise.

Therefore, the above experiments demonstrated that many applications only required DNNs to encode a few salient Harsanyi interactions, instead of modeling an exponential number of Harsanyi interactions. From this perspective, the HarsanyiNet has sufficient representation capacity.

4. Experiments

4.1. Two types of HarsanyiNets

In the experiments, we constructed and tested two types of HarsanyiNets following the paradigm in Equations (5)–(7), *i.e.*, the HarsanyiNet constructed with fully-connected layers, namely *Harsanyi-MLP*, and the HarsanyiNet constructed with convolutional layers, namely *Harsanyi-CNN*. The Harsanyi-MLP was suitable for handling tabular data, and the Harsanyi-CNN was designed for image data.

• **Harsanyi-MLP** was designed as an extension of the MLP network. As mentioned at the beginning of Section 3.3, we chose not to connect each neuron (l, u) from the neurons in all $(l-1)$ previous blocks. Instead, we simply selected a set of children nodes $\mathcal{S}_u^{(l)}$ from the $(l-1)$-th block. Specifically, this was implemented by fixing all elements in $\boldsymbol{\tau}_u^{(l)}$ corresponding to all neurons in the 1st, 2nd,. . . ,$(l-2)$-th blocks to 0.

• **Harsanyi-CNN** mainly used the following two specific settings to adapt convolutional layers into the paradigm of HarsanyiNet. **Setting 1.** Similar to the Harsanyi-MLP, the Harsanyi-CNN constructed the children set $\mathcal{S}_u^{(l)}$ of each neuron (l, u) from the neurons in the $(l-1)$-th block. Let $C \times K \times K$ denote the tensor size of the convolutional kernel. As Figure 6 shows, we selected children nodes $\mathcal{S}_u^{(l)}$ of the neuron $(l, u = (c, h, w))$ from neurons in the $C \times K \times K$ sub-tensor, which was clipped from the feature tensor of the $(l-1)$-th block and corresponded to the upper neuron (l, u), where c, h, w represent the location of the neuron (l, u). Accordingly, we had $\boldsymbol{\tau}_u^{(l)}$ as a CK^2-dimensional vector.

Setting 2. Furthermore, we set all neurons $(l, u = (:, h, w))$ at the same location, but on different channels, to share the same children set $\mathcal{S}_{u=(:,h,w)}^{(l)}$ to reduce the number of parameters $\boldsymbol{\tau}_u^{(l)}$. This could be implemented by letting all neurons $(l, u = (1, h, w)), \ldots, (l, u = (C, h, w))$ share the same parameter $\boldsymbol{\tau}_u^{(l)}$. Based on the above design, we proved that all Harsanyi units $(l, u = (c, h, w))$ in the same location (h, w) on different channels $(c = 1, \ldots, C)$ had the same receptive field $\mathcal{R}_{u=(:,h,w)}^{(l)}$ and contributed to the same Harsanyi interaction $I(S = \mathcal{R}_{u=(:,h,w)}^{(l)})$.

[6] For image data, Ren et al. (2023a) computed Harsanyi interactions between randomly sampled image regions to reduce the computational cost.

[7] We used 5-layer MLP (MLP-5) and 5-layer residual MLP (ResMLP-5) with 100 neurons in each hidden layer respectively.

Please see Appendix E for the proof. Therefore, we further considered neurons in the same location (h, w) on different channels as a single Harsanyi unit. In this way, Equation (6) could be rewritten as $h_u^{(l)}(\mathbf{x}) = g_u^{(l)}(\mathbf{x}) \cdot \prod_{(l-1,u')\in\mathcal{S}_u^{(l)}} \mathbb{1}(\sum_{c=1}^{C} |z_{u'=(c,h,w)}^{(l-1)}(\mathbf{x})| \neq 0)$.

In the implementation, we first applied a convolutional layer, max-pooling layer, and ReLU layer on the input image to obtain the feature $\mathbf{z}^{(0)}$ in an intermediate layer. Subsequently, we regarded $\mathbf{z}^{(0)}$ as the input of the Harsanyi-CNN, instead of directly using raw pixel values as input variables. It was because using the ReLU operation enabled us to simply define $b_i = 0$ as the baseline value (*i.e.*, the masking state) for all the feature dimensions $\mathbf{z}^{(0)}$. In addition, according to Setting 2, we could consider the feature vector $\mathbf{z}_{(:,h,w)}^{(0)}$ at location (h, w) as a single input variable, and we used $\mathbf{z}_{(:,h,w)}^{(0)} = \mathbf{0}$ to identify its masking state.

4.2. Experiments and comparison

Dataset. We trained the Harsanyi-MLP on three tabular datasets from the UCI machine learning repository (Dua & Graff, 2017), including the Census Income dataset ($n = 12$), the Yeast dataset ($n = 8$) and the TV news commercial detection dataset ($n = 10$), where n denotes the number of input variables. For simplicity, these datasets were termed *Census*, *Yeast*, and *TV news*. We trained the Harsanyi-CNN on two image datasets: the MNIST dataset (LeCun & Cortes, 2010) and the CIFAR-10 dataset (Krizhevsky et al., 2009).

Accuracy of Shapley values and computational cost. We conducted experiments to verify whether the HarsanyiNet could compute accurate Shapley values in a single forward propagation. We evaluated both the accuracy and the time cost of calculating Shapley values. We computed the root mean squared error (RMSE) between the estimated Shapley values $\boldsymbol{\phi}_{\mathbf{x}}$ and the true Shapley values, *i.e.*, RMSE=$\mathbb{E}_{\mathbf{x}}[\frac{1}{\sqrt{n}}||\boldsymbol{\phi}_{\mathbf{x}} - \boldsymbol{\phi}_{\mathbf{x}}^{*}||]$, where the vector of ground-truth Shapley values $\boldsymbol{\phi}_{\mathbf{x}}^{*}$ on the sample $\mathbf{x}$ could be directly computed by following Definition 1 when $n \leq 16$.

Comparing with approximation methods. We compared the accuracy of Shapley values computed by the HarsanyiNet with those estimated by various approximation methods, including the sampling method (Castro et al., 2009), KernelSHAP (Lundberg & Lee, 2017), KernelSHAP with paired sampling (KernelSHAP-PS) (Covert & Lee, 2021), antithetical sampling (Mitchell et al., 2022), DeepSHAP (Lundberg & Lee, 2017) and FastSHAP (Jethani et al., 2021). The approximation methods also computed Shapley values on the HarsanyiNet for fair comparison. Figure 3 shows that many approximation methods generated more accurate Shapley values, when they conducted more inferences for approximation. The number of inferences was widely used (Lundberg & Lee, 2017; Ancona et al., 2019) to quantify the computational cost of approximating Shapley values. These methods usually needed thousands of network inferences to compute the relatively accurate Shapley values. In comparison, the HarsanyiNet only needed one forward propagation to obtain the exact Shapley values (see "$\star$" in Figure 3). DeepSHAP and FastSHAP could compute the Shapley values in one forward propagation, but as shown in Table 2, the estimated errors of Shapley values were considerably larger than the HarsanyiNet.

Table 1. Root mean squared errors of the estimated Shapley value and the classification accuracy of the DNN.

	HarsanyiNet	Shallow ShapNet	Deep ShapNet[8]
MNIST dataset			
Classification accuracy (↑)	**99.16**	40.18	93.85
Errors of Shapley values (↓)	**1.19e-07**	3.79e-07	0.891
CIFAR-10 dataset			
Classification accuracy (↑)	**89.34**	20.48	73.51
Errors of Shapley values (↓)	**6.88e-08**	2.41e-07	0.409
Census dataset			
Classification accuracy (↑)	84.57	84.14	**84.72**
Errors of Shapley values (↓)	**2.18e-08**	5.12e-07	0.412
Yeast dataset			
Classification accuracy (↑)	**59.91**	57.17	59.70
Errors of Shapley values (↓)	**3.36e-08**	1.97e-07	0.127
TV news dataset			
Classification accuracy (↑)	82.20	79.72	**82.46**
Errors of Shapley values (↓)	**5.69e-08**	2.47e-07	0.239

Table 2. Error of the computed Shapley values on the Census, Yeast and TV news dataset.

Models Datasets	HarsanyiNet	DeepSHAP	FastSHAP
Census	**2.18e-08**	0.701	0.270
Yeast	**3.36e-08**	1.311	0.467
TV news	**5.69e-08**	0.758	0.526

Comparing with the ShapNets. Besides, we also compared the classification accuracy and the accuracy of Shapley values with two types of ShapNet (Wang et al., 2021), namely *Shallow ShapNet* and *Deep ShapNet*. Input samples in the MNIST and CIFAR-10 datasets contained many more input variables. To calculate the ground-truth Shapley values through Definition 1, we randomly sampled $n = 12$ variables as input variables in the foreground of the sample $\mathbf{x}$. In this way, ground-truth Shapley values were computed by masking the selected 12 variables and keeping all the other variables as original values of these variables. Simi-

[8] The results were obtained using the codes released by the original paper (Wang et al., 2021). In particular, for image datasets, each experiment was run for ten rounds with different random initialization, and the best result from the 10 runs was presented.

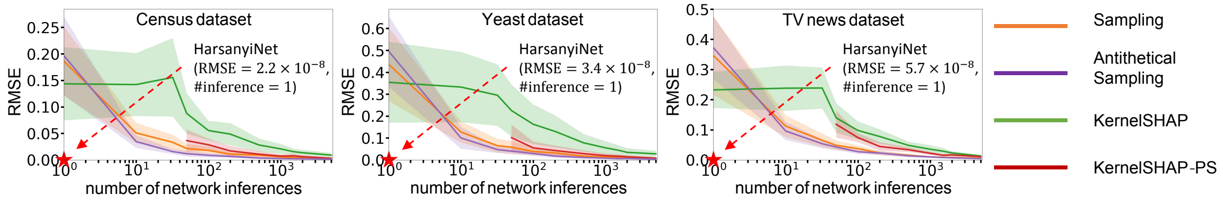

Figure 3. Comparison of estimation errors and the computational cost (number of network inferences) required by different methods.

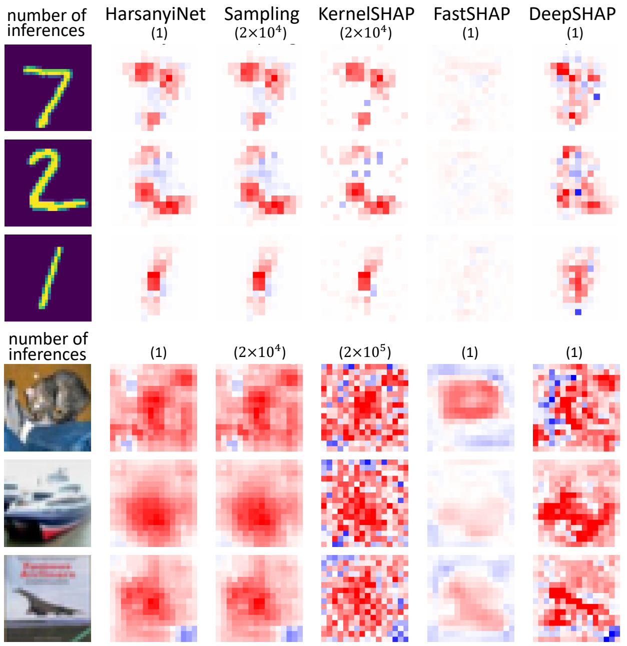

Figure 4. Shapley values computed[2] by different methods. The number of inferences conducted for approximation is also shown.

larly, we could still use the Harsanyi-CNN and ShapNets to derive the Shapley value when we only considered $n = 12$ input variables (please see Appendix F.8 for details).

Table 1 shows that both the HarsanyiNet and the Shallow ShapNet generated exact Shapley values with negligible errors, which were caused by unavoidable computational errors, but the HarsanyiNet had much higher classification accuracy than the Shallow ShapNet. This was because the representation capacity of the Shallow ShapNet was limited and could only encode interactions between a few input variables. On the other hand, the Deep ShapNet could not compute the exact Shapley values, although the Deep ShapNet achieved higher classification accuracy than the Shallow ShapNet. This was because the Deep ShapNet managed to encode interactions between more input variables, but the cost was that the Deep ShapNet could no longer theoretically guarantee the accuracy of the estimated Shapley values. Despite of this, the HarsanyiNet performed much better than the Deep ShapNet on more sophisticated tasks, such as image classification on the CIFAR-10 dataset.

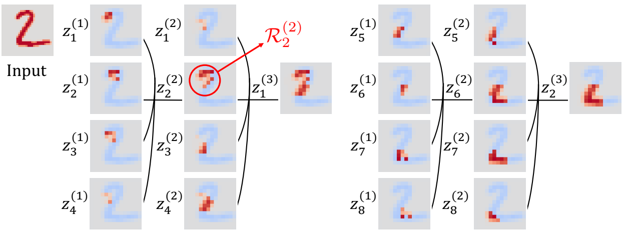

Figure 5. Visualization of the receptive field of Harsanyi units and the corresponding children nodes.

Visualization. We generated attribution maps based on the Shapley values estimated by each method on the MNIST and CIFAR-10 datasets. As Figure 4 shows[9], the attribution maps generated by the HarsanyiNet were almost the same as Shapley values, which were estimated by conducting inferences on 20000 sampled masked images and had converged to the true Shapley values. We also visualized the receptive fields of Harsanyi units on digit image in Figure 5. It verified that we could obtain the receptive field of a Harsanyi unit $z_u^{(l)}$ by merging receptive fields of its children nodes.

Implementation details. The Harsanyi-MLP was constructed with 3 cascaded Harsanyi blocks, where each was formulated by following Equations (5)–(7), and each Harsanyi block had 100 neurons. The Harsanyi-CNN was constructed with 10 cascaded Harsanyi blocks upon the feature $\mathbf{z}^{(0)}$, and each Harsanyi block had $512 \times 16 \times 16$ neurons, where 512 is the number of channels. The hyperparameters were set to $\beta = 10$ and $\gamma = 100$ for Harsanyi-MLP trained on tabular data, and set $\beta = 1000$ and $\gamma = 1$ for Harsanyi-CNN trained on the image data respectively. For the Harsanyi-MLP, we randomly selected 10 neurons in the previous layer as the initial children set $\mathcal{S}_u^{(l)}$, and set the corresponding dimensions in $\boldsymbol{\tau}_u^{(l)}$ to 1. For all other neu-

[9] To facilitate comparison with other methods, for the MNIST dataset, the Harsanyi-CNN was constructed with 4 cascaded Harsanyi blocks, and each Harsanyi block had $32 \times 14 \times 14$ neurons, where 32 is the number of channels. The hyperparameters were set to $\beta = 100$ and $\gamma = 0.05$, respectively. For the CIFAR-10 dataset, the Harsanyi-CNN was constructed with 10 cascaded Harsanyi blocks, and each Harsanyi block had $256 \times 16 \times 16$ neurons, where 256 is the number of channels. The hyperparameters were set to $\beta = 1000$ and $\gamma = 1$, respectively. Please see Appendix F.7 for more details.

rons in the previous layer, their corresponding dimensions in $\boldsymbol{\tau}_u^{(l)}$ were initialized to -1. For the Harsanyi-CNN, we initialized each parameter $(\boldsymbol{\tau}_u^{(l)})_i \sim \mathcal{N}(0, 0.01^2)$, which randomly selected about half of the neurons in the previous layer to satisfy $(\boldsymbol{\tau}_u^{(l)})_i > 0$ as the initial children set $\mathcal{S}_u^{(l)}$.

Discussion on evaluation metrics for attributions. Actually, many other metrics have been used to evaluate attribution methods, such as ROAR (Hooker et al., 2019) and weakly-supervised object localization (Zhou et al., 2016; Schulz et al., 2020). As Table 1 and Figure 3 show that the HarsanyiNet generated the fully accurate Shapley values, the evaluation of the attribution generated by the HarsanyiNet should be the same as the Shapley values, and the performance of Shapley values had been sophisticatedly analyzed in previous studies (Lundberg & Lee, 2017; Chen et al., 2019; Wang et al., 2021; Jethani et al., 2021). In particular, the Shapley value did not always perform the best in all evaluation metrics, although it was considered one of the most standard attribution methods and satisfied *linearity*, *dummy*, *symmetry*, and *efficiency* axioms.

5. Conclusion

In this paper, we have proposed the HarsanyiNet that can simultaneously perform model inference and compute the exact Shapley values of input variables in a single forward propagation. We have theoretically proved and experimentally verified the accuracy of Shapley values computed by the HarsanyiNet. Only negligible errors at the level of 10^{-8} – 10^{-7} were caused by unavoidable computational errors. Furthermore, we have demonstrated that the HarsanyiNet does not constrain the interactions between input variables, thereby exhibiting strong representation power.

Acknowledgements

This work is partially supported by the National Nature Science Foundation of China (62276165), National Key R&D Program of China (2021ZD0111602), Shanghai Natural Science Foundation (21JC1403800,21ZR1434600), National Nature Science Foundation of China (U19B2043), Shanghai Municipal Science and Technology Key Project (2021SHZDZX0102).

References

Ancona, M., Oztireli, C., and Gross, M. Explaining deep neural networks with a polynomial time algorithm for shapley value approximation. In *International Conference on Machine Learning*, pp. 272–281. PMLR, 2019.

Bach, S., Binder, A., Montavon, G., Klauschen, F., Müller, K.-R., and Samek, W. On pixel-wise explanations for non-linear classifier decisions by layer-wise relevance propagation. *PloS one*, 10(7):e0130140, 2015.

Bengio, Y., Léonard, N., and Courville, A. Estimating or propagating gradients through stochastic neurons for conditional computation. *arXiv preprint arXiv:1308.3432*, 2013.

Castro, J., Gómez, D., and Tejada, J. Polynomial calculation of the shapley value based on sampling. *Computers & Operations Research*, 36(5):1726–1730, 2009.

Chen, H., C. Covert, I., M. Lundberg, S., and Lee, S.-I. Algorithms to estimate shapley value feature attributions. *arXiv preprint arXiv:2207.07605*, 2022.

Chen, J., Song, L., Wainwright, M. J., and Jordan, M. I. L-shapley and c-shapley: Efficient model interpretation for structured data. *International Conference on Learning Representation*, 2019.

Covert, I. and Lee, S.-I. Improving kernelshap: Practical shapley value estimation using linear regression. In *International Conference on Artificial Intelligence and Statistics*, pp. 3457–3465. PMLR, 2021.

Covert, I., Lundberg, S., and Lee, S.-I. Explaining by removing: A unified framework for model explanation. *Journal of Machine Learning Research*, 2021.

Covert, I. C., Lundberg, S., and Lee, S.-I. Understanding global feature contributions with additive importance measures. *Advances in Neural Information Processing Systems*, 33, 2020.

Deng, H., Zou, N., Chen, W., Feng, G., Du, M., and Hu, X. Mutual information preserving back-propagation: Learn to invert for faithful attribution. In *Proceedings of the 27th ACM SIGKDD Conference on Knowledge Discovery & Data Mining*, pp. 258–268, 2021.

Deng, H., Ren, Q., Zhang, H., and Zhang, Q. Discovering and explaining the representation bottleneck of dnns. *International Conference on Learning Representation*, 2022.

Dua, D. and Graff, C. UCI machine learning repository, 2017. URL `http://archive.ics.uci.edu/ml`.

Fong, R. C. and Vedaldi, A. Interpretable explanations of black boxes by meaningful perturbation. In *Proceedings of the IEEE International Conference on Computer Vision*, pp. 3429–3437, 2017.

Goodfellow, I. J., Shlens, J., and Szegedy, C. Explaining and harnessing adversarial examples. In *International Conference on Learning Representation*, 2015.

Grabisch, M. et al. *Set functions, games and capacities in decision making*, volume 46. Springer, 2016.

Harsanyi, J. C. A simplified bargaining model for the n-person cooperative game. *International Economic Review*, 4(2):194–220, 1963.

He, K., Zhang, X., Ren, S., and Sun, J. Deep residual learning for image recognition. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 770–778, 2016.

Hooker, S., Erhan, D., Kindermans, P.-J., and Kim, B. A benchmark for interpretability methods in deep neural networks. *Advances in Neural Information Processing Systems*, 32, 2019.

Jethani, N., Sudarshan, M., Covert, I. C., Lee, S.-I., and Ranganath, R. Fastshap: Real-time shapley value estimation. In *International Conference on Learning Representations*, 2021.

Jia, R., Dao, D., Wang, B., Hubis, F. A., Gurel, N. M., Li, B., Zhang, C., Spanos, C. J., and Song, D. Efficient task-specific data valuation for nearest neighbor algorithms. In *International Conference on Very Large Databases*, 2019a.

Jia, R., Dao, D., Wang, B., Hubis, F. A., Hynes, N., Gurel, N. M., Li, B., Zhang, C., Song, D., and Spanos, C. Towards efficient data valuation based on the shapley value. In *International Conference on Artificial Intelligence and Statistics*, 2019b.

Jia, R., Wu, F., Sun, X., Xu, J., Dao, D., Kailkhura, B., Zhang, C., Li, B., and Song, D. Scalability vs. utility: Do we have to sacrifice one for the other in data importance quantification? In *IEEE Conference on Computer Vision and Pattern Recognition*, 2021.

Krizhevsky, A., Hinton, G., et al. Learning multiple layers of features from tiny images. 2009.

LeCun, Y. and Cortes, C. Mnist handwritten digit database, 2010. URL `http://yann.lecun.com/exdb/mnist/`.

Li, M. and Zhang, Q. Does a neural network really encode symbolic concepts? *International Conference on Machine Learning*, 2023.

Lundberg, S. M. and Lee, S.-I. A unified approach to interpreting model predictions. *Advances in Neural Information Processing Systems*, 30, 2017.

Madry, A., Makelov, A., Schmidt, L., Tsipras, D., and Vladu, A. Towards deep learning models resistant to adversarial attacks. In *International Conference on Learning Representation*, 2018.

Mitchell, R., Cooper, J., Frank, E., and Holmes, G. Sampling permutations for shapley value estimation. *Journal of Machine Learning Research*, 23:1–46, 2022.

Montavon, G., Lapuschkin, S., Binder, A., Samek, W., and Müller, K.-R. Explaining nonlinear classification decisions with deep taylor decomposition. *Pattern recognition*, 65:211–222, 2017.

Nilsback, M.-E. and Zisserman, A. Automated flower classification over a large number of classes. In *Indian Conference on Computer Vision, Graphics & Image Processing*, 2008.

Okhrati, R. and Lipani, A. A multilinear sampling algorithm to estimate shapley values. *International Conference on Pattern Recognition*, 2021.

Pavlova, M., Terhljan, N., G Chung, A., Zhao, A., Surana, S., Aboutalebi, H., Gunraj, H., Sabri, A., Alaref, A., and Wong, A. Covid-net cxr-2: An enhanced deep convolutional neural network design for detection of covid-19 cases from chest x-ray images. In *Front Med (Lausanne)*, 2022.

Plumb, G., Molitor, D., and Talwalkar, A. S. Model agnostic supervised local explanations. *Advances in Neural Information Processing Systems*, 31, 2018.

Ren, J., Li, M., Chen, Q., Deng, H., and Zhang, Q. Defining and quantifying the emergence of sparse concepts in dnns. *IEEE Conference on Computer Vision and Pattern Recognition*, 2023a.

Ren, Q., Gao, J., Shen, W., and Zhang, Q. Where we have arrived in proving the emergence of sparse symbolic concepts in ai models. *arXiv preprint arXiv:2305.01939*, 2023b.

Ribeiro, M. T., Singh, S., and Guestrin, C. ”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*, pp. 1135–1144, 2016.

Schulz, K., Sixt, L., Tombari, F., and Landgraf, T. Restricting the flow: Information bottlenecks for attribution. *International Conference on Learning Representation*, 2020.

Selvaraju, R. R., Cogswell, M., Das, A., Vedantam, R., Parikh, D., and Batra, D. Grad-cam: Visual explanations from deep networks via gradient-based localization. In *Proceedings of the IEEE International Conference on Computer Vision*, pp. 618–626, 2017.

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

Shrikumar, A., Greenside, P., Shcherbina, A., and Kundaje, A. Not just a black box: Learning important features through propagating activation differences. *arXiv preprint arXiv:1605.01713*, 2016.

Shrikumar, A., Greenside, P., and Kundaje, A. Learning important features through propagating activation differences. In *International Conference on Machine Learning*, pp. 3145–3153. PMLR, 2017.

Simon, G. and Vincent, T. A projected stochastic gradient algorithm for estimating shapley value applied in attribute importance. In *International Cross-Domain Conference on Machine Learning and Knowledge Extraction*, 2020.

Simonyan, K. and Zisserman, A. Very deep convolutional networks for large-scale image recognition. In *International Conference on Learning Representation*, 2015.

Simonyan, K., Vedaldi, A., and Zisserman, A. Deep inside convolutional networks: Visualising image classification models and saliency maps. *International Conference on Learning Representations*, 2014.

Springenberg, J. T., Dosovitskiy, A., Brox, T., and Riedmiller, M. Striving for simplicity: The all convolutional net. *International Conference on Learning Representations*, 2015.

Strumbelj, E. and Kononenko, I. An efficient explanation of individual classifications using game theory. *The Journal of Machine Learning Research*, 11:1–18, 2010.

Sundararajan, M. and Najmi, A. The many shapley values for model explanation. In *International Conference on Machine Learning*, pp. 9269–9278. PMLR, 2020.

Sundararajan, M., Taly, A., and Yan, Q. Axiomatic attribution for deep networks. In *International Conference on Machine Learning*, pp. 3319–3328. PMLR, 2017.

Touvron, H., Bojanowski, P., Caron, M., Cord, M., El-Nouby, A., Grave, E., Izacard, G., Joulin, A., Synnaeve, G., Verbeek, J., et al. Resmlp: Feedforward networks for image classification with data-efficient training. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2022.

Wang, G., Chuang, Y.-N., Du, M., Yang, F., Zhou, Q., Tripathi, P., Cai, X., and Hu, X. Accelerating shapley explanation via contributive cooperator selection. In *International Conference on Machine Learning*, pp. 22576–22590. PMLR, 2022.

Wang, L., Lin, Z. Q., and Wong, A. Covid-net: a tailored deep convolutional neural network design for detection of covid-19 cases from chest x-ray image. In *Scientific Reports*, 2020.

Wang, R., Wang, X., and Inouye, D. Shapley explanation networks. In *International Conference on Learning Representations*, 2021.

Weber, R. J. Probabilistic values for games. *The Shapley Value. Essays in Honor of Lloyd S. Shapley*, 101–119, 1988.

Wightman, R., Touvron, H., and Jégou, H. Resnet strikes back: An improved training procedure in timm. In *Neural Information Processing Systems*, 2021.

Young, H. P. Monotonic solutions of cooperative games. *International Journal of Game Theory*, 14:65–72, 1985.

Zhou, B., Khosla, A., Lapedriza, A., Oliva, A., and Torralba, A. Learning deep features for discriminative localization. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 2921–2929, 2016.

Zintgraf, L. M., Cohen, T. S., Adel, T., and Welling, M. Visualizing deep neural network decisions: Prediction difference analysis. *International Conference on Learning Representations*, 2017.

A. The Shapley values

In this section, we revisits the four axioms that the Shapley values satisfy, which ensures the Shapley values as relatively faithful attribution values. Let us consider the following cooperative game $V : 2^N \mapsto \mathbb{R}$, in which a set of n players $N = \{1, 2, \ldots, n\}$ collaborate and win a reward R. Here, $V(S)$ is equivalent to $v(\mathbf{x}_S) - v(\mathbf{x}_\emptyset)$ mentioned in the paper, and we have $V(\emptyset) = 0$. Young (1985) proved that the Shapley value was the unique solution which satisfied the four axioms, including the *linearity* axiom, *dummy* axiom, *symmetry* axiom and *efficiency* axiom (Weber, 1988) .

(1) *Linearity axiom:* If the game $V(\cdot)$ is a linear combination of two games $U(\cdot)$, $W(\cdot)$ for all $S \subseteq N$, *i.e.* $V(S) = U(S) + W(S)$ and $(c \cdot V)(S) = c \cdot V(S), \forall c \in \mathbb{R}$, then the Shapley value in the game V is also a linear combination of that in the games U and W , *i.e.* $\forall i \in N, \phi^V(i) = \phi^U(i) + \phi^W(i)$ and $\phi^{c \cdot V}(i) = c \cdot \phi^V(i)$.

(2) *Dummy axiom:* A player i is defined as a dummy player if $V(S \cup \{i\}) = V(S) + V(\{i\})$ for every $S \subseteq N \setminus \{i\}$. The dummy player i satisfies $\phi(i) = V(\{i\})$, which indicates player i influence the overall reward alone, without interacting/cooperating with other players in N.

(3) *Symmetry axiom:* For two players i and j, if $\forall S \subseteq N \setminus \{i, j\}$, $V(S \cup \{i\}) = V(S \cup \{j\})$, then the Shapley values of players i and j are equal, *i.e.* $\phi(i) = \phi(j)$.

(4) *Efficiency axiom:* The overall reward is equal to the sum of the Shapley value of each player, *i.e.* $\sum_{i=1}^{n} \phi(i) = V(N)$.

B. Proofs of Theorems

In this section, we prove the theorems in the paper.

Theorem 2. *Let a network output $v(\mathbf{x}) \in \mathbb{R}$ be represented as $v(\mathbf{x}) = \sum_{l=1}^{L} (\mathbf{w}_v^{(l)})^\intercal \mathbf{z}^{(l)}(\mathbf{x})$, according to Equation* (2)*. In this way, the Harsanyi interaction between input variables in the set S computed on the network output $v(\mathbf{x})$ can be represented as $I(S) = \sum_{l=1}^{L} \sum_{u=1}^{m^{(l)}} w_u^{(l)} J_u^{(l)}(S)$.*

Proof. We have

$$\begin{aligned} v(\mathbf{x}) &= \sum\nolimits_{l=1}^{L} (\mathbf{w}_v^{(l)})^\intercal \mathbf{z}^{(l)}(\mathbf{x}) \\ &= \sum\nolimits_{l=1}^{L} \sum\nolimits_{u=1}^{m^{(l)}} w_u^{(l)} z_u^{(l)}(\mathbf{x}). \end{aligned}$$

According to the linearity property of the Harsanyi interactions, if $\forall S \subseteq N$, $v(\mathbf{x}_S) = u(\mathbf{x}_S) + w(\mathbf{x}_S)$ and $(cv)(\mathbf{x}_S) = c \cdot v(\mathbf{x}_S), \forall c \in \mathbb{R}$, then the Harsanyi interaction $I^v(S)$ is also a linear combination of $I^u(S)$ and $I^w(S)$, *i.e.*, $\forall S \subseteq N$, $I^v(S) = I^u(S) + I^w(S)$ and $I^{(cv)}(S) = c \cdot I^v(S)$. Therefore, as $J_u^{(l)}(S)$ denotes the Harsanyi interaction computed on the function $z_u^{(l)}(\mathbf{x})$, we have the Harsanyi interaction computed on network output $v(\mathbf{x})$ a linear combination of $J_u^{(l)}(S)$, *i.e.*,

$$I(S) = \sum\nolimits_{l=1}^{L} \sum\nolimits_{u=1}^{m^{(l)}} w_u^{(l)} J_u^{(l)}(S).$$

□

Theorem 3 (Deriving Shapley values from Harsanyi units in intermediate layers)**.** *The Shapley value $\phi(i)$ can be computed as*

$$\phi(i) = \sum\nolimits_{l=1}^{L} \sum\nolimits_{u=1}^{m^{(l)}} \frac{1}{|\mathcal{R}_u^{(l)}|} w_u^{(l)} z_u^{(l)}(\mathbf{x}) \mathbb{1}(\mathcal{R}_u^{(l)} \ni i).$$

Proof. According to Theorem 1 and Theorem 2, we have

$$\begin{aligned}\phi(i) &= \textstyle\sum_{S\subseteq N: S\ni i} \frac{1}{|S|} I(S)\\ &= \textstyle\sum_{S\subseteq N} \frac{1}{|S|} I(S)\mathbb{1}(S \ni i)\\ &= \sum_{l=1}^{L}\sum_{u=1}^{m^{(l)}} \frac{1}{|\mathcal{R}_u^{(l)}|} w_u^{(l)} z_u^{(l)}(\mathbf{x})\mathbb{1}(\mathcal{R}_u^{(l)} \ni i).\end{aligned}$$

□

Theorem 4. *Based on Equations* (5)–(7)*, the receptive field $\mathcal{R}_u^{(l)}$ of the neuron $z_u^{(l)}$ automatically satisfies the Requirement 1 and 2. The receptive field $\mathcal{R}_u^{(l)}$ of a neuron (l, u) is defined recursively by $\mathcal{R}_u^{(l)} := \cup_{(l',u')\in\mathcal{S}_u^{(l)}} \mathcal{R}_{u'}^{(l')}$,* s.t. $\mathcal{R}_u^{(1)} := \mathcal{S}_u^{(1)}$.

Proof. **(1) Proof of the receptive field $\mathcal{R}_u^{(l)}$ of the neuron $z_u^{(l)}$ satisfies the Requirement 1**.

Given two arbitrary samples $\tilde{\mathbf{x}} = \tilde{\mathbf{z}}^{(0)}$ and $\mathbf{x} = \mathbf{z}^{(0)}$, to satisfy the Requirement 1, we will prove that if $\forall\, i \in \mathcal{R}_u^{(l)}, \tilde{\mathbf{x}}_i = \mathbf{x}_i$, then $z_u^{(l)}(\tilde{\mathbf{x}}) = z_u^{(l)}(\mathbf{x})$.

Firstly, for the first layer, $\forall u', \mathcal{R}_{u'}^{(1)} = \mathcal{S}_{u'}^{(1)} \subseteq \mathcal{R}_u^{(l)}$, we prove $z_{u'}^{(1)}(\tilde{\mathbf{x}}) = z_{u'}^{(1)}(\mathbf{x})$.

We get $g_{u'}^{(1)}(\tilde{\mathbf{x}}) = (\mathbf{A}_{u'}^{(1)})^{\intercal} \cdot \left(\mathbf{\Sigma}_{u'}^{(1)} \cdot \tilde{\mathbf{z}}^{(0)}\right) = (\mathbf{A}_{u'}^{(1)})^{\intercal} \cdot \boldsymbol{\zeta}$, where $\forall i \in \mathcal{R}_{u'}^{(1)} = \mathcal{S}_{u'}^{(1)} \subseteq \mathcal{R}_u^{(l)}, \boldsymbol{\zeta}_i = \tilde{\mathbf{z}}_i^{(0)}$, otherwise $\boldsymbol{\zeta}_i = 0$. We also get $g_{u'}^{(1)}(\mathbf{x}) = (\mathbf{A}_{u'}^{(1)})^{\intercal} \cdot \left(\mathbf{\Sigma}_{u'}^{(1)} \cdot \mathbf{z}^{(0)}\right) = (\mathbf{A}_{u'}^{(1)})^{\intercal} \cdot \boldsymbol{\eta}$, where $\forall i \in \mathcal{R}_{u'}^{(1)}, \boldsymbol{\eta}_i = \mathbf{z}_i^{(0)} = \tilde{\mathbf{z}}_i^{(0)}$, otherwise $\boldsymbol{\eta}_i = 0$.

Thus, $g_{u'}^{(1)}(\tilde{\mathbf{x}}) = g_{u'}^{(1)}(\mathbf{x})$ and $z_{u'}^{(1)}(\tilde{\mathbf{x}}) = z_{u'}^{(1)}(\mathbf{x})$.

Secondly, we prove $z_u^{(l)}(\tilde{\mathbf{x}}) = z_u^{(l)}(\mathbf{x})$ using the above conclusion.

For the second layer, $\forall u', \mathcal{R}_{u'}^{(2)} = \cup_{(1,u'')\in\mathcal{S}_{u'}^{(2)}} \mathcal{R}_{u''}^{(1)} \subseteq \mathcal{R}_u^{(l)}$, we can get $z_{u'}^{(2)}(\tilde{\mathbf{x}}) = z_{u'}^{(2)}(\mathbf{x})$ easily, since its children nodes is selected from $\forall u'', \mathcal{R}_{u''}^{(1)} = \mathcal{S}_{u''}^{(1)} \subseteq \mathcal{R}_u^{(l)}$, and the output of which satisfies $z_{u''}^{(1)}(\tilde{\mathbf{x}}) = z_{u''}^{(1)}(\mathbf{x})$. Similarly, we can derive $z_u^{(l)}(\tilde{\mathbf{x}}) = z_u^{(l)}(\mathbf{x})$ recursively.

In this way, we have proved that the receptive field $\mathcal{R}_u^{(l)}$ of the neuron $z_u^{(l)}$ satisfies the Requirement 1.

(2) Proof of the receptive field $\mathcal{R}_u^{(l)}$ of the neuron $z_u^{(l)}$ satisfies the Requirement 2.

Given a sample $\mathbf{x} = \mathbf{z}^{(0)}$ and its arbitrary masked sample $\mathbf{x}_S = \mathbf{z}_S^{(0)}$, to satisfy the Requirement 2, we will prove that $z_u^{(l)}(\mathbf{x}_S) = z_u^{(l)}(\mathbf{x}) \cdot \prod_{i\in\mathcal{R}_u^{(l)}} \mathbb{1}(i \in S)$. Specifically, we will prove that under the conditions of (1) $\forall S \supseteq \mathcal{R}_u^{(l)}$, (2) $\forall S \subsetneq \mathcal{R}_u^{(l)}$, or $\forall S, S \cup \mathcal{R}_u^{(l)} \neq S$ and $S \cup \mathcal{R}_u^{(l)} \neq \mathcal{R}_u^{(l)}$, we can get $z_u^{(l)}(\mathbf{x}_S) = z_u^{(l)}(\mathbf{x}) \cdot \prod_{i\in\mathcal{R}_u^{(l)}} \mathbb{1}(i \in S)$, respectively.

Firstly, we can easily get $\forall S \supseteq \mathcal{R}_u^{(l)}, z_u^{(l)}(\mathbf{x}_S) = z_u^{(l)}(\mathbf{x}) \cdot \prod_{i\in\mathcal{R}_u^{(l)}} \mathbb{1}(i \in S)$. Since $\forall i \in \mathcal{R}_u^{(l)}, (\mathbf{x}_S)_i = \mathbf{x}_i$, let us use the proven conclusion of (1) to derive $z_u^{(l)}(\mathbf{x}_S) = z_u^{(l)}(\mathbf{x}) = z_u^{(l)}(\mathbf{x}) \cdot \prod_{i\in\mathcal{R}_u^{(l)}} \mathbb{1}(i \in S)$.

Secondly, we prove that under the conditions of $\forall S \subsetneq \mathcal{R}_u^{(l)}$, or $\forall S, S \cup \mathcal{R}_u^{(l)} \neq S$ and $S \cup \mathcal{R}_u^{(l)} \neq \mathcal{R}_u^{(l)}$, we can get $z_u^{(l)}(\mathbf{x}_S) = z_u^{(l)}(\mathbf{x}) \cdot \prod_{i\in\mathcal{R}_u^{(l)}} \mathbb{1}(i \in S)$.

Let $\mathbf{x}_S$ denote the sample obtained by masking variables with $\mathbf{b}$ in the set $N \setminus S$ in the sample $\mathbf{x}$, then $\mathbf{z}^{(0)} = \mathbf{x} - \mathbf{b} \in \mathbb{R}^n$. In both settings, there exists at least a variable j that belongs to $\mathcal{R}_u^{(l)}$ but not to S, *i.e.*, $\exists j \in \mathcal{R}_u^{(l)}, j \notin S$, we have $(\mathbf{x}_S)_j = b$, $(\mathbf{z}_S^{(0)})_j = 0$ and $\prod_{i\in\mathcal{R}_u^{(l)}} \mathbb{1}(i \in S) = 0$.

For the first layer, there exists at least a neuron $(1, u')$ which satisfies $j \in \mathcal{R}_{u'}^{(1)} = \mathcal{S}_{u'}^{(1)} \subseteq \mathcal{R}_u^{(l)}$. Then $\forall u', h_{u'}^{(1)}(\mathbf{x}_S) = g_{u'}^{(1)}(\mathbf{x}_S) \cdot \prod_{(0,u'')\in\mathcal{S}_{u'}^{(1)}} \mathbb{1}(z_{u''}^{(0)}(\mathbf{x}_S) \neq 0) = g_{u'}^{(1)}(\mathbf{x}_S) \cdot \mathbb{1}((\mathbf{z}_S^{(0)})_j \neq 0) = 0$ and $z_{u'}^{(1)}(\mathbf{x}_S) = 0$. Since $j \in \mathcal{R}_u^{(l)} =$

$\cup_{(l',u'')\in\mathcal{S}_u^{(l)}}\mathcal{R}_{u''}^{(l')}$, there exists at least a neuron $(1,u')$ will affect the neuron (l,u) recursively, *i.e.*, $h_u^{(l)}(\mathbf{x}_S)=0$ and $z_u^{(l)}(\mathbf{x}_S)=0$. Thus, $z_u^{(l)}(\mathbf{x}_S)=z_u^{(l)}(\mathbf{x})\cdot\prod_{i\in\mathcal{R}_u^{(l)}}\mathbb{1}(i\in S)=0$.

In this way, we have proved that the receptive field $\mathcal{R}_u^{(l)}$ of the neuron $z_u^{(l)}$ satisfies the Requirement 2.

□

C. Proof of Lemma 1

Lemma 1 (Harsanyi interaction of a Harsanyi unit)**.** *Let us consider the output of a Harsanyi unit $z_u^{(l)}(\mathbf{x})$ as the reward. Then, let $J_u^{(l)}(S)$ denote the Harsanyi interaction* w.r.t. *the function $z_u^{(l)}(\mathbf{x})$. Then, we have $J_u^{(l)}(\mathcal{R}_u^{(l)})=z_u^{(l)}(\mathbf{x})$, and $\forall S\neq\mathcal{R}_u^{(l)}, J_u^{(l)}(S)=0$, according to Requirements 1 and 2.*

Proof. According to Definition 2, *i.e.*, $I(S)=v(S)-\sum_{L\subsetneq S}I(L)$ subject to $I(\emptyset):=0$, and Requirements 2, *i.e.*, $z_u^{(l)}(\mathbf{x}_S)=z_u^{(l)}(\mathbf{x})\cdot\prod_{i\in\mathcal{R}_u^{(l)}}\mathbb{1}(i\in S)$, the Harsanyi interaction of a Harsanyi unit can be written as,

$$\begin{aligned}J_u^{(l)}(S)&=z_u^{(l)}(\mathbf{x}_S)-\sum_{L\subsetneq S}J_u^{(l)}(L)\\&=z_u^{(l)}(\mathbf{x})\cdot\prod_{i\in\mathcal{R}_u^{(l)}}\mathbb{1}(i\in S)-\sum_{L\subsetneq S}J_u^{(l)}(L)\end{aligned}$$

(1) Proof of $J_u^{(l)}(\mathcal{R}_u^{(l)})=z_u^{(l)}(\mathbf{x})$.

Firstly, let us use the inductive method to prove $\forall L\subsetneq\mathcal{R}_u^{(l)}, J_u^{(l)}(L)=0$.

If $|L|=1,\forall L'\subseteq L\subsetneq\mathcal{R}_u^{(l)}$, we have $\prod_{i\in\mathcal{R}_u^{(l)}}\mathbb{1}(i\in L')=0$, then we get $J_u^{(l)}(L')=z_u^{(l)}(\mathbf{x})\cdot\prod_{i\in\mathcal{R}_u^{(l)}}\mathbb{1}(i\in L')=0$.

Assume that if $|L|=k,\forall L'\subseteq L\subsetneq\mathcal{R}_u^{(l)}$, we have $J_u^{(l)}(L')=0$.

Then if $|L|=k+1$, $\forall L'\subseteq L\subsetneq\mathcal{R}_u^{(l)}$, we have $\prod_{i\in\mathcal{R}_u^{(l)}}\mathbb{1}(i\in L)=0$ and $\forall L'\subsetneq L, J_u^{(l)}(L')=0$. Thus, we get $J_u^{(l)}(L)=z_u^{(l)}(\mathbf{x})\cdot\prod_{i\in\mathcal{R}_u^{(l)}}\mathbb{1}(i\in L)-\sum_{L'\subsetneq L}J_u^{(l)}(L')$ = 0.

In this way, we have proved that $\forall 1\leq|L|<|\mathcal{R}_u^{(l)}|,\forall L\subsetneq\mathcal{R}_u^{(l)}, J_u^{(l)}(L)=0$.

Secondly, let us use the proven conclusion $\forall L\subsetneq\mathcal{R}_u^{(l)}, J_u^{(l)}(L)=0$ to derive $J_u^{(l)}(\mathcal{R}_u^{(l)})=z_u^{(l)}(\mathbf{x})$.

Since $\prod_{i\in\mathcal{R}_u^{(l)}}\mathbb{1}(i\in\mathcal{R}_u^{(l)})=1$, we get $J_u^{(l)}(\mathcal{R}_u^{(l)})=z_u^{(l)}(\mathbf{x})\cdot\prod_{i\in\mathcal{R}_u^{(l)}}\mathbb{1}(i\in\mathcal{R}_u^{(l)})-\sum_{L\subsetneq\mathcal{R}_u^{(l)}}J_u^{(l)}(L)=z_u^{(l)}(\mathbf{x})$.

In this way, we have proved that $J_u^{(l)}(\mathcal{R}_u^{(l)})=z_u^{(l)}(\mathbf{x})$.

(2) Proof of $\forall S\neq\mathcal{R}_u^{(l)}, J_u^{(l)}(S)=0$.

To prove $\forall S\neq\mathcal{R}_u^{(l)}, J_u^{(l)}(S)=0$, we will prove that under the conditions of (1) $\forall S\subsetneq\mathcal{R}_u^{(l)}$, (2) $\forall S\supsetneq\mathcal{R}_u^{(l)}$, and (3) $\forall S, S\cup\mathcal{R}_u^{(l)}\neq S$ and $S\cup\mathcal{R}_u^{(l)}\neq\mathcal{R}_u^{(l)}$, we can get $J_u^{(l)}(S)=0$, respectively.

Firstly, we have proved that $\forall S\subsetneq\mathcal{R}_u^{(l)}, J_u^{(l)}(S)=0$.

Secondly, let us use the inductive method to prove $\forall S\supsetneq\mathcal{R}_u^{(l)}, J_u^{(l)}(S)=0$.

In this setting, $\prod_{i\in\mathcal{R}_u^{(l)}}\mathbb{1}(i\in S)=1$. If $|S|=|\mathcal{R}_u^{(l)}|+1,\forall S\supsetneq\mathcal{R}_u^{(l)}$, we have $J_u^{(l)}(S)=z_u^{(l)}(\mathbf{x})\cdot\prod_{i\in\mathcal{R}_u^{(l)}}\mathbb{1}(i\in S)-\sum_{L\subsetneq S}J_u^{(l)}(L)=z_u^{(l)}(\mathbf{x})-[J_u^{(l)}(\mathcal{R}_u^{(l)})+\sum_{L\subsetneq S,L\neq\mathcal{R}_u^{(l)}}J_u^{(l)}(L)]=0$. (Similarly, $\forall L\subsetneq S$ and $L\neq\mathcal{R}_u^{(l)}, J_u^{(l)}(L)=0$ can be proved by the inductive method.)

Assume that if $|S|=|\mathcal{R}_u^{(l)}|+k,\forall S\supsetneq\mathcal{R}_u^{(l)}$, we have $J_u^{(l)}(S)=0$.

Then if $|S|=|\mathcal{R}_u^{(l)}|+(k+1),\forall S\supsetneq\mathcal{R}_u^{(l)}$, we have $J_u^{(l)}(S)=0$.

Thirdly, let us use the inductive method to prove $\forall S, S \cup \mathcal{R}_u^{(l)} \neq S$ and $S \cup \mathcal{R}_u^{(l)} \neq \mathcal{R}_u^{(l)}$, $J_u^{(l)}(S) = 0$.

In this setting, $\prod_{i \in \mathcal{R}_u^{(l)}} \mathbb{1}(i \in S) = 0$. Then we have $J_u^{(l)}(S) = z_u^{(l)}(\mathbf{x}) \cdot \prod_{i \in \mathcal{R}_u^{(l)}} \mathbb{1}(i \in S) - \sum_{L \subsetneq S} J_u^{(l)}(L) = 0$. (Similarly, $\forall L \subsetneq S$, $J_u^{(l)}(S) = 0$ can be proved by the inductive method.)

In this way, we have proved that $\forall S \neq \mathcal{R}_u^{(l)}$, $J_u^{(l)}(S) = 0$.

□

D. Discussion on Equations (5)–(7)

Section 3.3 introduced that the neural activation $z_u^{(l)}(\mathbf{x})$ of the neuron (l, u) in a Harsanyi block was computed by applying a linear operation (Equation (5)), an AND operation (Equation (6)), and a ReLU operation (Equation (7)). We provide further discussions on the above three operations as follows.

Unlike a linear layer in a traditional DNN, Equation (5) shows that among neurons in all previous $(l-1)$ blocks, only outputs of the children nodes $\mathbf{\Sigma}_u^{(l)} \cdot \mathbb{z}^{(l-1)}$ can affect the output of the neuron (l, u). Equation (6) denotes that if all children nodes in $\mathcal{S}_u^{(l)}$ are activated, then the activation score $g_u^{(l)}(\mathbf{x})$ can pass through the AND operation, *i.e.*, $h_u^{(l)}(\mathbf{x}) = g_u^{(l)}(\mathbf{x})$. Otherwise, if any children node is not activated, *i.e.*, $\exists (l', u') \in \mathcal{S}_u^{(l)}$, *e.g.*, $z_{u'}^{(l')}(\mathbf{x}) = 0$, then we have $h_u^{(l)}(\mathbf{x}) = 0$.

E. Proofs and implementation details for Harsanyi-CNN

Harsanyi-CNN architecture. Figure 6 illustrates the architecture of the Harsanyi-CNN. As introduced in Section 4.1, we first applied a convolutional layer, max-pooling layer and ReLU layer to obtain the feature $\mathbf{z}^{(0)}$. Then, we built cascaded Harsanyi blocks on $\mathbf{z}^{(0)}$. Similar to the traditional CNN, each neuron $(l, u = (c, w, h))$ in the convolutional layer of each HarsanyiBlock corresponds to a subtensor $\mathbf{T}_u^{(l)} \in \mathbb{R}^{C \times K \times K}$ *w.r.t.* the previous layer, where C is the number of channels in the previous layer and $K \times K$ is the 2D kernel size. The neurons which share the same location but on different channels $(l, u = (:, w, h))$ correspond to the same subtensor $\mathbf{T}_u^{(l)}$. The children set $\mathcal{S}_u^{(l)}$ of each neuron (l, u) were selected from the subtensor $\mathbf{T}_u^{(l)}$. Moreover, neurons on the same location but on different channels $(l', u' = (:, w', h'))$ belong to the children set $\mathcal{S}_u^{(l)}$ simultaneously. The output of the HarsanyiBlock was constructed following Equations (5)–(7). Finally, each dimension of the network output $v(\mathbf{x})$ is constructed as the weighted sum of the output of each HarsanyiBlock using linear transformations and the skip-connection.

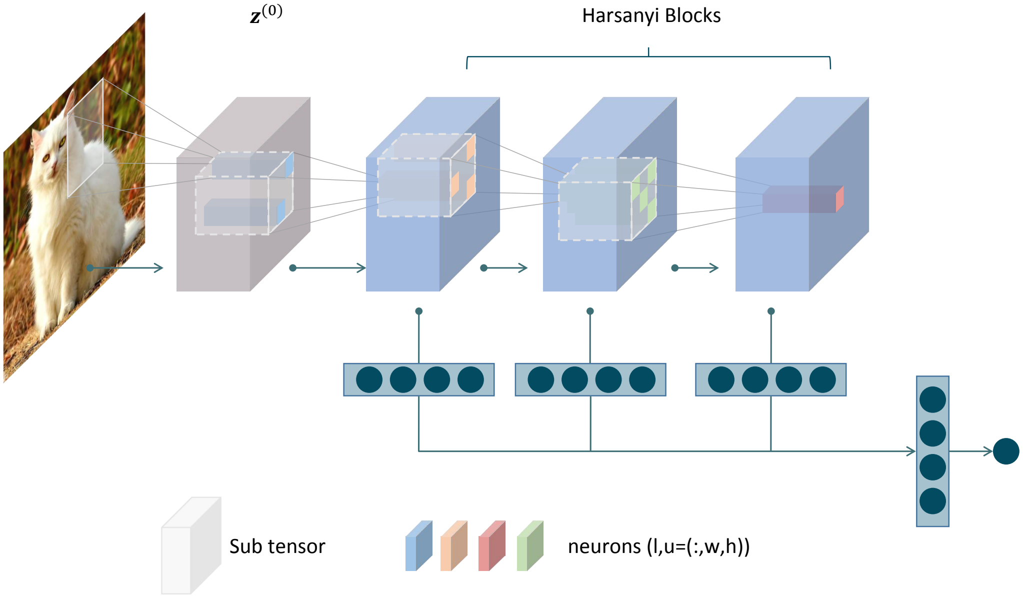

Figure 6. Schematic diagram of the Harsanyi-CNN architecture.

Proof of the conclusion in Setting 2 that *based on the design of letting all neurons* $(l, u = (1, h, w)), \dots, (l, u = (C, h, w))$ *share the same parameter* $\boldsymbol{\tau}_u^{(l)}$*, all Harsanyi units* $(l, u = (c, h, w))$ *in the same location* (h, w) *on different channels* $(c = 1, \dots, C)$ *had the same receptive field* $\mathcal{R}_{u=(:,h,w)}^{(l)}$ *and contributed to the same Harsanyi interaction* $I(S = \mathcal{R}_{u=(:,h,w)}^{(l)})$.

Proof. According to the implementation $\forall (l, u), \boldsymbol{\tau}_{u=(1,h,w)}^{(l)} = \boldsymbol{\tau}_{u=(2,h,w)}^{(l)} = \cdots = \boldsymbol{\tau}_{u=(C,h,w)}^{(l)} \in \mathbb{R}^{CK^2}$, we will prove that $\forall (l, u), \mathcal{R}_{u=(1,h,w)}^{(l)} = \mathcal{R}_{u=(2,h,w)}^{(l)} = \cdots = \mathcal{R}_{u=(C,h,w)}^{(l)}$.

Since $\forall (l, u), (\mathbf{\Sigma}_u^{(l)})_{i,i} = \mathbb{1}((\boldsymbol{\tau}_u^{(l)})_i > 0)$, then for arbitrary binary diagonal matrix $\mathbf{\Sigma}_u^{(l)}$, we have $\forall (l, u), \mathbf{\Sigma}_{u=(1,h,w)}^{(l)} = \mathbf{\Sigma}_{u=(2,h,w)}^{(l)} = \cdots = \mathbf{\Sigma}_{u=(C,h,w)}^{(l)}$. The children set $\mathcal{S}_u^{(l)}$ is implemented by $\mathbf{\Sigma}_u^{(l)}$, then we have $\forall (l, u), \mathcal{S}_{u=(1,h,w)}^{(l)} = \mathcal{S}_{u=(2,h,w)}^{(l)} = \cdots = \mathcal{S}_{u=(C,h,w)}^{(l)}$. According to Equation (8), we derive $\mathcal{R}_u^{(l)}$ from $\mathcal{S}_u^{(l)}$ recursively, then we have $\forall (l, u), \mathcal{R}_{u=(1,h,w)}^{(l)} = \mathcal{R}_{u=(2,h,w)}^{(l)} = \cdots = \mathcal{R}_{u=(C,h,w)}^{(l)}$. In this way, the Harsanyi units $(l, u = (c, h, w))$ in the same location (h, w) on different channels $(c = 1, \dots, C)$ had the same receptive field $\mathcal{R}_{u=(:,h,w)}^{(l)}$.

Next, we will show that considering C channels as C Harsanyi units, and considering C channels together as a single Harsanyi unit, their Harsanyi interactions are equal in both cases.

Considering C channels as C Harsanyi units, we have totally $m^{(l)} = H \times W \times C$ Harsanyi units in the l-th layer. We have $I(S = \mathcal{R}_{u=(1,h,w)}^{(l)}) = I(S = \mathcal{R}_{u=(2,h,w)}^{(l)}) = \cdots = I(S = \mathcal{R}_{u=(C,h,w)}^{(l)})$, which is abbreviated to $I(S = \mathcal{R}_u^{(l)})$. According to Theorem 2 and Lemma 1, we have

$$I(S = \mathcal{R}_u^{(l)}) = \sum\nolimits_{l=1}^{L} \sum\nolimits_{u=1}^{m^{(l)}} w_{v,u}^{(l)} J_u^{(l)}(S = \mathcal{R}_u^{(l)}) = \sum\nolimits_{l=1}^{L} \sum\nolimits_{u=1}^{H \times W \times C} w_{v,u}^{(l)} z_u^{(l)}(\mathbf{x})$$

where $w_{v,u}^{(l)} \in \mathbb{R}$ and $z_u^{(l)}(\mathbf{x}) \in \mathbb{R}$. Based on Equations (5)–(7), note that $\forall c \in \{1, 2, \dots, C\}, (l, u = (c, h, w))$ share the same children nodes $\mathcal{S}_{u=(1,h,w)}^{(l)} = \mathcal{S}_{u=(2,h,w)}^{(l)} = \cdots = \mathcal{S}_{u=(C,h,w)}^{(l)}$, then $h_{u=(c,h,w)}^{(l)}(\mathbf{x})$ at the same location on different channels is activated or deactivated at the same time, due to the AND operation on the child nodes. Besides, $z_u^{(l)}(\mathbf{x})$ is determined by the linear combination of the child nodes $g_u^{(l)}(\mathbf{x}) = (\mathbf{A}_u^{(l)})^\intercal \cdot \left(\mathbf{\Sigma}_u^{(l)} \cdot \mathbb{z}^{(l-1)}\right)$, where $\mathbf{A}_u^{(l)} \in \mathbb{R}^{CK^2}$ is the parameter of a convolution kernel (A total of C convolution kernels, denoted as $\mathbf{B}_u^{(l)} \in \mathbb{R}^{(CK^2) \times C}$, can derive C harsanyi units at the same position on different channels), $\mathbf{\Sigma}_u^{(l)} \in \mathbb{R}^{(CK^2) \times (CK^2)}$ denotes the selected children nodes and $\mathbb{z}^{(l-1)} \in \mathbb{R}^{CK^2}$ denotes the feature maps of the $(l-1)$-th layer within the coverage of the convolution kernel.

Considering C channels together as a single Harsanyi unit, we have totally $m^{(l)} = H \times W$ Harsanyi units in the l-th layer. We use $I(S = \mathcal{R}_{u=(:,h,w)}^{(l)})$ to denote this case. According to Theorem 2 and Lemma 1, we have

$$I(S = \mathcal{R}_{u=(:,h,w)}^{(l)}) = \sum\nolimits_{l=1}^{L} \sum\nolimits_{u=1}^{m^{(l)}} w_{v,u}^{(l)} J_u^{(l)}(S = \mathcal{R}_u^{(l)}) = \sum\nolimits_{l=1}^{L} \sum\nolimits_{u=1}^{H \times W} (\mathbf{w}_{v,u}^{(l)})^\intercal \mathbf{z}_u^{(l)}(\mathbf{x})$$

where $\mathbf{w}_{v,u}^{(l)} \in \mathbb{R}^C$ and $\mathbf{z}_u^{(l)}(\mathbf{x}) \in \mathbb{R}^C$. Based on Equations (5)–(7), note that $\forall c \in \{1, 2, \dots, C\}, \mathcal{S}_{u=(:,h,w)}^{(l)} = \mathcal{S}_{u=(c,h,w)}^{(l)}$, then the single C-dimensional Harsanyi unit has the same activation state as C Hassani units in above case. Besides, $\mathbf{z}_u^{(l)}(\mathbf{x})$ is determined by the linear combination of the child nodes $\mathbf{g}_u^{(l)}(\mathbf{x}) = (\mathbf{B}_u^{(l)})^\intercal \cdot \left(\mathbf{\Sigma}_u^{(l)} \cdot \mathbb{z}^{(l-1)}\right) \in \mathbb{R}^C$, where $\mathbf{B}_u^{(l)} \in \mathbb{R}^{(CK^2) \times C}$ is the parameters of a total of C convolution kernels, $\mathbf{\Sigma}_u^{(l)} \in \mathbb{R}^{(CK^2) \times (CK^2)}$ denotes the selected children nodes and $\mathbb{z}^{(l-1)} \in \mathbb{R}^{CK^2}$ denotes the feature maps of the $(l-1)$-th layer within the coverage of the convolution kernel.

In this way, we proved that the Harsanyi units $(l, u = (c, h, w))$ in the same location (h, w) on different channels $(c = 1, \dots, C)$ had the same receptive field $\mathcal{R}_{u=(:,h,w)}^{(l)}$ and contributed to the same Harsanyi interaction $I(S = \mathcal{R}_{u=(:,h,w)}^{(l)})$.

□

F. More experiment results and details

F.1. Experiment results of more challenging datasets on the HarsanyiNets

To further explore the classification performance of the HarsanyiNets, we conducted experiments on more challenging datasets, including the Oxford Flowers-102 (Nilsback & Zisserman, 2008) and COVIDx dataset (Wang et al., 2020). To

Table 3. Classification accuracy (%) of the HarsanyiNet and baseline models on more challenging datasets

Dataset	HarsanyiNet	baseline models
Oxford Flowers-102	95.48	97.9 (ResNet50 (Wightman et al., 2021))
COVIDx	96.75	96.3 (COVID-Net CXR-2 (Pavlova et al., 2022))

Table 4. Error between the Shapley values computed by the HarsanyiNet and the Shapley values estimated by the sampling method

Dataset	Errors of Shapley values (5000 iterations)	Errors of Shapley values (10000 iterations)
MNIST	0.017	0.012
CIFAR-10	0.007	0.004

compare the classification accuracy of the HarsanyiNet with a traditional DNN, we used ResNet-50 (He et al., 2016) and COVID-Net CXR-2 (Pavlova et al., 2022) as baseline models and reported the results in Table 3. Specifically, we used the intermediate-layer features with the size of $512 \times 14 \times 14$ from the pre-trained VGG-16 model (Simonyan & Zisserman, 2015) as $\mathbf{z}^{(0)}$, and then trained the HarsanyiNet upon $\mathbf{z}^{(0)}$ with the same hyperparameters as described in Section 4.

As shown in Table 3, the classification accuracy of the HarsanyiNet on the Oxford Flowers-102 is slightly lower than ResNet-50. However, on medical dataset COVIDx, the classification accuracy of the HarsanyiNet is slightly higher than COVID-Net CXR-2. Despite this relatively small sacrifice in classification accuracy on certain datasets, the HarsanyiNet computed the exact Shapley values in a single forward propogation, which was its main advantage over other neural networks.

F.2. Experiment results for verifying the accuracy of the Shapley values on the HarsanyiNets

To further verify the accuracy of the Shapley values on high-dimensional image datasets, we compared the Shapley values calculated by HarsanyiNet with those estimated by the sampling method. Specifically, we ran the sampling algorithm with 5000 and 10000 iterations on the MNIST dataset and the CIFAR-10 dataset, respectively. Table 4 shows the root mean square error (RMSE) between the Shapley values calculated by HarsanyiNet and the Shapley values estimated by the sampling algorithm. The estimation errors between both methods are quite small. Nevertheless, we need to emphasize that the sampling algorithm was more accurate when the sampling number was large, there was still a non-negligible error between the the estimated Shapely values and the ground-truth Shapley values.

F.3. Experiment results of the training cost of the HarsanyiNets

To further explore the training cost of the HarsanyiNets, we conducted experiments on the Census, MNIST, and CIFAR-10 datasets to evaluate the training cost of HarsanyiNets and traditional DNNs with comparable sizes. Table 5 shows that the computational cost of training the HarsanyiNet is higher than training a comparable DNN, and the computational cost of the HarsanyiNet is about twice the cost of a traditional DNN with the same number of parameters.

F.4. Experiment results of the robustness of the HarsanyiNets

We conducted more experiments to analyze the robustness of the HarsanyiNet. Specifically, we estimate the adversarial robustness of the classification performance and the adversarial robustness of the estimated Shapley values (Jia et al., 2019b).

To estimate the adversarial robustness of the classification performance on HarsanyiNet, we conducted experiments on the CIFAR-10 dataset to evaluate the model robustness by examining its classification accuracy on the test set of adversarial examples. To generate adversarial examples, we used the FGSM attack (Goodfellow et al., 2015), a gradient-based method, with a maximum perturbation of 8/255 (Madry et al., 2018). Table 6 shows that the classification accuracy of the adversarial examples of HarsanyiNet is slightly higher than that of ResNet-18 (He et al., 2016).

To estimate the adversarial robustness of the estimated Shapley values on HarsanyiNet, we assessed the robustness of its estimated Shapley values by computing the ℓ_2 norm of the difference in Shapley values between the adversarial and natural examples, *i.e.*, $||\boldsymbol{\phi}^{nat} - \boldsymbol{\phi}^{adv}||_{\ell_2}$, where $\boldsymbol{\phi}^{nat}$ denotes the Shapley values of natural examples, and $\boldsymbol{\phi}^{adv}$ denotes the Shapley values of adversarial examples. To calculate the Shapley values of the ResNet-18 model, we estimate Shapley values using the sampling algorithm with 1000 iterations. Table 6 shows that the adversarial robustness of the estimated Shapley values

Table 5. Training cost per epoch (s) of the HarsanyiNet and the comparable DNN

Dataset	HarsanyiNet	Comparable DNN
Census	5.0	1.9
MNIST	243.3	127.0
CIFAR-10	205.0	106.7

Table 6. Model robustness and Shapley value robustness

Model	Classification accuracy of adversarial examples (%)	ℓ_2 norm of the Shapley value difference
HarsanyiNet	13.83%	1.44
ResNet-18	8.21%	1.50

on HarsanyiNet (estimated by the ℓ_2 norm of the difference of Shapley values, the lower the better) is slightly higher than that of ResNet-18.

Both experiments indicate that HarsanyiNet has a robustness close to, or even slightly higher than, that of the traditional model.

F.5. More results of the estimated Shapley values on the HarsanyiNets

We conducted more experiments to show the explanations produced by our HarsanyiNets. Specifically, we trained the Harsanyi-MLP on tabular datasets and the Harsanyi-CNN on image datasets.

For the tabular datasets including the Census, Yeast and TV news datasets, we compared the estimated Shapley values for each method in Figure 7, Figure 8, and Figure 9, respectively. It can be seen that the Shapley values calculated by our HarsanyiNet were exactly the same as the ground-truth Shapley values calculated by Definition 1, while the approximation methods, including the sampling method (Castro et al., 2009), antithetical sampling (Mitchell et al., 2022), KernelSHAP (Lundberg & Lee, 2017), and KernelSHAP with paired sampling (KernelSHAP-PS) (Covert & Lee, 2021), needed thousands of network inferences to compute the relatively accurate Shapley values.

For the image datasets including the MNIST and CIFAR-10 datasets, we generated more attribution maps on different categories in Figure 10 and Figure 11, respectively.

F.6. Experiment details for computing interaction strength of Harsanyi interactions encoded by a DNN

When the number of input variables is small (*e.g.*, $n < 16$), we can iteratively calculate the interaction strength of all Harsanyi interactions following Definition 2. For tabular datasets, including the Census, Yeast and TV news datasets, we set the baseline value of each input variable the mean value of this variable over all training samples. Then we computed each Harsanyi interaction's strength in a brute-force manner.

For the MNIST dataset, it is impractical to directly compute the Harsanyi strength of all Harsanyi interactions. To reduce the computational cost, we randomly sampled 8 image regions in the foreground of each image following the previous work (Ren et al., 2023a). Then we were able to compute the Harsanyi effect of all possible Harsanyi interactions among the sampled 8 image regions (In total we obtained $2^8 = 256$ different Harsanyi interactions). In terms of the baseline value, we set, for each pixel, the baseline value to zero.

F.7. Experiment details for generating attribution maps in Figure 4

We compare the attribution maps of each method on the Harsanyi-CNN models. To facilitate comparison with other methods, the Harsanyi-CNN for the MNIST dataset was constructed with 4 cascaded Harsanyi blocks, and each Harsanyi block had $32 \times 14 \times 14$ neurons, where 32 is the number of channels. The hyperparameters were set to $\beta = 100$ and $\gamma = 0.05$ respectively. The Harsanyi-CNN for the CIFAR-10 dataset was constructed with 10 cascaded Harsanyi blocks, and each Harsanyi block had $256 \times 16 \times 16$ neurons, where 256 is the number of channels. The hyperparameters were set to $\beta = 1000$ and $\gamma = 1$ respectively. In this way, the HarsanyiNet and the other four model-agnostic methods used the same model to ensure the fairness of the comparison.

Besides, since the Harsanyi-CNN model calculated Shapley values on the feature $\mathbf{z}^{(0)}$, we also calculated Shapley values on

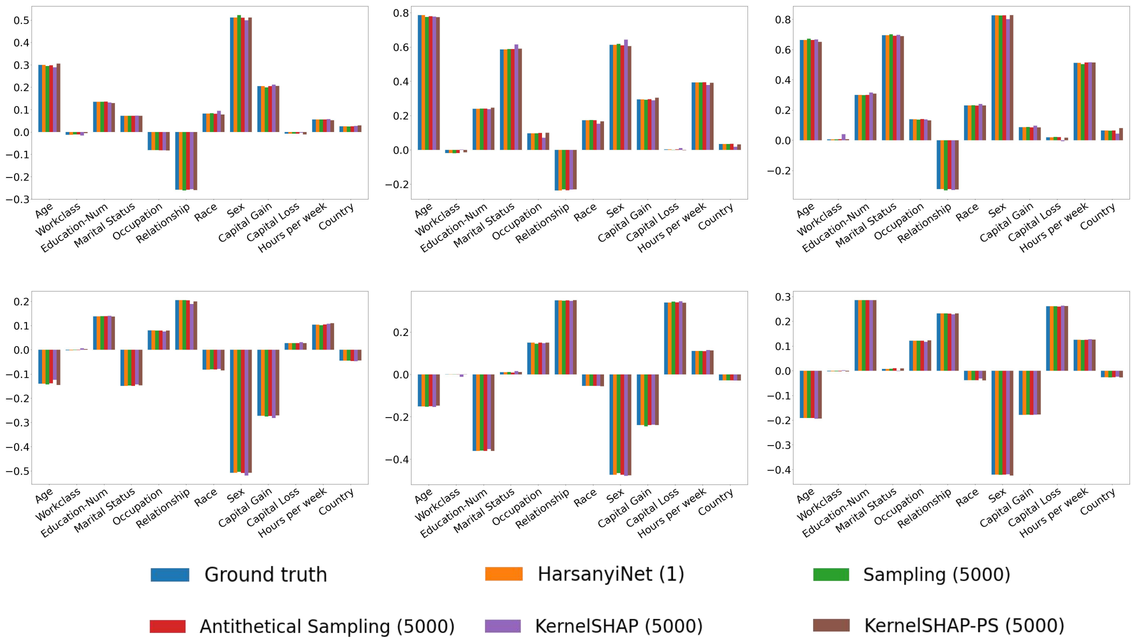

Figure 7. Shapley values computed by different methods on the Census dataset. The number of inferences conducted for each method is indicated in the brackets. The samples in the first row are from category '≤50K' and the samples in the second row are from the category '>50K'.

$\mathbf{z}^{(0)}$ using the sampling, KernelSHAP, and DeepSHAP methods. For the MNIST dataset, we run about 20000 iterations of the sampling method and 20000 iterations of the KernelSHAP method until convergence. For the CIFAR-10 dataset, we run about 20000 iterations of the sampling method and 200000 iterations of the KernelSHAP method until convergence.

For the FastSHAP method, we used the training samples $\mathbf{x}$ and the model predictions of the Harsanyi-CNN to train a explainer model ϕ_{fast}, and slightly modified the model architecture to return the attribution maps with a tensor of the same size as the size of $\mathbf{z}^{(0)}$, *i.e.*, 14×14 for the MNIST dataset and 16×16 for the CIFAR-10 dataset. For the MNIST dataset, since (Jethani et al., 2021) did not report the explainer model ϕ_{fast}, we trained a explainer model with the same structure as which the CIFAR-10 dataset used, and computed the attribution maps on the MNIST dataset.

F.8. Experiment details for comparing computed Shapley values with true Shapley values on the MNIST and CIFAR-10 datasets

As mentioned in Section 4.2, in order to reduce the computational cost, we randomly sampled $n = 12$ variables in the foreground of the sample as input variables on image datasets. In this way, ground-truth Shapley values were computed by masking the selected 12 variables and keeping all the other variables as the original variables of the sample. Let us denote the set of the selected variables as $\hat{N}$, thus, $|\hat{N}| = 12$. Specifically, we set all the variables $x_i, i \notin \hat{N}$ as the baseline value, *i.e.*, $\forall i \notin \hat{N}, b_i = x_i$ and $\forall i \in \hat{N}, b_i = 0$, to obtain a baseline sample $v(\mathbf{x}_\emptyset)$. Based on the baseline sample $v(\mathbf{x}_\emptyset)$, we obtained $2^{|\hat{N}|}$ different masked samples. For the HarsanyiNet, when we computed the Shapley values for the selected variables based on Theorem 1, we only visited the sets that contain the selected variables, *i.e.*, $S \ni i, \forall i \in \hat{N}$. Besides, $|S|$ denoted the number of the selected variables in S. In this way, we computed the Shapley values for the selected variables by $\phi(i) = \sum_{S \subseteq N: S \ni i, i \in \hat{N}} \frac{1}{|S|} I(S)$ and $\sum_{i=1}^{n} \phi(i) = v(\mathbf{x}_N = \mathbf{x}) - v(\mathbf{x}_\emptyset)$. For the ShapNets, we set all the variables $x_i, i \notin \hat{N}$ as the baseline value $b_i = 0$ to obtain a masked sample $\mathbf{x}'$, *i.e.*, $x'_i = x_i, \forall i \in \hat{N}$; $x'_i = 0$, otherwise. Then, with the masked input sample, we could compute the Shapley values for the selected variables with the ShapNet.

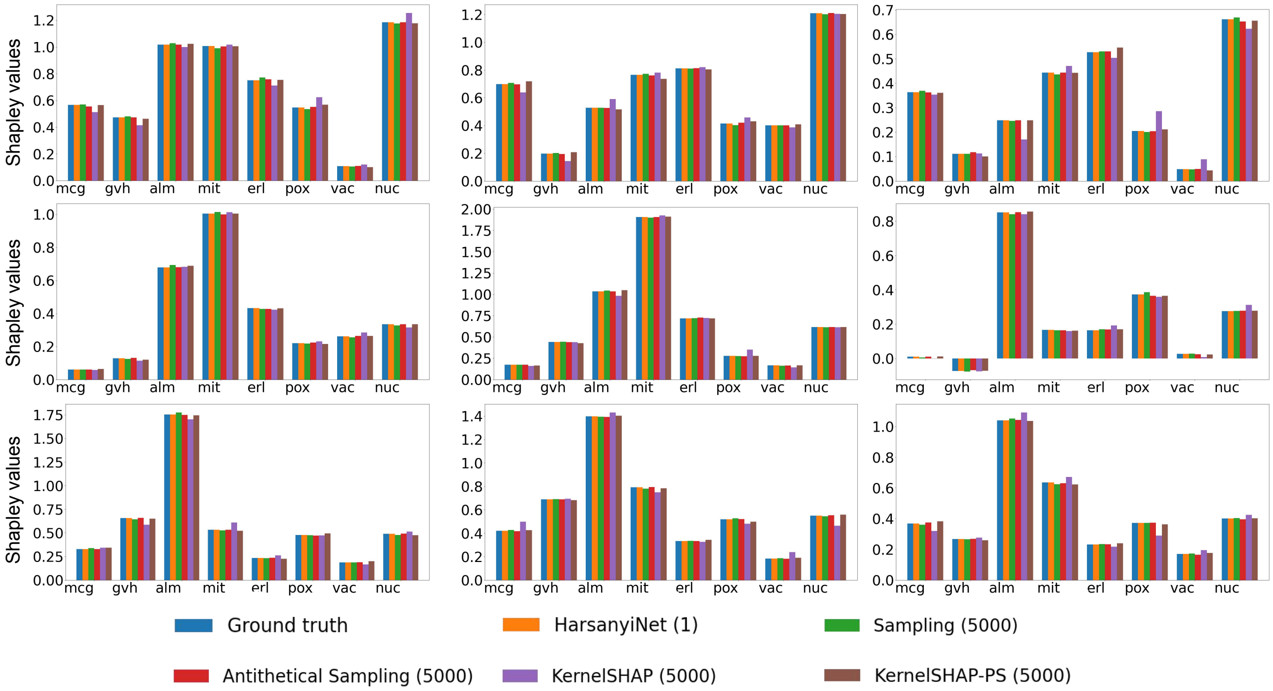

Figure 8. Shapley values computed by different methods on the Yeast dataset. The number of inferences conducted for each method is indicated in the brackets. The Shapley values calculated on samples from 3 categories (out of 10 categories) are shown. Samples in the first row are from category ‘CYT’, samples in the second row are from category ‘MIT’, and samples in the last row are from category ‘NUC’.

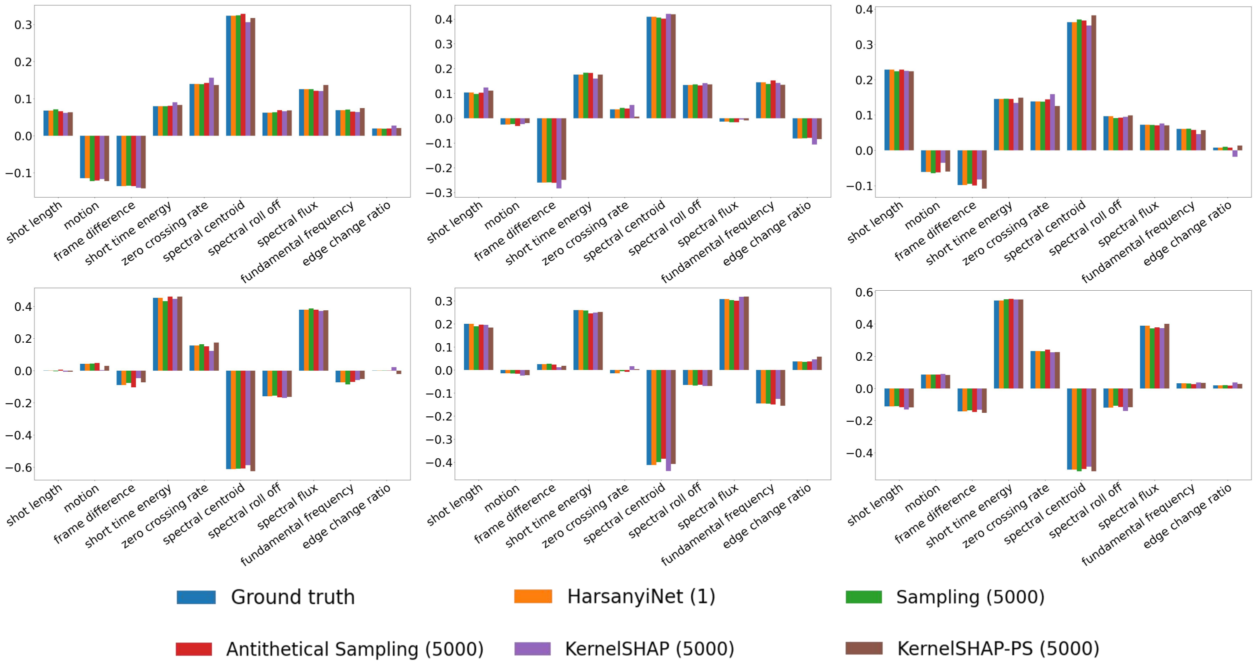

Figure 9. Shapley values computed by different methods on the TV News dataset. The number of inferences conducted for each method is indicated in the brackets. The samples in the first row are from category ‘Non Commercials’ and the samples in the second row are from the category ‘Commercials’.

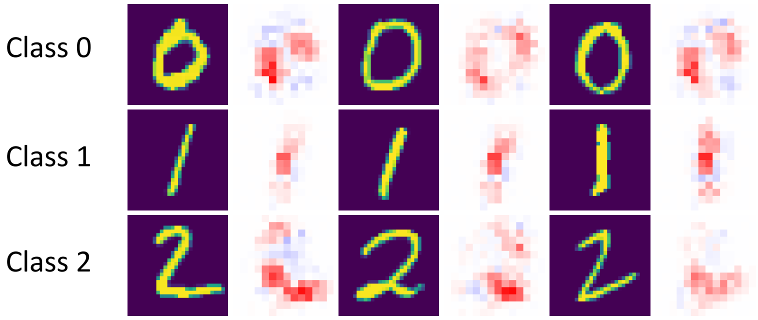

Figure 10. Shapley values produced by the HarsanyiNet on the MNIST dataset. The Shapley value is computed by setting $v(\mathbf{x}_S)$ as the output dimension of the ground-truth category of the input sample $\mathbf{x}$.

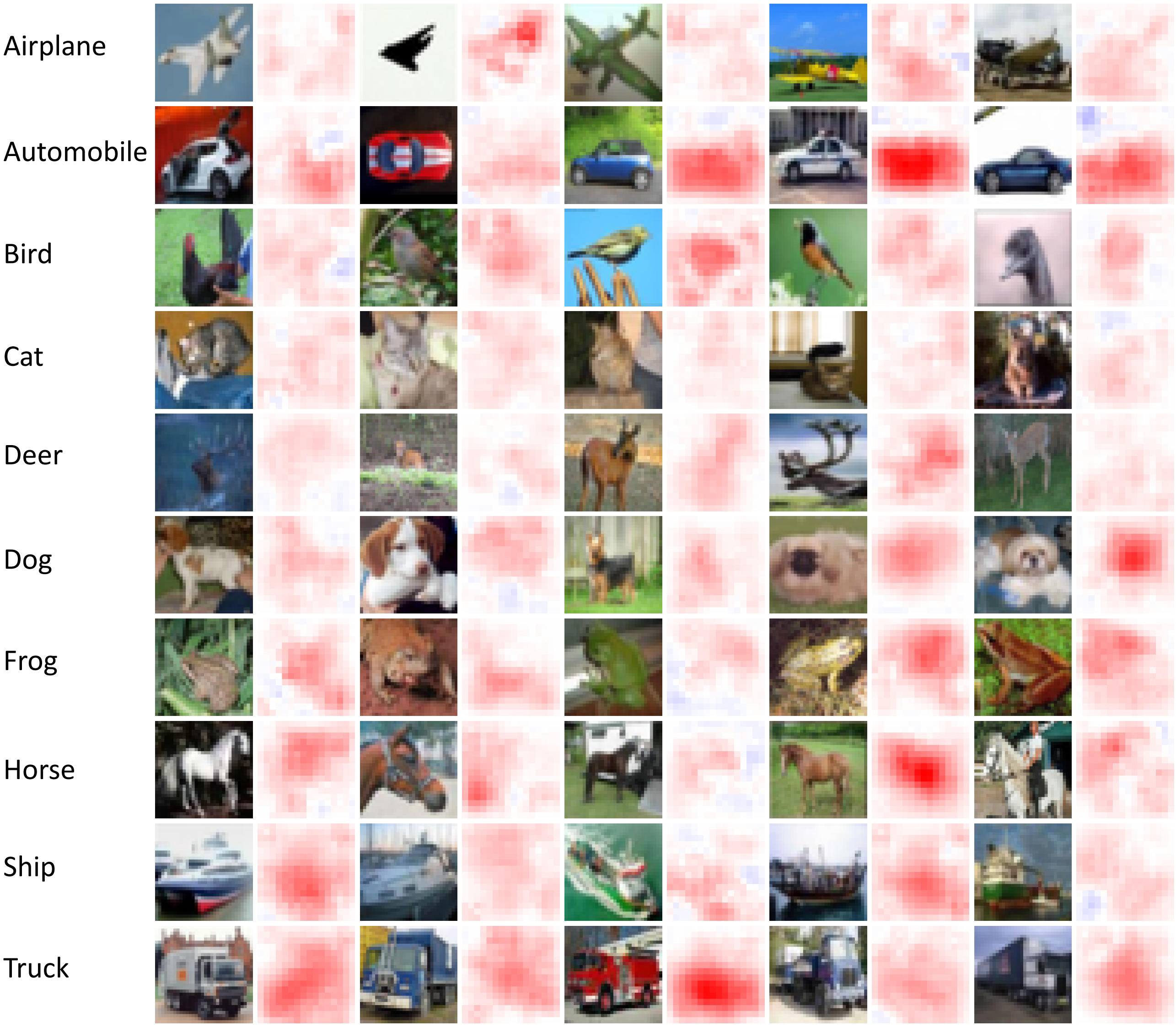

Figure 11. Shapley values produced by the HarsanyiNet on the CIFAR-10 dataset. The Shapley value is computed by setting $v(\mathbf{x}_S)$ as the output dimension of the ground-truth category of the input sample $\mathbf{x}$.